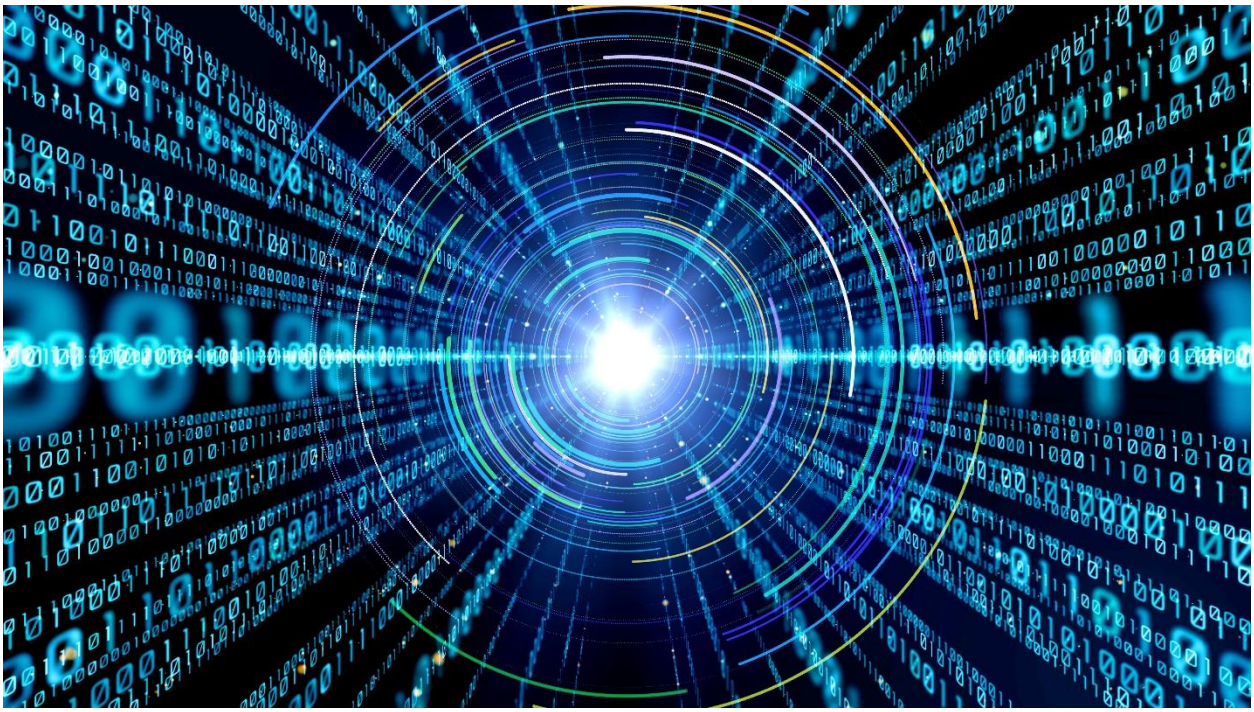
The Proceedings of the Fourth

IEEE International Conference on Rebooting Computing (ICRC 2019)



San Mateo, California, USA
November 6-8, 2019

IEEE Catalog Number: CFP19G30-ART
ISBN: 978-1-7281-5221-9

2019 IEEE International Conference on Rebooting Computing (ICRC)

IEEE Catalog Number: CFP19G30-ART *(Article)*
ISBN: 978-1-7281-5221-9 *(Article)*

Printed copies of this publication are available from:

Curran Associates, Inc
57 Morehouse Lane
Red Hook, NY 12571 USA
Phone: (845) 758-0400
Fax: (845) 758-2633
E-mail: curran@proceedings.com

IEEE eXpress Conference Publishing

Produced by IEEE eXpress Conference Publishing

For information on producing a conference proceedings and receiving an estimate, contact conferencepublishing@ieee.org
http://www.ieee.org/conferencepublishing

About ICRC 2019

Now in its fourth year, the IEEE International Conference on Rebooting Computing (ICRC) is the premier venue for novel computing approaches, including algorithms and languages, system software, system and network architectures, new devices and circuits, and applications of new materials and physics. This is an interdisciplinary conference that has participation from a broad technical community, with emphasis on all aspects of the computing stack.

IEEE ICRC 2019 is part of the IEEE Rebooting Computing Week. ICRC 2019 will be held November 6-8, 2019 at the San Mateo Marriott in San Matteo, California.

ICRC 2019 Committees

General Co-chairs: Cullen Bash, *Hewlett Packard Enterprise* and
Vivek Sarkar, *Georgia Tech*
Technical Program Co-chairs: James Ang, *DOE, PNNL*
Paolo Faraboschi, *Hewlett Packard Enterprise*

2019 Technical Program Committee:

- Rosa Badia, *BSC*
- Daniel Brunner, *FEMTO-ST/CNRS*
- Anastasiia Butko, *DOE, LBNL*
- Greg Byrd, *NCSU*
- John Daly, *DoD*
- Marc Duranton, *CEA*
- Wendy Elsasser, *ARM*
- Maya Gokhale, *DOE, LLNL*
- Christopher Granade, *Microsoft*
- Wen-mei Hwu, *UIUC*
- Todd Hylton, *UCSD*
- Giacomo Indiveri, *ETH*
- Ali Javadi, *IBM*
- Ulya Karpuzcu, *Univ. Minnesota*
- Sriram Krishnamoorthy, *DOE, PNNL*
- Katie Lewis, *DOE, LLNL*
- Srilatha (Bobbie) Manne, *Microsoft*
- Satoshi Matsuoka, *RIKEN*
- Avi Mendelson, *Technion*
- David Mountain, *DoD/LPS*
- Banu Nagasundaram, *Intel*
- Robert Patti, *NHanced Semiconductors*
- Arun Rodrigues, *DOE, SNL*
- Catherine Schuman, *DOE, ORNL*
- John Shalf, *DOE, LBNL*
- Antonino Tumeo, *DOE, PNNL*
- Didem Unat, *Koç Üniversitesi*
- Thomas Van Vaerenbergh, *Hewlett Packard Enterprise*
- William Vanderlinde, *DOE-SC, ASCR*
- Jeffrey Vetter, *DOE, ORNL*
- Gwendolyn Voskuilen, *DOE, SNL*
- Cliff Young, *Google*

Secretary: Terence Martinez, *IEEE Technical Activities*
Co-Treasurers: Elie Track, *nVizix*
Marco Fiorentino, *Hewlett Packard Enterprise*
Patronage Co-Chairs: Kirk Bresniker, *Hewlett Packard Enterprise*
Martina Trucco, *Micron*
Local Arrangements Chair: Cat Graves, *Hewlett Packard Enterprise*
Publications Chair: Erik Blair, *Baylor University*

2019 Technical Reviewers

- Rosa Badia, *BSC*
- Daniel Brunner, *FEMTO-ST/CNRS*
- Anastasiia Butko, *DOE, LBNL*
- Greg Byrd, *NCSU*
- John Daly, *DoD*
- Marc Duranton, *CEA*
- Wendy Elsasser, *ARM*
- Maya Gokhale, *DOE, LLNL*
- Christopher Granade, *Microsoft*
- Wen-mei Hwu, *UIUC*
- Todd Hylton, *UCSD*
- Giacomo Indiveri, *ETH*
- Ali Javadi, *IBM*
- Ulya Karpuzcu, *Univ. Minnesota*
- Sriram Krishnamoorthy, *DOE, PNNL*
- Katie Lewis, *DOE, LLNL*
- Srilatha (Bobbie) Manne, *Microsoft*
- Satoshi Matsuoka, *RIKEN*
- Avi Mendelson, *Technion*
- David Mountain, *DoD/LPS*
- Banu Nagasundaram, *Intel*
- Robert Patti, *NHanced Semionductors*
- Melika Payvand, *Univ. of Zurich*
- Nicoletta Risi, *Univ. of Zurich*
- Arun Rodrigues, *DOE, SNL*
- Catherine Schuman, *DOE, ORNL*
- Ashkan Seyedi, *Hewlett Packard Enterprise*
- John Shalf, *DOE, LBNL*
- Sunil Sheelavant, *Delft Univ. of Technology*
- John Paul Strachan, *Hewlett Packard Enterprise*
- Antonino Tumeo, *DOE, PNNL*
- Didem Unat, *Koç Üniversitesi*
- Thomas Van Vaerenbergh, *Hewlett Packard Enterprise*
- William Vanderlinde, *DOE-SC, ASCR*
- Jeffrey Vetter, *DOE, ORNL*
- Gwendolyn Voskuilen, *DOE, SNL*
- Cliff Young, *Google*

2019 Steering Committee

- Cullen Bash, *Hewlett Packard Enterprise,* 2019 General Co-Chair
- Vivek Sarkar, *Georgia Tech,* 2019 General Co-Chair
- Paolo Faraboschi, *Hewlett Packard Enterprise,* 2019 Program Co-Chair
- James Ang, *DOE, PNNL,* 2019 Program Co-Chair
- Antonino Tumeo, *DOE, PNNL,* 2019 Publicity Chair

2019 Rebooting Computing Initiative

- Tom Conte, *Georgia Institute of Technology,* Co-Chair
- Elie Track, *nVizix,* Co-Chair
- Dejan Milojicic, *Hewlett-Packard Enterprise,* Committee member
- Bruce Kraemer, *Marvell Corporation,* Committee member
- Erik DeBenedictis, *IEEE,* Committee member
- Terence Martinez, *IEEE*
- Alan Kadin, *IEEE*

Message from the 2019 ICRC General Co-Chairs


(left) Mr. Cullen Bash, *Hewlett Packard Enterprise* and
(right) Dr. Vivek Sarkar, *Georgia Tech*

Welcome to ICRC 2019! ICRC is the IEEE's premier forum for novel computing approaches. ICRC aims to advance the future of computing by considering the whole computing stack, from system and network architectures to algorithms and languages and software. As such, ICRC is broad, multidisciplinary, conversation between a diverse community of stakeholders with a passion for shaping the future of computing and along with it, the future of humanity.

ICRC also is set within the context of Rebooting Computing Week. Doing so provides attendees with a unique opportunity to participate in multiple co-located events focused on a variety of topics associated with computing's future; including an Industry Summit on the Future of Computing, a meeting of the International Roadmap for Devices and Systems, and select topics associated with IEEE's Quantum Initiative.

Whether you are here to attend the entire Rebooting Computing Week or ICRC alone, we hope you have an engaging and stimulating time participating in the formal agenda and, perhaps more importantly, engaging in conversations with the breadth of participants that the week brings together.

Message from the 2019 ICRC Program Co-Chairs


(left) Dr. James Ang, *DOE, PNNL* and
(right) Dr. Paolo Faraboschi, *Hewlett Packard Enterprise*

We are pleased to present the technical program for the 2019 International Conference on Rebooting Computing (ICRC). This is the fourth year of the conference and we are seeing strong signs of the vitality of the community in the quality of the received submissions.

For over 25 years, the computing community has leveraged the commodity computing ecosystem. Moore's Law technology advances and the clock speed multiplier of Dennard Scaling gave rise to the "Killer Micros" that led to the corresponding decline of specialized machines, like the custom vector supercomputers. In this timeframe, even supercomputers were built from the integration of commodity computing components into large-scale systems. The predictability of performance improvements for general purpose, CPU processor technologies provided computer scientists and application developers with a solid foundation for the evolution and co-design of the applications with the software stack. Processor and hardware architects could reliably use the ITRS roadmap as the driver for their performance requirements. This led to multi-disciplinary co-designs among different development activities: software, tools, algorithms and applications to improve scalability. With the end of Moore's Law rapidly approaching, the computing community is becoming increasingly more open to new opportunities for fundamental research and development in microelectronics technologies. To support these objectives significant investments have started around the world in Quantum Information Systems, Artificial Intelligence and Machine Learning, and Neuromorphic Computing. This broad span of novel computing technologies are the core domain of the IEEE International Conference on Rebooting Computing.

The Program Committee (PC) selected 20 papers (5 short and 15 regular) out of 40 submissions to appear at the conference. The selection process required 160 reviews from the 32 PC members in a single-phase review process. The vast majority of papers received four reviews and all papers were available for general discussion during the online discussion step. After the review period, we held an online discussion phase by which the PC members who reviewed the work attempted to reach a consensus for each paper, and program chairs opinion weighed in only when necessary. Conflicts of interests with one of program chairs were handled by the other co-chair.

It takes the enthusiasm of many people to put together a successful program and we would like to thank the entire Program Committee for their dedication, as well as the ICRC Steering Committee to provide useful guidance. We would also like to mention Erik Blair who dealt with collecting the final version of the papers, and all the other aspects of the proceedings publication.

ICRC 2019 Keynote Talk

9:00 AM Wednesday, November 6, 2019

Keynote Speaker: Dr. Cliff Young, *Data Scientist, Google Brain Team*

Cliff Young is a data scientist on the Google Brain team, where he works on codesign for deep learning accelerators. He is one of the designers of Google's Tensor Processing Unit (TPU), which is used in production applications including Search, Maps, Photos, and Translate. TPUs also powered AlphaGo's historic 4-1 victory over Go champion Lee Sedol. Previously, Cliff built special-purpose supercomputers for molecular dynamics at D. E. Shaw Research and worked at Bell Labs. A member of ACM and IEEE, he holds AB, MS, and PhD degrees in computer science from Harvard University.

Talk: "*Neural networks have rebooted computer architecture; what should we reboot next?*"

Abstract: We are in the midst of a neural-network revolution, where breakthroughs in speech, vision, translation, and many other areas have created exponential demands on our computing resources. To meet this demand, Google started building its own chips, Tensor Processing Units (TPUs), six years ago. TPUs can be seen as a reboot of general-purpose computer architecture, based on systolic arrays, reduced-precision arithmetic, and dedicated interconnects. But TPUs still use standard CMOS processes, and the end of Moore's Law seems nigh. A rebooted architecture is a reasonable first step, but how should we evaluate novel technologies, and when should we deploy them? I think that we will need to widen our notions of codesign even more than we already have. Technologists will need to learn how neural networks work, and system builders will need to understand how the new technologies behave differently, to build working solutions together.

ICRC 2019 Keynote Talk

9:00 AM Thursday, November 7, 2019

Keynote Speaker: Dr. Krysta M. Svore, *General Manager, Microsoft Quantum Software*

Named one of Business Insider's 39 most powerful engineers of 2018, Dr. Krysta M. Svore is the General Manager of Quantum Software at Microsoft where she leads the Quantum Architectures and Computation group. Her research focuses on the development and implementation of quantum algorithms, including the design of a scalable, fault-tolerant software architecture for translating a high-level quantum program into a low-level, device-specific quantum implementation. She has also developed techniques for protecting quantum computers from noise, including methods of quantum error correction, establishment of noise thresholds, and the development of improved decoders. She spent her early years at Microsoft developing machine-learning methods for web applications, including ranking, classification, and summarization algorithms. Her work in machine learning has expanded to include quantum algorithms for improve machine learning methods.

In addition to her work at Microsoft, Krysta has been named to the Defense Advanced Research Projects Agency (DARPA) Information Science and Technology (ISAT) Study Group for a three-year term. Dr. Svore was recently appointed as a member of the Advanced Scientific Computing Advisory Committee of the Department of Energy and chaired the 2017 Quantum Information Processing Conference. Svore received an ACM Best of 2013 Notable Article award. In 2010, she was a member of the winning team of the Yahoo! Learning to Rank Challenge. Dr. Svore is honored as a Kavli Fellow of the National Academy of Sciences. She is a Senior Member of the Association for Computing Machinery (ACM), serves as a representative for the Academic Alliance of the National Center for Women and Information Technology (NCWIT), and is an active member of the American Physical Society (APS). Dr. Svore has authored over 65 papers and has filed over 20 patents. She received her PhD in computer science with highest distinction from Columbia University and her BA from Princeton University in Mathematics with a minor in Computer Science and French. This quarter, she is teaching an undergraduate course on quantum computing at the University of Washington. She hopes to inspire young women around the world, to show them that technology is a field for everyone, and that they will be the ones to unlock technologies that will be ground-breaking and transformative. Krysta lives in Seattle, WA where she is raising her 1 year old daughter, Daisy.

Talk: "*Developing Our Quantum Future*"

Abstract: In 1981, Richard Feynman proposed a device called a "quantum computer" to take advantage of the laws of quantum physics to achieve computational speed-ups over classical methods. Quantum computing promises to revolutionize how and what we compute. Over the

course of three decades, quantum algorithms have been developed that offer fast solutions to problems in a variety of fields including number theory, optimization, chemistry, physics, and materials science. Quantum devices have also significantly advanced such that components of a scalable quantum computer have been demonstrated; the promise of implementing quantum algorithms is in our near future. I will attempt to explain some of the mysteries of this disruptive, revolutionary computational paradigm and how it will transform our digital age.

ICRC 2019 Sponsor

ICRC 2019 Patrons

ICRC 2019 Technical Program

Wed. Nov. 6, 2019	
Day 1 Introduction and Keynote	
Neural networks have rebooted computer architecture; what should we reboot next?	Cliff Young (Google)
Session 1 - Machine Learning Systems	
Reconfigurable Probabilistic AI Architecture for Personalized Cancer Treatment	Sourabh Kulkarni, Sachin Bhat, Csaba Andras Moritz (University of Massachusetts Amherst)
On a Learning Method of the SIC Fuzzy Inference Model with Consequent Fuzzy Sets	Genki Ohashi, Masahiro Inuiguchi, Hirosato Seki (Osaka University)
Deep Learning Cookbook: Recipes for your AI Infrastructure and Applications	Sergey Serebryakov, Dejan Milojicic, Natalia Vassilieva, Stephen Fleischman, Robert Clark (Cerebras, Hewlett Packard Enterprise)
Session 2 - Technology for Machine Learning	
Non-Volatile Memory Array Based Quantization- and Noise-Resilient LSTM Neural Networks	Wen Ma, Pi-Feng Chiu, Won Ho Choi, Minghai Qin, Daniel Bedau, Martin Lueker-Boden (Western Digital)
A Comparator Design Targeted Towards Neural Nets	David Mountain (Department of Defense)
FPGA demonstrator of a Programmable Ultra-efficient Memristor-based Machine Learning Inference Accelerator	Martin Foltin, Craig Warner, Eddie Lee, Sai Rahul Chalamalasetti, Chris Brueggen, Charles Williams, Nathaniel Jansen, Felipe Saenz, Luis Li, Dejan Milojicic, John Paul Strachan, Amit Sharma (Hewlett Packard Enterprise)
Panel: Machine Learning in the Valley	

Thurs. Nov. 7, 2019	
Day 2 Introduction and Keynote	
Developing our Quantum Future	Krysta Svore (Microsoft)
Session 3 - Quantum Computing	
Improved Implementation of Quantum Phase Estimation Algorithms on Quantum Computers	Hamed Mohammadbagherpoor, Young-Hyun Oh, Patrick Dreher, Anand Singh, Xianqing Yu, Andy Rindos (IBM, NCSU)
Optimizing the spin reversal transform on the D-Wave 2000Q	Elijah Pelofske, Georg Hahn, Hristo Djidjev (Los Alamos National Laboratory)
Entangled State Preparation for Non-binary Quantum Computing	Kaitlin Smith, Mitch Thornton (Southern Methodist University)
Session 4 - Future Computing Challenges	
Experimental Insights from the Rogues Gallery Testbed	Jeffrey Young, Jason Riedy, Tom Conte, Vivek Sarkar, Prasanth Chatarasi, Srisehan Srikanth (Georgia Institute of Technology)
Future Computing Systems (FCS) to Support	Ray Beausoleil, Kirk Bresniker, Cat Graves,

ICRC 2019 Technical Program

	"Understanding" Capability	Kimberly Keeton, Suhas Kumar, Can Li, Dejan Milojicic, Sergey Serebryakov, John Paul Strachan, Thomas Van Vaerenbergh (Hewlett Packard Enterprise)
	On the Limits of Stochastic Computing	Florian Neugebauer, Ilia Polian, John P. Hayes (University of Michigan, University of Stuttgart)
	Session 5 - Novel Computing Approaches	
	Design of a 16-bit Adiabatic Microprocessor	Rene Celis-Cordova, Alexei O. Orlov, Tian Lu, Jason M. Kulick, Gregory L. Snider (Indiana Integrated Circuits LLC, University of Notre Dame)
	Hierarchical Memcapacitive Reservoir Computing Networks	Dat Tran, Christof Teuscher (Portland State University)
	Integrating Motion into Vision Models for Better Visual Prediction	Michael Hazoglou, Todd Hylton (University of California San Diego)
	Fast solution of linear systems with analog resistive switching memory (RRAM)	Zhong Sun, Giacomo Pedretti, Daniele Ielmini (Politecnico di Milano)
	Design and Comparison of Crosstalk Circuits at 7nm	Md Arif Iqbal, Naveen Kumar Macha, Bhavana Tejaswini Repalle, Mostafizur Rahman (University of Missouri Kansas City)

Fri. Nov. 8, 2019	**Session 6 - Photonics**	
	Integrated Nanophotonic Architectures for Residue Number System Computations	Jiaxin Peng, Yousra Alkabani, Shuai Sun, Sorger J. Volker, Tarek El-Ghazawi (The George Washington University)
	Energy Efficiency of Microring Resonator (MRR) Based Binary Decision Diagram (BDD) Circuits	Ozan Yakar, Yuqi Nie, Kazumi Wada, Anuradha Agarwal, İlke Ercan (Boğaziçi University, Massachusetts Institute of Technology, Peking University)
	Coherent optical parallel computation of n-bit addition	Bogdan Reznychenko, Raphael Levine, Francoise Remacle, Barbara Fresch, Hugo Gattuso, Emmanuel Mazer, Maurizio Coden, Elisabetta Collini, Carlo Nazareno DiBenedetto, Ariela Donval, Noam Gross, Yossi Paltiel, Marinella Striccoli (Elbit Systems, Hebrew University of Jerusalem, Institute for Physical and Chemical Processes, Probayes, University of Liège, University of Padova)
	Panel: SRC Decadal Semiconductor Plan	

Table of Contents

Session 1 - Machine Learning Systems

Session 2 - Technology for Machine Learning

Session 3 - Quantum Computing

Session 4 - Future Computing Challenges

Session 5 - Novel Computing Approaches

Session 6 - Photonics

Reconfigurable Probabilistic AI Architecture for Personalized Cancer Treatment

Sourabh Kulkarni, Sachin Bhat, Csaba Andras Moritz
Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA, USA
skulkarni@umass.edu, andras@ecs.umass.edu

***Abstract*— The machinery of life operates on the complex interactions between genes and proteins. Attempts to capture these interactions have culminated into the study of Genetic Networks. Genetic defects lead to erroneous interactions, which in turn lead to diseases. For personalized treatment of these diseases, a careful analysis of Genetic Networks and a patient's genetic data is required. In this work, we co-design a novel probabilistic AI model along with a reconfigurable architecture to enable personalized treatment for cancer patients. This approach enables a cost-effective and scalable solution for widespread use of personalized medicine. Our model offers interpretability and realistic confidences in its predictions, which is essential for medical applications. The resulting personalized inference on a dataset of 3k patients agrees with doctor's treatment choices in 80% of the cases. The other cases are diverging from the universal guideline, enabling individualized treatment options based on genetic data. Our architecture is validated on a hybrid SoC-FPGA platform which performs 25x faster than software, implemented on a 16-core Xeon workstation, while consuming 25x less power.**

Keywords*— *Computer Architecture, Reconfigurable Computing, SoC-FPGA, Probabilistic Computing, Bayesian Networks, Personalized Medicine, Interpretable AI, Genomics.

I. Introduction

This work focuses on the problem of discovering gene networks (GNs) and their application in determining treatment choices for breast cancer patients. Globally, breast cancer is the most common cancer in women with close to 1.7 million cases diagnosed annually. The average 5-year survival rate of stage 1 (local) breast cancer is 98.5% but drops significantly to 84% and 24% for stage 2 (regional) and stage 3 (distant), respectively[1]. Hence, an early diagnosis and treatment of breast cancer is crucial, which this work aims to impact. Decades of breast cancer research has culminated into a deep understanding of its various subtypes and has led to the formation of various GNs for breast cancer[2][3]. GNs, as shown in Fig. 1, are a compact representation of our understanding of the *genetic basis of diseases*[4]. Recent developments indicate that *probabilistic* AI models may be used to capture GN structure[5]. These AI models, such as Bayesian Networks (BNs), are based on probability theory and Bayesian statistics. They intrinsically support asking questions such as those in a personalized medicine context. They represent knowledge in an interpretable graph and can provide realistic confidences in their predictions, which is important for medical applications. Once a GN structure is discovered and modeled using probabilistic AI, it can be used to predict the likelihood that a certain patient has breast cancer based on their gene expression profile. Furthermore, it can also be used to infer the best possible treatment for a patient already diagnosed of cancer.

Currently, this process is time-consuming, and expensive. A state-of-the-art algorithm performing a GN-based benchmark with 3000+ genes requires ~325 Xeon processors and ~170 hours of runtime[6]. This is because, implementing probabilistic AI in software involves stochastic computations over several layers of abstraction, on circuits and architectures that are deterministic in nature. Alternatively, personalized treatment choices are made by expert oncologists at a per-patient basis. Currently, only few institutions in the world have access to such enormous computing resources and expertise. Hence, for most cancer patients, the treatment choice is not personalized, but is generalized to factors such as age, gender, stage of cancer etc. These limitations can be overcome by designing architectures that support these probabilistic AI models where compute-intensive operations are significantly accelerated.

In this work, we co-design a probabilistic AI model and reconfigurable architecture for (i) Discovering GN structures by modelling them as BNs; and (ii) Performing inference on these BNs for personalized-treatment selection. To enable these operations, we develop modified, hardware-aware versions of BN learning and inference operations. These are implemented on an architecture that maps BN graphs onto a reconfigurable FPGA fabric that consists of several Stochastic Bayesian Nodes and supports arbitrary connectivity among them. The reconfigurability allows incorporating new domain expertise by supporting modifications to GNs, while the stochasticity of nodes allows for efficient probabilistic computations. We utilize gene expression data of ~3k breast cancer patients for validating our approach on an SoC-FPGA platform. This approach can be extended to other diseases and medical applications for humans and other life forms.

The paper is organized as follows: Section II provides a background on Bayesian Networks, the probabilistic AI models used in this work. We then describe the problem statement in section III. Section IV consists of the architectural and implementation details. Section V discusses the results and related analysis. Section VI concludes the paper.

II. Background

Bayesian Networks (BNs) are graph-based probabilistic AI models that attempt to capture knowledge of a domain by encoding the various entities as random variables and the relationships between them as causal connections. The strength of causal relation between variables is represented as a

978-1-7281-5221-9/19/$31.00 ©2019 IEEE

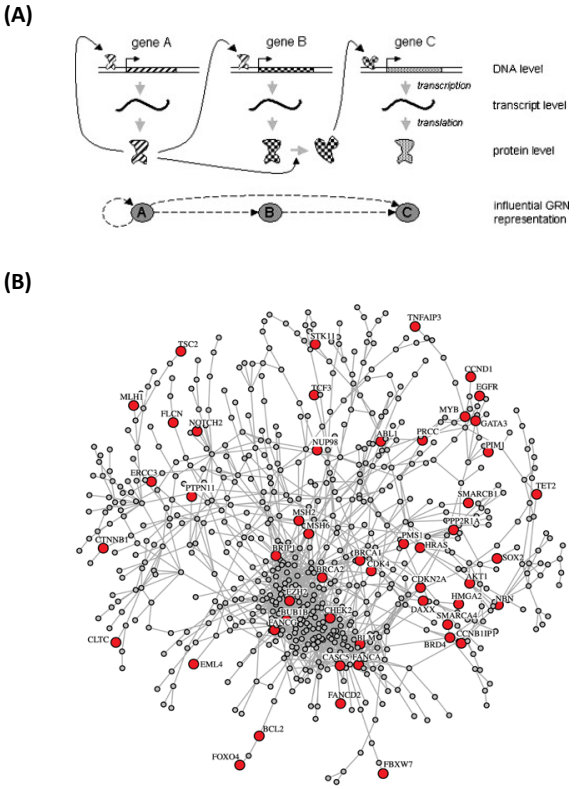


Fig. 1. (A) Schematic of a simple gene network[9] portraying gene-gene interaction through transcription and protein synthesis. (B) Subnetwork of the genome-scale breast cancer GN (19k genes); part consists of 51 breast cancer genes, 677 related genes and their interactions[10].

conditional probability table (CPT). The two major aspects for using BN models is learning and inference. Both operations are of high computational complexity and scale exponentially in the worst case[7][8].

Learning: The two major aspects involved in learning a BN model are its graph structure and its CPT parameters. The structure of the BN is inferred from data through combinatorial optimization techniques or constructed through domain experts' knowledge. This work employs a combination of both of these techniques. For learning the CPT values, we use the maximum likelihood estimate (MLE) method, which is quite suitable for our architecture and is one of the most widely used.

Inference: BN model inference can be either analytic or sampling based. Analytic inference methods tend to be exact, but are compute intensive, while sampling-based methods are approximate but potentially more computationally feasible, depending on platform. In this work, the sampling-based method is used, as it can be massively parallelized in hardware and is amenable to modifications in its implementation. These properties are useful in design of a hardware-aware inference method for BNs which can be implemented by simple computational elements.

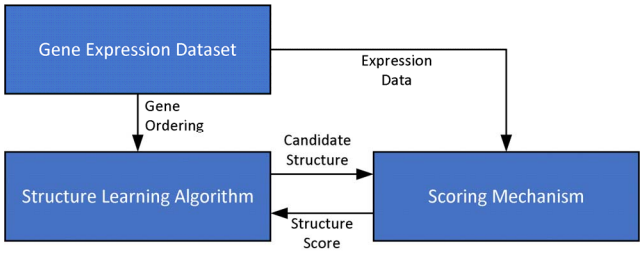


Fig. 2. Design flow for structure learning in BNs. Learning algorithm generates graphs from data, which are scored by the scoring mechanism. The algorithm iteratively improves upon the graph structure until structure scores over a certain threshold.

III. Probabilistic AI Model

This section discusses the learning and inference of the probabilistic AI model for personalized treatment. Here, GNs whose structure is to be discovered are modelled as BNs; each gene in the GN becomes a node in the BN. Hence, the problem of structure discovery for GNs is mapped to the problem of finding the best possible connections between nodes of a BN. Fig. 2 shows a high-level view of this process. The structure learning algorithm comes up with candidate structures, which are scored by a scoring mechanism. Candidate structures are generated by incrementally building on smaller structures. The scoring mechanism determines how well the candidate structure explains the genetic data. The scoring mechanism computes the *posterior structure probability* $P(structure|data)$ using the Bayes' rule as follows:

$$P(structure|data) \propto P(data|structure) * P(structure) \quad (1)$$

Where $P(data|structure)$ is the *structure likelihood* of the structure while $P(structure)$ is the *structure prior*. *Structure prior* incorporates the order in which genes are added to the structure and can be obtained from literature[3][11]. *Structure likelihood* is computed by averaging over *individual node likelihoods* in the structure:

$$P(data|structure) = \frac{1}{N}\sum_{i\in N} P(node_i, data|structure) \quad (2)$$

The *individual node likelihoods* are independent of the rest of the structure given the *parent* state, hence are approximated by performing local inference over the node and its parent using Gibbs sampling method:

$$P(node_i, data|structure) \approx 1 - \left(\frac{1}{S}\sum_{j\in S}\left(P\left(node_i|parentSample_j\right)\right) - P(node_i|data)\right) \quad (3)$$

Here, the first term represents the Gibbs' sampling inference performed over S samples and the second term is the *marginal probability* of the node given the data. The smaller the difference between these terms, the better the node's position

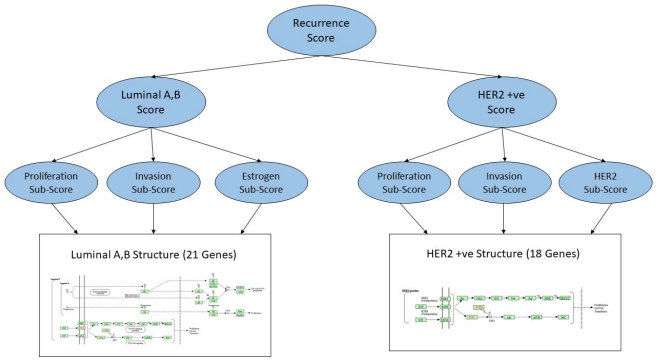


Fig. 3. Hierarchical Bayesian Latent Variable Model designed for computing probability of cancer recurrence. 'Observed' variables (bottom squares) correspond to gene expression values, 'latent' variables (blue nodes) correspond to known sub-scores associated with cancer recurrence.

and data are in agreement with the structure. Although this is an approximation, it converges to the exact result with enough samples. In this case we choose a sample size of 20k to obtain a precision of 0.00005 within the correct answer, which was sufficient for the application.

The nature of this optimization problem prohibits any proof of optimality of the candidate structure; hence, the design process is terminated when the structure is 'good enough', which is denoted by a threshold of loss function. It is therefore possible for several GN structures to agree with genetic data.

The other important aspect of this work is to predict the best possible personalized treatment. This is done by introducing latent variables in the learnt structure. The model takes personalized genetic information of a patient, and the latent models hierarchically compute a 'recurrence score': the probability of the cancer tumor to recur post-treatment. The recurrence score can then be used to select a treatment that is best suitable for the patient. The approach of recurrence-score based treatment choice selection has been validated through clinical trials[12], although by using less complex AI algorithms.

We construct the model, as shown in Fig. 3, that utilizes the GNs like the HER2 network learnt previously and another network which corresponds to Luminal A and B type breast cancers to hierarchically compute a recurrence score for each patient. These intermediate sub-scores cannot be observed from data, hence are called latent (or hidden) variables. Computing the parameters for these latent variables requires specialized analysis of data, known as factor analysis. Factor analysis is a statistical method used to describe variability among observed, correlated variables in terms of a lower number of unobserved (latent) variables called factors. The factors correspond to the first order of latent variables, which are sub-scores (proliferation, invasion, etc.) related to various subtypes of Breast Cancer. The second-order latent variables encode the probability of recurrence of certain types of Breast Cancer (Luminal A/B, HER2 +ve). The third order latent variable is the total recurrence score - the overall likelihood of recurrence of the cancer tumor. A succinct representation of the hierarchical inference operation we do over the model is as follows:

$$Recurrence\ Score = P(recurrence = high|subscores) * P(subscores|expression\ data) \quad (4)$$

Here, the $subscores$ are inferred from gene $expression\ data$ of genes in GN, while the $Recurrence\ Score$ is inferred from the $subscores$. Each of these computations uses Gibbs' sampling for inference and resembles equation (3). The recurrence score is helpful for doctors to decide on treatment for the patient. A low recurrence score (low probability of recurrence of cancer) can indicate that a less harmful treatment (e.g., endocrine treatment) should suffice. A high recurrence score indicates that a more potent but also potentially more harmful treatment (like, e.g., chemo plus endocrine treatment) may be required.

Dataset: The dataset used in this project is obtained from the Gene Expression Omnibus (GEO) Database[13]. It consists of expression values of 20,000 genes of 3,070 breast cancer patients, along with their medical information – the type of cancer diagnosed, treatments given, and the survival event. To facilitate the structure discovery, it is helpful to refer to prior information from literature. This prior information is related to what subset among the 20k genes to consider (40 genes considered). The data has been discretized into three states *High expression* (H), *Medium expression* (M) and *Low expression* (L). For validating the personalized treatment model, the treatment predictions based on the expression profiles of patients are compared with the actual treatments given to the patients and the effectiveness of those treatments.

IV. System Architecture

From a computational point of view, the probabilistic AI model can be divided into four important steps:

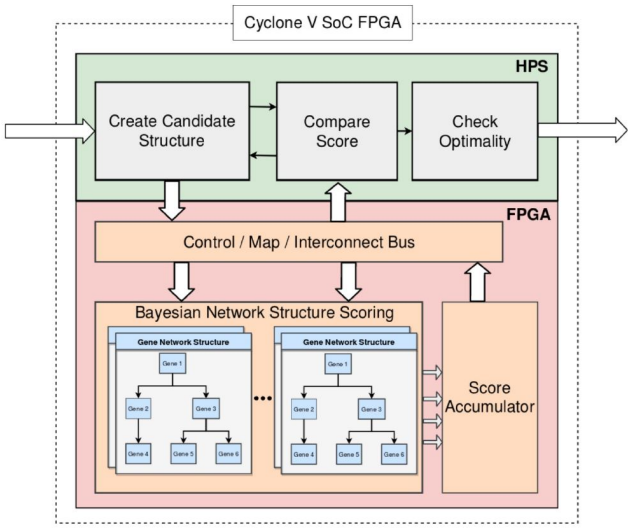


Fig. 4.Overall Architecture for structure learning task. The candidate structures are generated and incremented in software on the HPS and are mapped to FPGA for scoring.

1. *Candidate Structure Generation* – Given the ordering of genes, generate a structure with random connections.
2. *Scoring Candidate Structures* – Multiple inference operations through Gibbs sampling to generate a score for a given structure.
3. *Structure Selection* – Given multiple structures, compare their scores to select the best possible structure given data.
4. *Recurrence Score Computation* – Inference operation on the learnt structure with latent variables based on the gene expression data of a patient.

The computations in the first and third steps have a sequential flow while the second and fourth steps can be massively parallelized. Here, we chose to implement the sequential computations using software on a general-purpose processor and the parallel computations on a customized hardware based on FPGA. The software implements the sequential control flow while the custom hardware significantly accelerates scoring aspects. We chose an SoC-FPGA platform to implement our design as shown in Fig. 4. SoC-FPGAs tightly integrate both processor (HPS) and FPGA architectures on a single die. This configuration allows us to design an architecture where computational loads could be distributed among the two platforms for efficient utilization of both.

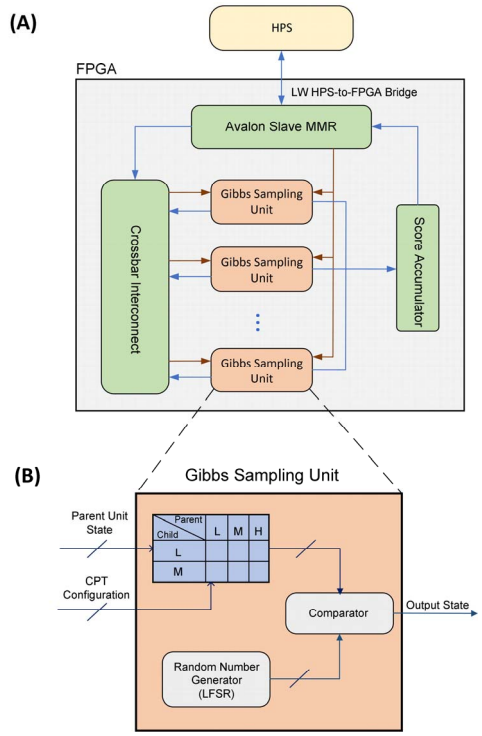


Fig. 5. FPGA-Based Proposed Architecture. (A) Shows connectivity architecture for instantiating a candidate structure in FPGA fabric. Individual nodes are mapped into Gibbs Sampling Units by instantiating the corresponding CPTs, while graph connectivity is programmed through crossbar interconnect; (B) Shows architecture of a Gibbs Sampling Unit. Parent unit state selects an entry in the CPT, which is compared to output of LFSR to determine output state of current unit.

A. Hardware Architecture for Scoring Candidate Structures

Scoring of the candidate structures is implemented on a custom reconfigurable hardware instantiated on the FPGA. The architecture allows for efficient implementation of Gibbs-sampling based inference for scoring candidate structures. The candidate structure to be scored is mapped on the FPGA. The hardware comprises of Gibbs Sampling Units which can be arbitrarily connected to each other through the programmable crossbar interconnect. Each unit consists of a soft-configurable input multiplexer that can be configured to select any other unit in the structure as its parent unit. The system is interfaced to the software using a SOPC wrapper that implements an Avalon slave memory-mapped register (MMR) IP. The software configures the network parameters, such as the input multiplexer configuration, CPT values, number of samples, by writing to the MMRs. Fig. 5(A) shows a detailed block diagram of the proposed architecture.

Gibbs Sampling Unit – A detailed block diagram of the Gibbs Sampling Unit in FPGA is shown in Fig. 5(B). The candidate structure to be scored is mapped onto the framework by mapping its nodes to Gibbs Sampling Units. Each unit consists of registers to store the CPT values with 8-bit precision. The stochastic sampling is achieved by employing a Linear Feedback Shift Register (LFSR) based pseudo-random generator in each node. It is seeded by a unique value to generate an independent sequence of 8-bit random values. A row is selected from the CPT based on the parent nodes state which is then compared with the random value to generate output state. Each output state is a 'sample' in the Gibbs sampling process. These samples are accumulated and sent to the HPS where the structure score is computed according to equation (3).

B. Software Design

The software running on the processor implements the first and third step of the AI model. The algorithm is summarized as follows:

```
BEGIN
READ expressionValues // gene expression data
READ nodeOrdering // structure prior
INITIALIZE candidateStructure with nodeOrdering.firstNode()
APPEND candidateStructure with nodeOrdering.nextNode()
CalculateCPTs()
WHILE (size(candidateStructure) < maxSize):
   APPEND candidateStructure with nodeOrdering.nextNode()
   FOR node in candidateStructure:
      AddAsParentNode()
      CalculateCPTs()
      score = ComputeStructureScore() // done in FPGA
      IF (score >= bestScore):
         KeepAsParentNode()
      ELSE:
         RemoveAsParentNode()
OUPUT bestStructure = candidateStructure
END
```

Candidate Structure Generation: The order in which the genes are added (nodeOrdering) is predetermined from literature. The software follows this order for generating the candidate structures. Initially, the search starts with a single gene in the network. As there are no parent genes to the first gene, the CPT entries for that node corresponds to the marginal probabilities of the three states for that gene, i.e., percentage of patients who have high, medium and low expression values. Subsequently more genes are added. The CPT values for these genes correspond to the probability of a gene being in one of the three states given that the parent gene is in one of the states. Once the CPT values are populated for the candidate structure, it is ready for scoring. The network along with its CPT values are written to the FPGA memory-mapped registers. The FPGA then performs the sampling and returns the score for that network.

Structure Selection: In this step, the network that will produce the best score from each order is determined. This is done by comparing the score of the network to the ground-truth (data). More and more genes are subsequently added to the network and scored. Finally, the structure with the best score is selected.

Recurrence Score Computation: Essentially, the structure discovery process involves performing multiple inferences on the various candidate structures. Hence, the same FPGA architecture can be used to perform inference as well as structure scoring. In the case of recurrence score computation, the structure is already known. The structure along with the CPTs is mapped onto the FPGA. After the sampling, the inference results for recurrence score are written back to the processor memory.

V. Analysis and Results

A. Correctness of Learnt Gene Network

The final structure generated by the algorithm is compared with a reference structure from the *KEGG Gene Pathway* database to validate the approach. Note that such comparison is not expected to be available for all applications, but we choose this specific GN such that we can also prove our design. The KEGG database is a state-of-the-art extensive library which maintains and updates a large collection of reference gene networks. Finding the correct structure for a GN from expression data is a hard problem and is currently an area of active research. As of now, there is no well-established notion of 'correctness' for a GN. As this is a small GN, we could compare it with the pre-existing GN from the KEGG database. The implementations generated are not an exact copy of the KEGG reference GN, as the reference GN is formed by compiling the results of several research efforts and we used currently a more limited dataset. This is, however, not a limitation as the qualitative benefits when scored across a larger dataset would persist. Furthermore, for the sake of simplicity, the structures being learnt in this project were restricted to be trees, which leads to few changes relative to KEGG.

B. Personalized Treatment Inference

We have developed a validation scheme for the model to gauge its accuracy. The validation is done as follows: By using the recurrence score generated by those models, we suggest a treatment. The Patient dataset has data regarding what treatment was provided to each patient along with 5-year survival event (whether patient survived for 5-yrs post treatment). This information is used to validate to what degree the treatment suggested by the model agrees with the doctors. We do a cross-validation of over 3070-patient dataset based on data from multiple hospitals[13]. The treatments provided to the patients did not include a genetic analysis but were made based on clinical factors and the doctor's expertise. As expected, from our validation, the treatments suggested by the doctors and the ones suggested by our inference model did not fully overlap but overlapped around 80% of the time. For 20% of the cases, the treatment choice suggested was different; this is likely because the gene expression-based approach is able to infer a more personalized treatment choice. That could either mean that a stronger treatment was given to patient than necessary, leading to unnecessary side effects, or that the treatment given was not strong enough, leading to sub-optimal treatment. It is to be noted that (in the general sense) it has already been shown in the research community that the gene expression-based approach for inference is more accurate[11][15] than using clinical factors alone.

TABLE 1. CYCLONE V FPGA RESOURCE USAGE FOR LEARNING AND INFERENCE OF 41-NODE GN.

	Usage	%
Logic Utilization	17060/41910	41
Total LABs	2570/4191	61
I/O Pins	172/314	55
M10K Blocks	25/553	4.5

C. Prior Related Works

While this work, to the best of our knowledge, is the first attempt for implementing Probabilistic AI Architecture for personalized medicine, there have been some works that focus on implementing BNs on FPGAs[17][18]. These approaches build on producing 'processing units' on the FPGA and distributing the inference task workload. Such an approach leads to resource-heavy architecture which is not scalable. For example, one unit in [17]takes 12-24% of the FPGA area. This requires that designs be partitioned and loaded partially into the hardware several times, hence supporting larger applications would not be feasible. In our work, the individual units are relatively simple stochastic circuits tasked for probability distribution sampling which makes our approach much more scalable.

D. Scalability of Probabilistic AI Architecture

The size of BNs which map GNs vary from few nodes to several tens of thousands of nodes depending on the gene pathway considered. Since the number of nodes that are to be mapped scales linearly with the logic blocks, the application can be easily scaled conditioned upon the availability of the logic blocks on the FPGA. For the breast cancer gene network considered, 14% of the total resources were utilized for 18-node GN including the resources utilized by GHRD that came along with the board, while for a 41-node GN, 41% resources were utilized (See Table 1). We estimate that GNs with 100s of

TABLE 2. PERFORMANCE AND POWER BENEFITS OF FPGA-BASED PROPOSED ARCHITECTURE VS. SOFTWARE BASELINE IN R

	Mean Runtime(s)			Power (W)		
	Baseline	**Proposed**	**Benefit**	**Baseline**	**Proposed**	**Benefit**
Structure Learning	0.6	0.5	~1.2x	125	5	~25x
Inference	0.05	0.002	~25x	125	5	~25x

nodes are feasible on the Cyclone V FPGA. If the candidate Bayesian graphs are less than 100 nodes in size, several graphs can be mapped onto the FPGA at once, thus enabling several graphs to be scored concurrently. For the breast cancer network considered, up to 5x speedup is possible for the 18-node GN by scoring 5x candidate structures in parallel. Similarly, for inference, the same GN can be instantiated multiple times to obtain a performance gain of 3-5x (vs. 1.2x for a single GN instantiation).

E. *Performance Evaluation and Comparison with Software Baseline*

Baseline: We compare our results with a similar implementation in R, which is a data-analysis programming language widely used in the bioinformatics domain. R has several libraries dedicated to Bayesian structure discovery, of which '*bnlearn'* was chosen. The hill-climbing algorithm (HC) was used for structure discovery. This algorithm was executed on a 4GHz, 8-core/16-thread Xeon workstation. The runtime for structure discovery operation using this configuration is listed in the table below. Performance benefits will be greater for larger networks, as even high-end CPUs are limited by core count, while the FPGA implementation of the score operation is not. The runtimes for larger network sizes will increase in CPUs while remaining near constant on proposed architecture.

Upon comparisons with the software-based approach, we observed around 25x improvement in the per-patient inference time as shown in Table 2. We estimate additional 3-5x improvement in both learning and inference if multiple GN models are instantiated on the FPGA in parallel. This is because some of the resources can be shared by multiple GN model instantiations. While the entire design with just one GN model takes up ~41% (see Table 1), we estimate that we can fit up to 5 GN models to achieve full utilization of the FPGA. The human genome consists of over 20k genes. We expect that not only the performance will improve with the size of the network but also the type of problem that we could solve would go way beyond what we have shown. Acceleration in learning and inference with up to two orders of magnitude, which we demonstrated, would allow us to consider much larger gene networks and expand this system to a wide range of other diseases.

VI. CONCLUSION

We design a probabilistic AI model for suggesting personalized treatment options for cancer patients. Furthermore, we develop a reconfigurable architecture for implementing this model where the computational workload is distributed between software and FPGA. In our model, Gene Networks are modeled as Bayesian networks for structure discovery of Breast Cancer GNs from a Gene Expression dataset of ~3k patients. We provide interpretable and personalized treatment options to Breast Cancer patients based on the latent variable model we designed for utilizing the learnt GNs. Our model agrees with the doctor's treatment choices 80% of the times, while the departure in the rest of the treatment choices could be attributed to the personalized nature of the model. Several research efforts[11][12] have shown that use of personalized genetic information leads to better diagnosis and treatment of cancer patients. To that end, we are currently in collaboration with oncologists and genetic researchers from University of Nebraska Department of Genetics and Cell Biology, and University of Debrecen Medical School, Hungary toward verifying the effectiveness of treatment choices of our model. The architecture implemented on a Cyclone V SoC-FPGA performs 25x faster and is 25x more power efficient that software-only implementation on a Xeon workstation. Our prototype platform demonstrates the feasibility of realizing a relatively low-cost solution for targeting key applications in personalized medicine. Our ultimate vision is a personalized medicine system which can be made available in hospitals and clinics all over the world. Such system would be able to perform at low cost and high-performance Bayesian inferences towards interpretable and personalized diagnosis and treatment of cancer patients. The system's reconfigurability would allow support for multiple healthcare solutions and the ability to assimilate the latest breakthroughs in medicine toward improved effectiveness.

REFERENCES

[1] Breast Cancer Research Fondation, "Breast Cancer Statistics," Available: https://www.bcrf.org/breast-cancer-statistics

[2] M. Hecker, S. Lambeck, S. Toepfer, E. van Someren, and R. Guthke, "Gene regulatory network inference: Data integration in dynamic models-A review," BioSystems, vol. 96, no. 1, pp. 86–103, 2009.

[3] Kyoto Encyclopedia of Genes and Genomes, "Breast cancer - Homo sapiens (human)," Available: https://www.genome.jp/kegg-bin/show_pathway?hsa05224=

[4] W. K. Cavenee, and R. L. White, "The genetic basis of cancer," Scientific American, vol. 272, no. 3, Mar., pp. 72-79, 1995.

[5] J. M. Pena, "Learning and Validating Bayesian Network Models of Genetic Regulatory Networks," Advances in Probabilistic Graphical Models, vol. 214, pp. 359–375, 2007.

[6] A. Frolova and B. Wilczyński, "Distributed Bayesian networks reconstruction on the whole genome scale," PeerJ, vol. 6, p. e5692, 2018.

[7] Cooper, G. F. (1990). The computational complexity of probabilistic inference using Bayesian belief networks. Artificial intelligence, 42(2-3), 393-405.

[8] Dagum, P., & Luby, M. (1993). Approximating probabilistic inference in Bayesian belief networks is NP-hard. Artificial intelligence, 60(1), 141-153.

[9] M. Hecker, S. Lambeck, S. Toepfer, E. van Someren, and R. Guthke, "Gene regulatory network inference: Data integration in dynamic models-A review," *BioSystems*, vol. 96, no. 1, pp. 86–103, 2009.

[10] F. Emmert-Streib, R.M. Simoes, P. Mullan, B. Haibe-Kains, and M. Dehmer, "The gene regulatory network for breast cancer: integrated regulatory landscape of cancer hallmarks," Frontiers in genetics, vol. 5, 2014. Available: 10.3389/fgene.2014.00015

[11] M. L. McCullough, R. M. Bostick, and T. L. Mayo, "Vitamin D gene pathway polymorphisms and risk of colorectal, breast, and prostate cancer," Annual review of nutrition, vol. 29, pp. 111-132, 2009.

[12] J. A. Sparano, R. J Gray, D. F. Makower et al., "Adjuvant Chemotherapy Guided by a 21-Gene Expression Assay in Breast Cancer," New England Journal of Medicine, vol. 379, pp. 111–121, 2018.

[13] National Center for Biotechnology Information, "Gene Expression Omnibus," Available: https://www.ncbi.nlm.nih.gov/geo/

[14] T. O'Mara, D. Duffy, D. Thompson, et al., "A genome-wide association study to identify genetic markers associated with endometrial cancer grade," Hered Cancer Clin Pract, vol. 10, no. S2, p. A47, 2012.

[15] M. J. Garnett, E. J. Edelman, et al., "Systematic identification of genomic markers of drug sensitivity in cancer cells," Nature, vol. 483, no. 7391, pp. 570-575, 2012.

[16] Kyoto Encyclopedia of Genes and Genomes, "Breast cancer - Homo sapiens (human)," Available: https://www.genome.jp/kegg-bin/show_pathway?hsa05224

[17] Z. Kulesza and W. Tylman, "Implementation Of Bayesian Network In FPGA Circuit," Proceedings of the International Conference Mixed Design of Integrated Circuits and System, 2006. MIXDES 2006., Gdynia, 2006, pp. 711-715.

[18] M. Lin, I. Lebedev, and J. Wawrzynek, "High-Throughput Bayesian Computing Machine with Reconfigurable Hardware," Proc. Int. Symp. F. Program. Gate Arrays, pp. 73–82, 2010.

On a Learning Method of the SIC Fuzzy Inference Model with Consequent Fuzzy Sets

Genki Ohashi, *Student Member*, Hirosato Seki, *Member*, *IEEE*, Masahiro Inuiguchi, *Member*, *IEEE*
Department of Systems Innovation
Graduate School of Engineering Science, Osaka University
Osaka, Japan
g.ohashi@inulab.sys.es.osaka-u.ac.jp,{seki,inuiguti}@sys.es.osaka-u.ac.jp

***Abstract*—In the conventional fuzzy inference models, various learning methods have been proposed. It is generally impossible to apply the steepest descent method to fuzzy inference models with consequent fuzzy sets, such as Mamdani's fuzzy inference model because it uses min and max operations in the inference process. Therefore, the Genetic Algorithm (GA) was useful for learning of the above model. In addition, it has been also proposed the method for obtaining fuzzy rules of the fuzzy inference models unified max operation from the steepest descent method by using equivalence property. On the other hand, Single Input Connected (SIC) fuzzy inference model can set a fuzzy rule of 1 input 1 output, so the number of rules can be reduced drastically. In the learning method of SIC model unified max operation with consequent fuzzy sets, GA was only applied to the model. Therefore, this paper proposes a leaning method of SIC model unified max operation with consequent fuzzy sets by using equivalence. Moreover, the proposed method is applied to a medical diagnosis and compared with the SIC model by using GA.**

***Index Terms*—SIC fuzzy inference model, max operation, the steepest descent, genetic algorithm**

I. Introduction

Since Mamdani adapted concept of fuzzy inference to control of steam engine experimental equipment, it has been applied in various fields. The idea of the fuzzy set forming the basis of the fuzzy system which uses mathematically indescribable linguistic variable has been proposed by Zadeh in 1965 [1]. Fuzzy sets can handle consecutive "fuzzines" belonging to a set by expressing the degree of belonging to the set by continuous values. Based on this concept of fuzzy sets, fuzzy inference is the rule form adapted to conventional logic. As a result, it is possible to formulate reasoning that ambiguity is commonly used by humans, and ambiguity-containing information processing has become possible in computers.

There are typical fuzzy inference models such as the Mamdani-type fuzzy inference model, T-S fuzzy inference model, the simplified fuzzy inference model. With this method, we can ask for a conclusion without taking steps such as truncation or integration, so the processing speed is very fast. Then, a method of automatically adjusting fuzzy rules by applying the steepest descent method using teach data has been proposed [2]. On the other hand, in Mamdani-type fuzzy inference model [3], since the calculations of min and max operations are used in the output derivation process, it is difficult to differentiate. Therefore, applying it to steepest descent method is generally impossible. However, the fuzzy rules are easier to interpret linguistically than the simplified fuzzy inference model because of using fuzzy sets to the consequent part of fuzzy rules. Therefore, the Genetic Algorithm (GA) was useful for learning of the above model because genetic algorithm don't need to differentiate. In addition, it has been also proposed the method for obtaining fuzzy rules of the fuzzy inference models unified max operation from the steepest descent method by using equivalence property [4]. However, in the genetic algorithm [5] [6], random numbers are used to accidentally create multiple new learning individuals, and among them, there is an operation to select the best individual among them, so the computational process time is much slowly than the steepest descent method.

In the conventional if-then fuzzy inference model, all input items of the system are set to the antecedent part of if, and all output items are set to the consequent part of then. As a result, it becomes difficult to set and adjust fuzzy rules, and problems such as an increase in the number of rules occur. Moreover, considering the computational complexity of genetic algorithm, it is not necessarily suitable. On the other hand, Single Input Connected (SIC) fuzzy inference model [7] (SIC model in short) can reduce the number of rules drastically because the fuzzy rules constitute 1 input 1 output. The consequent parts of the conventional SIC model are real number although linguistic interpretation is possible and easy to understand if the consequent parts are fuzzy sets. In this study, the fuzzy rules of SIC model with consequent fuzzy sets are derived by using the steepest descent method and by using genetic algorithm. Moreover, this method is applied to a medical diagnosis and compared with the SIC model by using the steepest descent method and by using genetic algorithm

This paper is organized as follows. Section Ⅱ introduces SIC fuzzy inference model which form the basis of this research. Section Ⅲ introduces a learning method of SIC model, there are the steepest decent method. Section Ⅳ introduces, we construct a medical diagnosis system using it, perform numerical experiments and compare the accuracy with by using genetic algorithm. Section Ⅴ discusses the conclusion of this research and future tasks.

Identify applicable funding agency here. If none, delete this.

978-1-7281-5221-9/19/$31.00 ©2019 IEEE

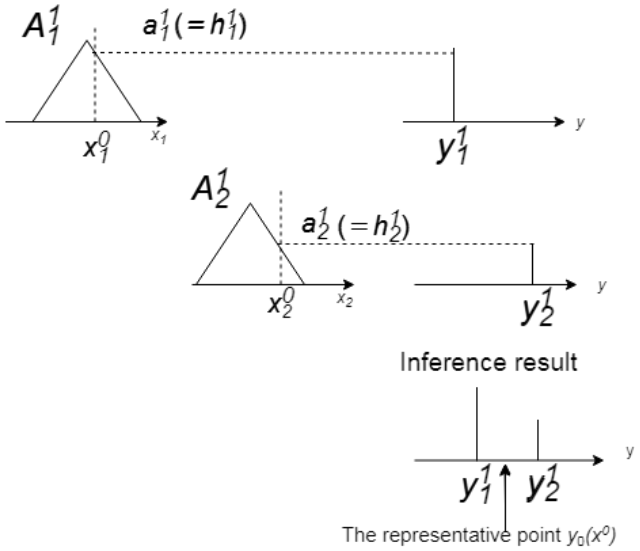


Fig. 1. Inference Figure of SIC model

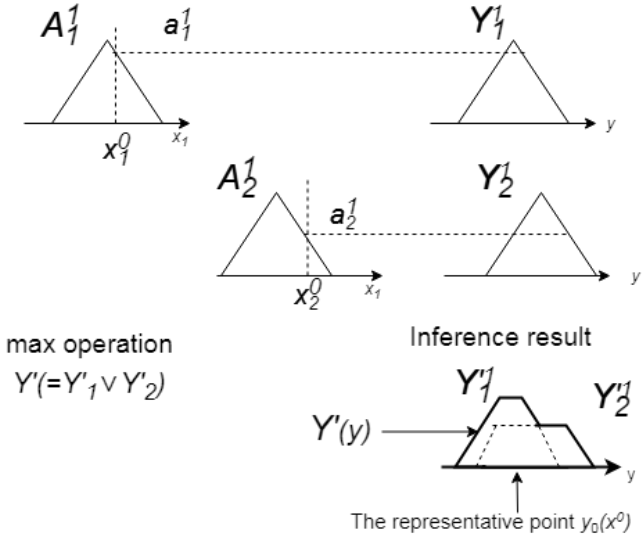


Fig. 2. Inference Figure of SIC model with consequent fuzzy sets

II. The SIC Fuzzy Inference model

A. SIC model

The rule module Rules-i for the ith input of the SIC model is given as follows:

$$\text{Rules}-i : \{x_i \text{ is } A_i^j \longrightarrow y_i \text{ is } y_i^j\}_{j=1}^{m_i}$$

where x_i is the ith input item in each rules, A_i^j is the antecedent fuzzy set belonging to ith input of the jth fuzzy rule, y_i^j is a real number belonging to ith input of the jth rule consequences, where $i = 1, \ldots, n$ and $j = 1, \ldots, m_i$. The number of rules of the simplified fuzzy inference model is $\prod_{i=1}^{n} m_i$, but that of SIC model is $\sum_{i=1}^{n} m_i$, so it can be seen that the number of rules in the SIC model is smaller than that of the simplified fuzzy inference model.

Given $\boldsymbol{x}^0 = (x_1^0, \ldots, x_n^0)$, the firing strength h_i^j for the jth rule in the ith rule module is calculated by

$$h_i^j = A_i^j(x_i^0) \tag{1}$$

The inference result $y_0(\boldsymbol{x}^0)$ is calculated by

$$y_0(\boldsymbol{x}^0) = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m_i} h_i^j y_i^j}{\sum_{i=1}^{n}\sum_{j=1}^{m_i} h_i^j} \tag{2}$$

SIC model in the case of 2 input 2 rules is shown in Fig.1 as toy example.

B. SIC models with consequent fuzzy sets

We have proposed SIC models with consequent fuzzy sets [9]. The rule module Rules-i for the ith input of the SIC model extended to the consequent fuzzy set is given as follows:

$$\text{Rules}-i : \{x_i \text{ is } A_i^j \longrightarrow y_i \text{ is } Y_i^j\}_{j=1}^{m_i}$$

where x_i is the ith input item in each rules, A_i^j and Y_i^j are the antecedent and the consequent fuzzy sets belonging to the ith input of the jth fuzzy rule respectively, where $i = 1, \ldots, n$ and $j = 1, \ldots, m_i$. The number of rules of this model is $\sum_{i=1}^{n} m_i$, as the same as the conventional SIC model.

Given $\boldsymbol{x}^0 = (x_1^0, \ldots, x_n^0)$, the firing strength h_i^j for the jth rule in the ith rule module is calculated by

$$h_i^j = A_i^j(x_i^0) \tag{3}$$

The inference result for the jth rule in ith input is calculated by

$$Y_i'^j = T_1(h_i^j, Y_i^j) \tag{4}$$

The inference result can be derived using equation

$$Y'(y) = T_2{}_{i=1}^{n} T_3{}_{j=1}^{m_i}(Y_i'^j) \tag{5}$$

The representative point of the inference result (5) can be obtained by (6).

$$y_0(\boldsymbol{x}^0) = \frac{\int y \cdot Y'(y) dy}{\int Y'(y) dy} \tag{6}$$

In this study, we discuss the case where T_1 means min and T_2 and T_3 mean max. The SIC model with consequent fuzzy sets in the case of 2 input 2 rules is shown in Fig. 2 as toy example.

The fuzzy rule for fuzzy inference model unified max operation cannot be obtained by the steepest decent method in general because that is indifferentiable. This paper proposes a new learning method using equivalence and the steepest descent method. The model for applying the steepest descent method is described. The fuzzy sets of consequent parts are used as isosceles triangle in this paper. A rule number t is assigned to the rule module i and the jth rule related to the

rule module i in the SIC model. Here, when $i = 1, \ldots, n$ and $j = 1, \ldots, m_i$, so rule numbers $t = 1, \ldots, \sum_{i=1}^{n} m_i$. As a result, the center of gravity becomes the center a_t of the consequent part. Also, the overlapping part between two adjacent inference results Y'_u and Y'^{i}_{u+1} is a similar triangle. Where $u = 1, \ldots, \sum_{i=1}^{n} m_i - 1$. Therefore, the center of gravity of the overlapping portion coincides with the intersection point due to similarity relationship. The overlapping area is s^o_u, its center of gravity is a^o_u, where $u = 1, \ldots, \sum_{i=1}^{n} m_i - 1$. And, the area of the inference result Y'_t from each rule is s'_t. In the case of using max in (5) and (6) can be transformed into (7) by using equivalence [7] [8].

$$y_0(\boldsymbol{x}^0) = \frac{\sum_{t=1}^{nm_i} a_t s'_t - \sum_{u=1}^{nm_i-1} a^o_u s^o_u}{\sum_{t=1}^{nm_i} s'_t - \sum_{u=1}^{nm_i-1} s^o_u} \tag{7}$$

where nm_i means the number of rules. By transforming to (7), the consequent parts are fuzzy sets and even if integrated by max operation, the steepest descent method can be applied.

III. A Learning Algorithm

In this section, the consequent part are set to be the fuzzy sets and integrate even if it does, we explain to derive a learning method by using the steepest decent method.

A. Adaptation of the steepest descent method to SIC model with consequent fuzzy sets

In this paper, we apply the neuro fuzzy method [4] to the SIC model with consequent fuzzy sets using max operation based on the inference result of (7), and calculate the consequent part parameters (center a_t, width b) and the antecedent part parameters (center c_t, width d_t) of the fuzzy rule are learned so that the following evaluation function is minimized [4].

evaluation function:

$$E = \frac{1}{2}(y^T - y_0(\boldsymbol{x}^0))^2 \tag{8}$$

where y^T is the true value, and $y_0(\boldsymbol{x}^0)$ is the fuzzy inference output value. The width of the fuzzy set of the consequent part is commonly set to b in all rules so that the overlapping part of the inference result Y'_t from each fuzzy rule always becomes an isosceles triangle. Each parameter are defined by the steepest descent method as follows.

Learning methods are calculated by classifying depending on the existence of overlapping part of the consequent part and its shape. In the overlapping part, the calculation method changes depending on the height of the intersection of the consequent part fuzzy set and the antecedent membership value of the left and right rules. That is, when the height of the intersection point of the fuzzy set of the uth consequent part is q_u and the left and right uth membership values are h_u, h_{u+1}, it divided into three cases:(i) : $q_u < h_u$ and $q_u < h_{u+1}$, (ii) : $h_u < h_{u+1}$ and $h_u \leqq q_u$, (iii) : $h_{u+1} \leqq h_u$ and $h_{u+1} \leqq q_u$.

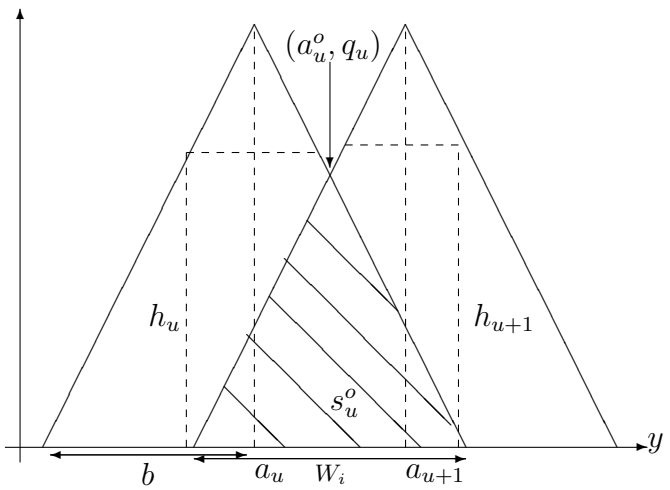


Fig. 3. Figure of the consequent part in the case of (i)

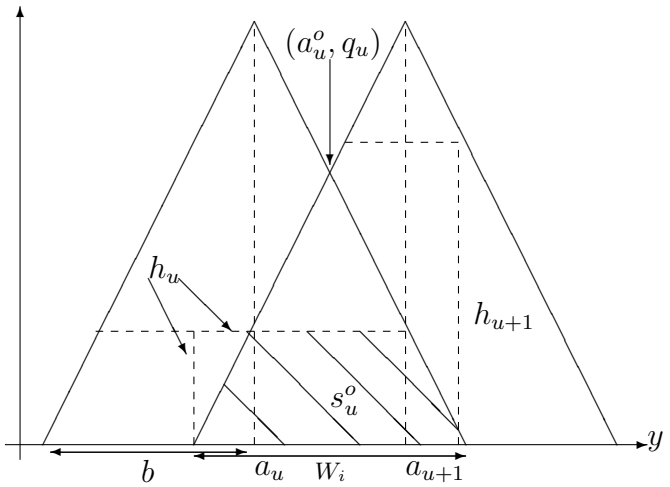


Fig. 4. Figure of the consequent part in the case of (ii)

The following Fig. 3 is the consequent part in the case of (i) and Fig. 4 is the consequent part in the case of (ii) and Fig. 5 is the consequent part in the case of (iii)

From the following Fig. 4, Fig. 5 and Fig. 6, we can easily define the following equation.

$$\begin{aligned} W_i &= (a_u + b) - (a_{u+1} - b) \\ &= a_u - a_{u+1} + 2b \end{aligned} \tag{9}$$

$$a^o_u = \frac{a_u + a_{u+1}}{2} \tag{10}$$

$$\begin{aligned} q_u &= 1 - \frac{a^o_u - a_u}{b} \\ &= 1 - \frac{a_{u+1} - a_u}{2b} \end{aligned} \tag{11}$$

The overlapping area s^o_u in the three cases of (i) (ii) (iii) is obtained as follows. In each case, we obtain $\frac{ds^o_u}{da_u}$, $\frac{ds^o_u}{da_{u+1}}$, $\frac{ds^o_u}{db}$, $\frac{ds^o_u}{dh_u}$, $\frac{ds^o_u}{dh_{u+1}}$.

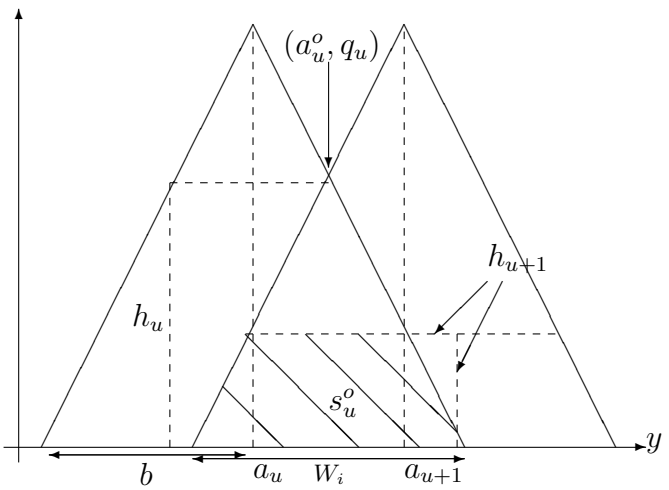


Fig. 5. Figure of the consequent part in the case of (iii)

(i) : $q_u < h_u$ and $q_u < h_{u+1}$

$$s_u^o = \frac{W_i \cdot q_u}{2} \tag{12}$$

$$\frac{ds_u^o}{da_u} = \frac{1}{2}(q_u + \frac{W_i}{2b}) \tag{13}$$

$$\frac{ds_u^o}{da_{u+1}} = -\frac{1}{2}(q_u + \frac{W_i}{2b}) \tag{14}$$

$$\frac{ds_u^o}{db} = (q_u + W_i \frac{a_{u+1} - a_u}{4b^2}) \tag{15}$$

$$\frac{ds_u^o}{dh_u} = 0 \tag{16}$$

$$\frac{ds_u^o}{dh_{u+1}} = 0 \tag{17}$$

(ii) : $h_u < h_{u+1}$ and $h_u \leqq q_u$

$$s_u^o = \frac{q_u W_i (1 - (1 - \frac{h_u}{q_U})^2)}{2} \tag{18}$$

$$\frac{ds_u^o}{da_u} = \frac{W_i(1 - (1 - \frac{h_u}{q_u})^2)}{4b} + \frac{q_u(1 - (1 - \frac{h_u}{q_u})^2)}{2} - \frac{W h_u (1 - \frac{h_u}{q_u})}{2bq_u} \tag{19}$$

$$\frac{ds_u^o}{da_{u+1}} = -\frac{W_i(1 - (1 - \frac{h_u}{q_u})^2)}{4b} - \frac{q_u(1 - (1 - \frac{h_u}{q_u})^2)}{2} + \frac{W h_u (1 - \frac{h_u}{q_u})}{2bq_u} \tag{20}$$

$$\frac{ds_u^o}{db} = q_u(1 - (1 - \frac{h_u}{q_u})^2) + \frac{W_i(a_{u+1} - a_u)(1 - (1 - \frac{h_u}{q_u})^2)}{4b^2} + \frac{W h_u (1 - \frac{h_u}{q_u})(a_{u+1} - a_u)}{2b^2 q_u} \tag{21}$$

$$\frac{ds_u^o}{dh_u} = W_i(1 - \frac{h_u}{q_u}) \tag{22}$$

$$\frac{ds_u^o}{dh_{u+1}} = 0 \tag{23}$$

(iii) : $h_{u+1} \leqq h_u$ and $h_{u+1} \leqq q_u$

$$s_u^o = \frac{q_u W_i (1 - (1 - \frac{h_{u+1}}{q_u})^2)}{2} \tag{24}$$

$$\frac{ds_u^o}{da_u} = \frac{W_i(1 - (1 - \frac{h_{u+1}}{q_u})^2)}{4b} + \frac{q_u(1 - (1 - \frac{h_{u+1}}{q_u})^2)}{2} - \frac{W h_{u+1} (1 - \frac{h_{u+1}}{q_u})}{2bq_u} \tag{25}$$

$$\frac{ds_u^o}{da_{u+1}} = -\frac{W_i(1 - (1 - \frac{h_{u+1}}{q_u})^2)}{4b} - \frac{q_u(1 - (1 - \frac{h_{u+1}}{q_u})^2)}{2} + \frac{W h_{u+1} (1 - \frac{h_{u+1}}{q_u})}{2bq_u} \tag{26}$$

$$\frac{ds_u^o}{db} = q_u(1 - (1 - \frac{h_{u+1}}{q_u})^2) + \frac{W_i(a_{u+1} - a_u)(1 - (1 - \frac{h_{u+1}}{q_u})^2)}{4b^2} + \frac{W h_{u+1} (1 - \frac{h_{u+1}}{q_u})(a_{u+1} - a_u)}{2b^2 q_u} \tag{27}$$

$$\frac{ds_u^o}{dh_u} = 0 \tag{28}$$

$$\frac{ds_u^o}{dh_{u+1}} = W(1 - \frac{h_{u+1}}{q_u}) \tag{29}$$

First, consider the learning equation at the center of the fuzzy set of the consequent part.

$$\begin{aligned} {a_t}^{\text{new}} &= {a_t}^{\text{old}} - \alpha \frac{dE}{da_t} \\ &= {a_t}^{\text{old}} - \alpha \frac{dE}{dX} \frac{dX}{dy_0} \frac{dy_0}{da_t} \\ &= {a_t}^{\text{old}} - \alpha (y_0(\boldsymbol{x}_k^0) - t_k) \frac{dy_0}{da_t} \end{aligned} \tag{30}$$

Inference results (1) B_1' has one overlapping part with the inference result in the left part of the inference result. Inference

results (2) B'_{nm_i} has one overlapping part with the inference result in the right part of the inference result. Inference results (3) $B'_t(t = 2, \ldots, nm_i - 1)$ is considered to have overlapping parts with left and right inference results. By defining the following equation, a learning model can be formulated.

$$T_u = \frac{1}{2}s_u^o + a_u^o \frac{ds_u^o}{da_u} \tag{31}$$

$$U_u = \frac{1}{2}s_u^o + a_u^o \frac{ds_u^o}{da_{u+1}} \tag{32}$$

$$V_u = \frac{ds_u^o}{da_u} \tag{33}$$

$$I_u = \frac{ds_u^o}{db} \tag{34}$$

$$J_u = \frac{ds_u^o}{dh_u} \tag{35}$$

$$K_u = \frac{ds_u^o}{dh_{u+1}} \tag{36}$$

The learning equation at the center of the consequent part is the following expression.

(1)$t = 0$

$$a_t{}^{\text{new}} = a_t{}^{\text{old}} - \alpha(y_0(\boldsymbol{x}_k^0) - t_k)\frac{M(s'_t - T_t) - N(-V_t)}{M^2} \tag{37}$$

(2)$t = nm_i$

$$a_t{}^{\text{new}} = a_t{}^{\text{old}} - \alpha(y_0(\boldsymbol{x}_k^0) - t_k)\frac{M(s'_t - U_{t-1}) - N(V_{t-1})}{M^2} \tag{38}$$

(3)$t = 2, \ldots, n-1$

$$a_t{}^{\text{new}} = a_t{}^{\text{old}} - \alpha(y_0(\boldsymbol{x}_k^0) - t_k) \frac{M(s'_t - T_t - U_{t-1}) - N(V_{t-1} - V_t)}{M^2} \tag{39}$$

Since the value of the width of the fuzzy set of the consequent part is unified to be b, the learning equation of the width of the consequent part is as follows.

$$\begin{aligned} b^{\text{new}} &= b^{\text{old}} - \beta\frac{dE}{db} \\ &= b^{\text{old}} - \beta\frac{dE}{dX}\frac{dX}{dy_0}\frac{dy_0}{db} \\ &= b^{\text{old}} - \beta(y_0(\boldsymbol{x}_k^0) - t_k) \\ &\quad \frac{M((\sum_{t=1}^{nm_i} a_t(1-(1-h_t)^2) - \sum_{i=u}^{nm_i-1} I_u a_u^o))}{M^2} \\ &\quad \frac{-N(\sum_{t=1}^{nm_i}(1-(1-h_t)^2) - \sum_{u=1}^{nm_i-1} I_u}{M^2} \end{aligned} \tag{40}$$

(1)$t = 0$

$$\begin{aligned} c_t{}^{\text{new}} &= c_t{}^{\text{old}} - \gamma\frac{dE}{dc_t} \\ &= c_t{}^{\text{old}} - \gamma\frac{dE}{dX}\frac{dX}{dy_0}\frac{dy_0}{dh_t}\frac{dh_t}{dc_t} \\ &= c_t{}^{\text{old}} - \gamma(y_0(\boldsymbol{x}_k^0) - t_k) \\ &\quad \frac{M(2a_t b(1-h_t) - a_u^o J_i) - N(2b(1-h_t) - J_t)}{M^2} \\ &\quad \frac{\text{sgn}(x_t - c_t)}{d_t} \end{aligned} \tag{41}$$

$$\begin{aligned} d_t{}^{\text{new}} &= d_t{}^{\text{old}} - \delta\frac{dE}{dd_t} \\ &= d_t{}^{\text{old}} - \delta\frac{dE}{dX}\frac{dX}{dy_0}\frac{dy_0}{dh_t}\frac{dh_t}{dd_t} \\ &= d_t{}^{\text{old}} - \delta(y_0(\boldsymbol{x}_k^0) - t_k) \\ &\quad \frac{M(2a_t b(1-h_t) - a_t^o J_t) - N(2b(1-h_t) - J_t)}{M^2} \\ &\quad \frac{|x_t - c_t|}{(d_t)^2} \end{aligned} \tag{42}$$

(2)$t = nm_i$

$$\begin{aligned} c_t{}^{\text{new}} &= c_t{}^{\text{old}} - \gamma\frac{dE}{dc_i^j} \\ &= c_t{}^{\text{old}} - \gamma\frac{dE}{dX}\frac{dX}{dy_0}\frac{dy_0}{dh_u}\frac{dh_u}{dc_t} \\ &= c_t{}^{\text{old}} - \gamma(y_0(\boldsymbol{x}_k^0) - t_k) \\ &\quad \frac{M(2a_t b(1-h_t) - a_{t-1}^o K_{t-1})}{M^2} \\ &\quad \frac{-N(2b(1-h_t) - K_{t-1})}{M^2}\frac{\text{sgn}(x_t - c_t)}{d_t} \end{aligned} \tag{43}$$

$$\begin{aligned} d_t{}^{\text{new}} &= d_t{}^{\text{old}} - \delta\frac{dE}{dd_i^j} \\ &= d_t{}^{\text{old}} - \delta\frac{dE}{dX}\frac{dX}{dy_0}\frac{dy_0}{dh_u}\frac{dh_u}{dd_t} \\ &= d_t{}^{\text{old}} - \delta(y_0(\boldsymbol{x}_k^0) - t_k) \\ &\quad \frac{M(2a_t b(1-h_t) - a_{t-1}^o K_{t-1})}{M^2} \\ &\quad \frac{-N(2b(1-h_t) - K_{t-1})}{M^2}\frac{|x_t - c_t|}{(d_t)^2} \end{aligned} \tag{44}$$

(3)$t = 2, \ldots, n-1$

$$
\begin{aligned}
{c_t}^{\text{new}} &= {c_t}^{\text{old}} - \gamma \frac{dE}{dc_i^j} \\
&= {c_t}^{\text{old}} - \gamma \frac{dE}{dX} \frac{dX}{dy_0} \frac{dy_0}{dh_u} \frac{dh_u}{dc_t} \\
&= {c_t}^{\text{old}} - \gamma (y_0(\boldsymbol{x}_k^0) - t_k) \\
&\quad \frac{M(2a_t b(1-h_t) - a_t^o J_t - a_{t-1}^o K_{t-1})}{M^2} \\
&\quad \frac{-N(2b(1-h_t) - J_t - K_{t-1})}{M^2} \frac{\text{sgn}(x_t - c_t)}{d_t}
\end{aligned} \tag{45}
$$

$$
\begin{aligned}
{d_t}^{\text{new}} &= {d_t}^{\text{old}} - \delta \frac{dE}{dd_i^j} \\
&= {d_t}^{\text{old}} - \delta \frac{dE}{dX} \frac{dX}{dy_0} \frac{dy_0}{dh_u} \frac{dh_u}{dd_t} \\
&= {d_t}^{\text{old}} - \delta (y_0(\boldsymbol{x}_k^0) - t_k) \\
&\quad \frac{M(2a_t b(1-h_t) - a_t^o J_t - a_{t-1}^o K_{t-1})}{M^2} \\
&\quad \frac{-N(2b(1-h_t) - J_t - K_{t-1})}{M^2} \frac{|x_t - c_t|}{(d_t)^2}
\end{aligned} \tag{46}
$$

where α, β, γ, and δ are learning coefficients, as trial and error, we set $\alpha = 0.0001, \beta = 0.00001, \gamma = 0.01$, and $\delta = 0.01$ so that fuzzy sets of the consequent part of the rule is not triple.

IV. Application to a medical diagnosis

In this section, we construct a medical diagnostic system to study this learning method. And we compare the learning result which are the average of mean squared error and the correct answer rate of the SIC model with using the steepest descent method and using genetic algorithm. When conducting numerical experiments, the real data [10] about diabetes diagnosed by a doctor are used in this paper. The 145 numerical data are normalized data consisting of 5 condition attributes and 1 decision attribute. The 5 input items of "relative weight", "normal blood sugar level", "fasting blood sugar level", "insulin level" and "SSPG value", and the output is "normal" if it is in the range of 0 to 0.25 for diabetes. If it is in the range of 0.25 to 0.75, it is considered as "chemical diabetes", and if it is in the range of 0.75 to 1.00, it is considered as "clinical diabetes". At the time of the experiment, 10 random data sets of 73 as teach data and the remaining 72 as evaluation data are randomly assigned 10 times, and 10 random data sets are created and used. On learning using by the steepest descent method, learning is performed 1500 times and the results are evaluated.

Also, the SIC model with consequent fuzzy sets is learned using the proposed learning method as shown Section Ⅲ, learning result is compared with genetic algorithm [6] [9] as a conventional learning method. Crossover is performed using "BLX-α" [6], and 10 individual offsprings are generated from two parent individuals according to the crossover probability, with α = 0.5. Mutation changes the value of a parent individual by a randomly given width according to the mutation probability. During selection and selection, 500 individuals are selected as the next generation in order from the individual with the highest correct answer rate. In addition, one individual is randomly selected from the population with the highest correct answer rate among the parent individuals of each generation, and the data is recorded as a representative of the generation. The computer system has a CPU of Intel (R) Core (TM) i7-4790 CPU @ 3.60 GHz and a memory of 8.00 GB.

A. Antecedent parts: Two-divided fuzzy sets, Consequent parts: fuzzy sets

1) The average of mean squared error and correct answer rate: The average of mean squared error and correct answer rate in average of 10 data when evaluation data are teach data are shown in Table I. The average of mean squared error and correct answer rate in average of 10 data when evaluation data are evaluation data are shown in Table II. The values in the table are rounded. Here, SD means the steepest decent method.

TABLE I
CORRECT ANSWER RATE AND MEAN SQUARED ERROR WHEN EVALUATION DATA ARE TEACH DATA

data	mean squared error		correct answer rate	
	SD	GA [9]	SD	GA [9]
1	0.01416	0.00000	86.3	100.0
2	0.07323	0.00000	80.8	100.0
3	0.02722	0.00000	91.8	100.0
4	0.02009	0.00343	86.3	98.6
5	0.01136	0.00000	87.7	100.0
6	0.02013	0.00000	82.2	100.0
7	0.01167	0.00000	89.0	100.0
8	0.01042	0.00000	87.7	100.0
9	0.01508	0.00000	86.3	100.0
10	0.00751	0.00000	91.8	100.0
average	0.02109	0.00034	87.0	99.9

TABLE II
CORRECT ANSWER RATE AND MEAN SQUARED ERROR WHEN EVALUATION DATA ARE EVALUATION DATA

data	mean squared error		correct answer rate	
	SD	GA [9]	SD	GA [9]
1	0.01332	0.00000	80.6	100.0
2	0.03481	0.00694	88.9	97.2
3	0.02555	0.03125	84.7	87.5
4	0.02644	0.01736	83.3	93.1
5	0.01512	0.01736	87.5	93.1
6	0.02073	0.00694	81.9	97.2
7	0.02258	0.02083	79.2	91.7
8	0.02197	0.03472	76.4	90.3
9	0.01784	0.00694	77.8	97.2
10	0.01367	0.00694	86.1	97.2
average	0.02120	0.01493	82.6	94.4

In the steepest descent method, there is no guarantee that the solution will fall into the global optimum solution, but it

may fall into the local solution depending on the initialization and the bias of the data set. As a result of the steepest descent method, it is considered that the correct answer rate is lower than that of using genetic algorithm because the data sets are imbalanced and it is a local solution. From the result, the average result of the steepest decent method is inferior to the genetic algorithm. The learning results of the genetic algorithm obtained good results for both teach data and evaluation data. In particular, the correct answer rate of the teach data is very close to 100 percent.

2) Calculation time: Table III shows the comparison of calculation processing average time of 10 data sets. Comparing the calculation time, compare the SIC model with consequent fuzzy sets learned using the steepest descent method and genetic algorithm.

TABLE III
CALCULATION TIME

	SD	GA [9]
average	12.85(second)	700.06(second)

The result of using genetic algorithm is very good in the average of mean squared error and the correct answer rate, but the model learned using the steepest descent method is able to reduce the calculation time overwhelmingly when comparing with the calculation time. And when the number of data increases, it is thought that it appears more notably.

B. Antecedent parts: Three-divided fuzzy sets, Consequent parts: fuzzy sets

1) The average of mean squared error and correct answer rate: The average of mean squared error and correct answer rate in average of 10 data when evaluation data are teach data are shown in Table IV. The average of mean squared error and correct answer rate in average of 10 data when evaluation data are evaluation data are shown in Table V. Here, SIC model with CFS means SIC model with consequent fuzzy sets, Mamdani's model means Mamdani's fuzzy inference model.

The results obtained by learning using the genetic algorithm are better than the results obtained by learning using the steepest descent method, as is the case of the antecedent part of two-divide. However, even in the case of the steepest descent method, correct answer rate of 88 percent or more is obtained, and it could be confirmed that learning is possible. As a result of comparing SIC model using the steepest decent method and Mamdani's fuzzy inference model using genetic algorithm, it is able to learn almost equally when teach data is used for evaluation data. And, when evaluation data is used for evaluation data, the result is better than Mamdani's fuzzy inference model.

2) Calculation time: Table VI shows the comparison of calculation processing average time of 10 data sets.

As for the calculation time, it is found that the model learned using the steepest descent method is shorter than the calculation time as in the case of the antecedent part of two-divide. In addition, as the number of divisions of the antecedent parts in fuzzy rules increases in the Mamdani's fuzzy inference model, the rules increase exponentially and the calculation time increases. On the other hand, the number of rules does not increase exponentially in these SIC models, and the calculation time does not increase rapidly like Mamdani's fuzzy inference model.

V. CONCLUTION

In this study, for the SIC fuzzy inference model with consequent fuzzy sets, fuzzy rules are derived using by the steepest descent method with equivalence and the performance results are compared with learning by using genetic algorithm. In order to compare the performance of learning, we construct a medical diagnostic system using the medical data used by the doctors for diagnosis, and compare the average of mean squared error, the correct answer rate and calculation time respectively. We use two division and three division models for the number of divisions in the antecedent part. The results obtain by learning using the genetic algorithm are better for the average of mean squared error and the correct answer rate in both the case of the two division and the case of the three division. Although the correct answer rate is not equal to genetic algorithm, the result is reasonably good, exceeding 80 percent. As the reason why the result of the steepest descent method is inferior to the result of genetic algorithm, the possibility of local optimal solution is considered. In genetic algorithm, the average of mean squared error and the correct answer rate are considered to be good because there is a way to escape from the local optimal solution by mutation. And, the calculation time is dramatically faster in the steepest descent method than in the learning time learned using the genetic algorithm. Further, when the number of input items is n and the number of divisions is m_i, the number of rules of conventional fuzzy inference models such as Mamdani's fuzzy inference model, the simplified fuzzy inference model, etc., is $\prod_{i=1}^{n} m_i$, while the number of rules of the SIC models is $\sum_{i=1}^{n} m_i$. In other words, it is expected that the number of rules can be drastically reduced. Therefor, the SIC model will be applied to large-scale data.

From the above results, knowledge can be expressed by obtaining fuzzy rules of the SIC model because of using consequent fuzzy sets. And, learning using the steepest descent method can be performed in a short time as compared with learning by genetic algorithm, it will be considered to be effective for application to big data. As future works, better results can be derived by appropriate division number of the antecedent parts in fuzzy rules and initial setting. Although all the widths of the consequent fuzzy sets are made same in this paper, it is also required to find conditions that allow learning. And the duplication number of the consequent part is learned without being restricted to double duplication.

REFERENCES

[1] L. A. Zadeh, "Fuzzy sets," *Information and Control* , Vol.8, pp. 338–353, 1965.

TABLE IV
CORRECT ANSWER RATE AND MEAN SQUARED ERROR WHEN EVALUATION DATA ARE TEACH DATA

data	mean squared error			correct answer rate		
	SIC model with CFS		Mamdani's model	SIC model with CFS		Mamdani's model
	SD	GA [9]	GA [9]	SD	GA [9]	GA [9]
1	0.01190	0.01027	0.04110	93.1	95.9	83.6
2	0.02200	0.00000	0.00685	87.5	100.0	97.3
3	0.02552	0.00000	0.05822	94.4	100.0	76.7
4	0.01478	0.00000	0.06849	86.1	100.0	72.6
5	0.01818	0.00000	0.01027	90.3	100.0	95.9
6	0.01825	0.00000	0.02055	80.6	100.0	91.8
7	0.02441	0.00000	0.01712	83.3	100.0	93.2
8	0.00793	0.00000	0.00000	94.4	100.0	100.0
9	0.01783	0.00000	0.03425	87.5	100.0	86.3
10	0.01325	0.00000	0.00000	88.9	100.0	100.0
average	0.01741	0.00103	0.02568	88.6	99.6	89.7

TABLE V
CORRECT ANSWER RATE AND MEAN SQUARED ERROR WHEN EVALUATION DATA ARE EVALUATION DATA

data	mean squared error			correct answer rate		
	SIC model with CFS		Mamdani's model	SIC model with CFS		Mamdani's model
	SD	GA [9]	GA [9]	SD	GA [9]	GA [9]
1	0.02477	0.00694	0.09375	83.3	97.2	62.5
2	0.02416	0.01389	0.03472	91.6	94.4	86.1
3	0.05423	0.02083	0.08333	80.5	95.8	66.7
4	0.04205	0.00347	0.08333	75.0	98.6	66.7
5	0.04486	0.01389	0.05208	76.4	94.4	79.2
6	0.03331	0.00000	0.03472	88.8	100.0	86.1
7	0.05991	0.02083	0.05208	75.0	91.7	79.2
8	0.03992	0.03125	0.04167	79.2	87.5	83.3
9	0.04308	0.00000	0.06597	83.3	100.0	73.6
10	0.04715	0.02083	0.01389	77.8	91.7	94.4
average	0.04134	0.01319	0.05556	81.1	95.1	77.8

TABLE VI
CALCULATION TIME

	SIC model with CFS		Mamdani's model
	SD	GA [9]	GA [9]
average	12.91(second)	1051.786(second)	27194.661(second)

[2] H. Ichihashi and T. Watanabe, "Learning control by fuzzy models using a simplified fuzzy reasoning," *Japan Society for Fuzzy Theory and System*, Vol.2, No.3, pp. 429–437,1990. (in Japanese)

[3] E. H. Mamdani, "Application of fuzzy algorithms for control of simple dynamic plant," *Proceedings of IEEE*, Vol.121, pp. 1585–1588, 1974.

[4] G. Ohashi, H. Seki, and M. Inuiguchi, "Knowledge Acquisition Using Fuzzy Inference Unified Max Operation and Its Application to a Medical Diagnosis System," *Proc. 2018 IEEE International Conference on System, Man, and Cybernetics*, pp.1767–1772, Miyazaki, Japan, 2018.

[5] Oscar Cordon, "A historical review of evolutionary learning methods for Mamdani-type fuzzy rule-based systems:Designing interpretable genetic fuzzy systems," *International Journal of Approximate Reasoning*, Vol.52, Issue 6, pp. 894–913, September 2011

[6] T. Fukuda, Y. Hasegawa, and K. Shimojima, "Structure Organization of Hierarchical Fuzzy Model using Genetic Algorithm" *Japan Society for Fuzzy Theory and System*,Vol.7, No.5, pp.988–996(1995) (in Japanese)

[7] H. Seki and M. Mizumoto, "On the equivalence conditions of fuzzy inference methods—part 1:basic concept and definition," *IEEE Trans. Fuzzy Syst.*, vol. 19, no. 6, pp. 1097–1106, 2011.

[8] H. Seki and M. Mizumoto, "On the equivalence of Mamdani model and fast calculation method of type-2 fuzzy inference model," *Proc. the 28th Fuzzy System Symposium*, pp.888–893, Nagoya, Japan, 2012.

[9] G. Ohashi, T. Shimizu, H. Seki, and M. Inuiguchi, "SIC Fuzzy Inference Model with Consequent Fuzzy Sets and Its Application to a Medical Diagnosis," *Proc. 2019 IEEE International Conference on System, Man, and Cybernetics*, Bari, Italy, 2019.(to appear)

[10] D. F. Andrews and A. M. Herzberg, *Data: A collection of problems from many fields for the students and research worker*, Springer, 1985.

Deep Learning Cookbook: Recipes for your AI Infrastructure and Applications

Sergey Serebryakov
System Architecture Lab
Hewlett Packard Labs
Palo Alto, CA, USA
sergey.serebryakov@hpe.com

Dejan Milojicic
System Architecture Lab
Hewlett Packard Labs
Palo Alto, CA, USA
dejan.milojicic@hpe.com

Natalia Vassilieva[1]
Product Management
Cerebras Systems
Los Altos, CA, USA
natalia@cerebras.net

Stephen Fleischman
AI Performance Engineering
Hewlett Packard Enterprise
Plano, TX, USA
stephen.fleischman@hpe.com

Robert D. Clark
HPC Storage
Hewlett Packard Enterprise
Houston, CA, USA
robert.d.clark@hpe.com

***Abstract*—Deep Learning (DL) has gained wide adoption and different DL models have been deployed for an expanding number of applications. It is being used both for inference at the edge and for training in datacenters. Applications include image recognition, video analytics, pattern recognition in networking traffic, and many others. Different applications rely on different neural network models, and it has proven difficult to predict resource requirements for different models and applications. This leads to the nonsystematic and suboptimal selection of computational resources for DL applications resulting in overpaying for underutilized infrastructure or, even worse, the deployment of models on underpowered hardware and missed service level objectives. In this paper we present the DL Cookbook, a toolset that a) helps with benchmarking models on different hardware, b) guides the use of DL and c) provides reference designs. Automated benchmarking collects performance data for different DL workloads (training and inference with different models) on various hardware and software configurations. A web-based tool guides a choice of optimal hardware and software configuration via analysis of collected performance data and applying performance models. And finally, it offers reference hardware/software stacks for particular classes of deep learning workloads. This way the DL Cookbook helps both customers and hardware vendors match optimal DL models to the available hardware and vice versa, in case of acquisition, specify required hardware to models in question. Finally, DL Cookbook helps with reproducibility of results.**

Keywords—deep learning, benchmark tool

I. Introduction

In the beginning of 2014, when Hewlett Packard embarked on its journey towards Memory Driven Computing and The Machine [1], software researchers attempted to find good applications where The Machine would perform exceptionally well. We started with algorithms and problems which were challenging to run on existing systems. Monte Carlo simulations, graph inference, search space optimization and deep learning were, among others, most promising. We started to model the performance of these algorithms for different system architectures. How will performance change if we have more powerful compute nodes or a faster interconnect between nodes in the system? For deep learning we soon realized that the answers largely depend on the topology of the artificial neural network. An optimal hardware configuration to train a deep convolutional neural network for detecting dogs and cats in images is not always the best one to train a fully connected model for detecting anomalies in datacenters. Depending on a model which you want to train (or to run inference in production), you'll need different hardware to minimize training or inference time. Similar to cooking, depending on what you cook, you need different ingredients in different proportions. Neural network models come in different flavors, some of which are listed in table 1. Models differ from each other in many dimensions including structure of a compute graph, number of trainable parameters, compute cost (number of floating point operations - FLOPs) for forward (inference) and backward (gradient computations) passes, and others.

TABLE I. Neural Network Models

Model	Parameters (millions)	gFLOPs (forward)	FLOPs per parameter	Size (MB)
AlexNet	61	1.4	23	233
ResNet-50	26	8.2	315	98
ResNet-152	60	24	400	230
GNMT	244	24.5	100	933
Transformer	213	13.56	64	813
BERT	340	366.5	1100	1300
GPT-2	1500	3507	2254	5934
3D U-Net	17.3	657	38000	66

In parallel with our Memory Driven Computing efforts, artificial intelligence and deep learning started to gain interest from enterprise customers. Through our interactions with customers, we encountered questions about the choice of optimal hardware/software environment for training neural networks in data centers and about using them to run inference on edge devices (we use the term *deep learning (DL) workloads* to refer to both training and inference). How to choose from a plethora of available options today? How to size and configure

[1] Currently with Cerebras Systems, work done while at Hewlett Packard Labs.

978-1-7281-5221-9/19/$31.00 ©2019 IEEE

infrastructure? Along with the rise of customer interest in DL workloads, we observed an increased demand in running DL benchmarks internally in response to customer requests, as well as running *HW integration, qualification and validation tests*. A need for the DL Cookbook became clear.

In the process of developing the Cookbook, we constructed analytical models to predict the performance of various machine learning algorithms (including deep learning) depending on compute power and the number of compute nodes in a cluster, as well as properties of the interconnect between the compute nodes [2]. These simple models were useful for rough estimates, but did not take into account many of the subtleties of computer systems. Therefore we had to introduce actual benchmarking, in addition to analytical modeling, and to reason based on real performance data.

We wanted to be able to collect performance data on various hardware systems, deep learning frameworks, and deep learning workloads. We wanted the results to be consistent, reproducible and comparable. This meant we needed to make sure that we ran the exact same workloads on multiple systems. For this to work, we needed a benchmarking tool. We had several options:

- Use existing projects that target the deep learning domain. Some of the more recognizable community projects included *DeepBench* [3] from BAIDU and the *convnet-benchmarks* [4]. These projects aim at benchmarking low-level functionality such as convolution, matrix multiply operations, or use simplified models.
- Use example training scripts that are part of their respective deep learning frameworks, something what most companies used to do. All major deep learning frameworks such as TensorFlow, MXNET, PyTorch, and Caffe2 among others provide example scripts to train widely used neural networks such as ResNet50 or Inception.
- Use TensorFlow benchmarks [5] for convolutional neural networks that target image classification workloads. While this would have been an easy choice, we determined we needed to support multiple frameworks beyond just TensorFlow and this particular type of DL workload.

Unfortunately, none of these options provides a way to collect performance data for different deep learning workloads in a consistent and reproducible manner across a range of software and hardware combinations. Thus, we decided to create our own collection of tools that we call the Deep Learning Cookbook [6]. DL Cookbook enables users to run reproducible and consistent deep learning benchmarks and analyze performance results using a web based application.

The Deep Learning Cookbook is a collection of tools consisting of several key assets:

- The first is the HPE Deep Learning Benchmarking Suite (DLBS) [7]: an automated benchmarking tool to collect performance measurements on various HW/SW configurations in a unified way. An introduction to DLBS, including architecture and examples is available online [8].
- The second is the HPE Deep Learning Performance Guide (DLPG) [9]: a web-based tool which provides access to a knowledge base of benchmarking results. It enables querying and analysis of measured results as well as performance prediction based on analytical performance models.

In this paper we present our approach to running DL workloads for benchmarking and hardware/software validation and qualification testing. We introduce the DL Cookbook, provide a description of our benchmarking protocols, present some of the interesting performance results and describe our experience of using the DL Cookbook in the enterprise environment. The paper is organized as follows. Section 2 introduces DL training workloads, our benchmarking protocol and some of the performance results. In section 3 we analyze in detail our experience of using DL Cookbook. In section 4 we review related work and in section 5 we provide summary and future work.

II. ANALYSIS OF DEEP LEARNING TRAINING PIPELINES

DL training workloads involve complex multi-stage computations that usually run in heterogeneous environments – a diverse set of compute devices involved in computations, usually CPUs and accelerators such as GPUs, TPUs or FPGAs.

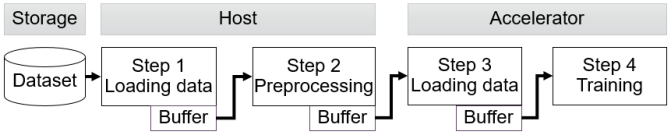


Fig. 1. High level overview of a DL training pipeline.

Fig. 1 provides a high level overview of a deep learning training pipeline. We provide a brief description here while a detailed explanation can be found in [10]:

Step 1. A typical DL pipeline starts with loading a portion of a training dataset into host memory. Modern datasets tend to be large, so keeping all data in memory is not always possible. Data loaders are usually multi-threaded components and run on CPUs.

Step 2. In most DL training workloads, data needs to be pre-processed. For instance, images may need to be resized to appropriate dimensions and augmented in order to enrich training data and make it more diverse. Similar to data loaders, preprocessing components are multi-threaded and usually run on CPUs. We'll discuss some scenarios where it's better to run these components on more performant devices such as GPUs.

Step 3. A subset of preprocessed data (called a batch) is copied into an accelerator's memory for one or more devices. In the latter case, a batch is usually split into equal chunks and each chunk is copied to a different accelerator.

In most efficient implementations, steps 1, 2 and 3 run as independent, overlapping processes on CPU or GPU cores in parallel, communicating with each other using queues. This method is called pipelining [10] and results in better overall performance by hiding data ingestion and utilizing accelerators more efficiently. In certain scenarios, accelerators are used to preprocess data (e.g. NVIDIA DALI [11]) (step 2) in which case step 3 becomes unnecessary. Environments where a CPU can be a bottleneck (such as multi-user systems and/or clusters comprised of nodes with large numbers of accelerators) can benefit from such schema.

Step 4. DL frameworks generally train neural networks using one of the variants of the mini-batch gradient descent (GD) optimization algorithm. In a standard iteration of a mini-batch GD, a DL framework first performs a forward pass – makes a prediction by computing outputs of a neural network for a small subset of training data, a batch. It then computes an error on the predictions using a loss function. The gradient (derivatives) of the error with respect to adjustable parameters are calculated using the calculus chain rule (backward pass). The gradients are then used to update network parameters to reduce prediction error. Training data for each batch is drawn randomly from a training dataset and data in one batch does not depend on data in another batch. Thus, this one step of a GD can run independently from data ingestion steps 1-3 as long as there's a new batch available.

When multiple accelerators are used, step 4 looks slightly different. We will be considering data parallel computations as the most common option (additional options include model parallelism and layer pipelining) and an overview of these approaches can be found in [10]. When running data parallel training, model replicas running on each accelerator need to be synchronized. This is usually done by summing the gradients across accelerators using some form of ring-based all reduce function. A key feature here is that frameworks can aggregate gradients in parallel with a backward pass, thus overlapping computation and communication. This is one of the key contributors to near-linear scaling for certain workloads.

Some of the interesting questions that one may ask regarding the DL training pipeline are the following:

1. When does a data loader become a bottleneck?
2. When does preprocessing become a bottleneck?
3. What is the role of the interconnect between accelerators?
4. What is the role of the interconnect between the host and accelerators?
5. Can scale-out be as efficient as scale-up?

A key characteristic of a training pipeline that determines the overall effectiveness is the computation time. If we assume that it takes *T* milliseconds to perform computations, then as long as the time it takes to prepare the next batch and the time for aggregating gradients are less than *T*, a pipeline operates in an optimal regime as the compute devices are not sitting idle while waiting for future batches to be prepared and for model updates to synchronize. This in general means that compute-intensive models such as deep convolutional neural networks should normally operate in this regime.

We have developed a benchmark protocol that allows us to consistently benchmark DL workloads and identify bottlenecks. We start by running a benchmark using synthetic data stored in an accelerator's memory. In the world of deep learning, a benchmark with synthetic data is a benchmark without data ingestion pipeline. It is usually implemented in form of fixed neural network inputs that are randomly initialized in advance. Using synthetic data is a standard approach to eliminate overhead associated with the data ingestion. Having data in an accelerator's memory means host-to-device transfers do not affect the performance which we consider to be the best possible performance. We then move synthetic data into the host's memory and benchmark this configuration. If we get the same results, host-to-device transfers are not a bottleneck. The next step is to use multiple accelerators with synthetic data. At this step we can identify whether the accelerator interconnect becomes a bottleneck, and if so, at which point in the process. Then, we replace synthetic data with real data and activate pre-processing. The new data is still placed in the host's memory, enabling us to identify when data pre-processing is a bottleneck. Our final step is to copy data to a storage under test (either a local SSD/NVMe drive or some kind of external parallel filesystem) and run tests.

In the following sub-sections, we briefly present our findings regarding these questions and demonstrate what kind of analysis can be done with the DL Cookbook.

A. Data ingestion for inference workloads

A DL pipeline operates in an optimal regime as long as the data ingestion, accelerator-to-accelerator communication and computations are perfectly overlapped. In this section we analyze the impact storage has on DL performance. In particular, we consider a datacenter inference workload - a large scale model validation scenario involving a distributed parallel filesystem. This scenario is critical for companies working on self-driving vehicles since they need to validate many models per day on large datasets. Inference is not as compute intensive as training, and thus our expectation is that storage should have a significant impact.

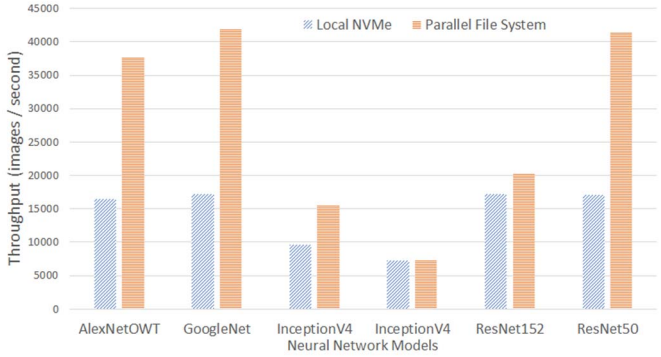


Fig. 2. Inference performance on a 8 GPU server with TensoRT inference engine. Data is from DLPG, standard report 10.

To test this theory, we used an 8-GPU server with NVIDIA V100-SXM2 GPUs and NVIDIA's TensorRT inference engine with models converted to use half precision. Since this is a throughput-oriented workload, we report the number of images processed per second.

Fig. 2 presents inference performance results using 8 GPUs. The number of per-GPU images (replica batch size) is tuned to achieve the best throughput, and is usually the maximal batch that can fit into a GPU's memory. Since preprocessing data (image decoding and resizing/cropping) as part of the benchmarking workflow may be a bottleneck at these inference rates, we preprocessed the common ImageNet dataset prior to running the benchmark and stored the preprocessed data in large binary files on our filesystem (this is acceptable for inference as opposed to training where online data augmentation results in better model accuracy). We also ensured that system caching is disabled and does not impact the performance. The inference

benchmark results show that the IO demands to 8-GPU servers can extend beyond the capability of local NVMe drives (fast storage drives attached via the PCI express interface). The impact of a parallel filesystem is not significant for very compute intensive models such as Inception4, but in the case of smaller models such as ResNet50 the difference may be far greater than 2x depending on the performance of the local drive(s).

To characterize the performance of 8 GPUs, we need to determine a baseline performance and understand how performance scales as we increase the number of accelerators. For our baseline, we chose to model the performance of GPUs

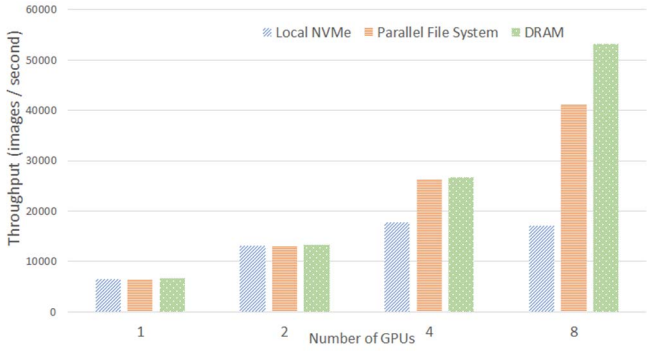


Fig. 3. Scaling ResNet50 inference workload across multiple GPUs. Data is from DLPG, standard report 11.

when data resides in memory (DRAM). Fig. 3 compares the baseline performance of the ResNet50 neural network and shows how ResNet50 inference workloads scale across GPUs. The results shows that a local NVMe drive can support up to 2 GPUs in this particular configuration, but the IO becomes a bottleneck when scaling beyond this point.

B. *Interconnect between accelerators*

As we discussed above, it is usually easier to scale compute intensive models. However, if a model is not as compute

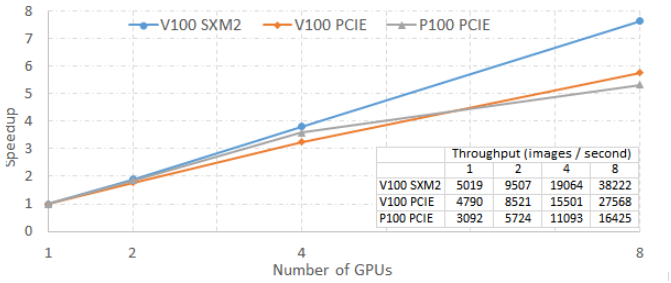


Fig. 4. Scaling of the AlexNet neural network training in various 8-GPU systems [12]. This benchmark uses synthetic data to eliminate effects of the data ingestion pipeline and isolate component under test – GPU interconnect.

intensive, and as the number of trainable parameters increases, scaling may not be perfect. Fig. 4 presents training speedup for the AlexNet neural network in various 8-GPU systems (configuration details can be found in DLPG [12]). Speedup is measured to be the current throughput divided by the throughput achieved with 1 GPU. AlexNet requires approximately 1.4 gFLOPs (or 0.7 FMACs, multiply-adds) in a forward pass per one training example (image). With 61 million trainable parameters (totaling 233MB in size), this network requires a high bandwidth interconnect between GPUs. With PCIe interconnect, the V100 cards do not scale as well as their SXM2 counterparts, implying fast accelerators need to come with fast interconnects for certain models.

In this experiment, the replica batch size (per-GPU size of input data), was 1024 images. Large batch sizes increase throughput, but usually result in worse model accuracy or longer time to achieve a certain level of accuracy; though this can often be ameliorated by using adaptive optimizers such as LARS [13] and LAMB [14]. We intentionally used the relatively large batch size to keep GPU computations at a moderate level, leaving more time for GPU-to-GPU communications, and thus, not saturating the interconnect bandwidth. Smaller batch sizes will put more stress on the interconnect and will be of greater importance in determining performance for non-compute intensive neural networks.

C. *Scale-up vs scale-out*

Another interesting topic of investigation is scaling-up vs scaling-out. Scaling-up means adding more accelerators into one server node while scaling-out means adding more nodes. There are pros and cons of each of these approaches. The tradeoff between scaling-up and out is between the ease of use of the single node approach versus the increased hardware complexity and cost that comes with increasing the number of GPUs in a single server node.

It is generally easier to run single-node training sessions, since one does not have to set up and maintain heterogeneous communication between the server nodes and individual accelerators. However a monolithic single node architecture has its limitations in terms of hardware complexity and cost, as mentioned above, and with flexibility when dealing with multiple workloads with differing requirements. A compute cluster may be built with servers with different number of GPUs and variety of options exists in the market today ranging from one to sixteen GPUs per node. In one situation a cluster of 4-GPU servers may be suitable, while 8-GPU servers would be a better choice in others. We ran the DL Cookbook in various distributed configurations, and the one we are presenting here (fig. 5) is from DLPG, standard report 13. Training performance of a moderately compute intensive neural network (ResNet50) is compared in different hardware

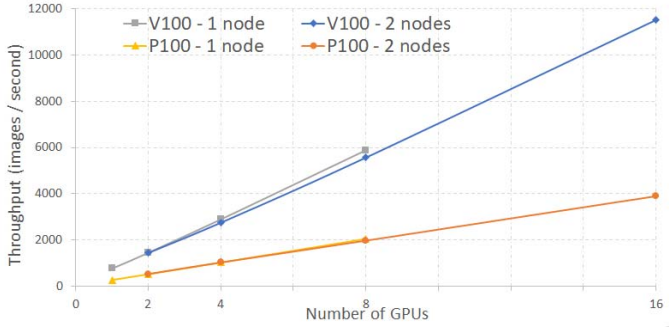


Fig. 5. Scale-up vs scale-out. Training ResNet50 with TensorFlow using one and two servers each having 8 GPUs. Results are from DLPG, report #13.

configurations. In one configuration, one and two nodes are used with eight P100-PCIe GPUs in each node. We fixed the number of GPUs and vary the number of nodes. For instance, in a point with 8 GPUs, one configuration corresponds to a single node with 8 GPUs, and the other configuration corresponds to two nodes with 4 GPUs each. Even though a relatively slow interconnect between GPUs is used (PCIe), there is no difference in performance (as opposed to the AlexNet model that was studied in the previous section). This is because in this particular configuration, TensorFlow can aggregate gradients within a time interval required to train one batch, and thus, perfectly overlaps computations and communications. We observe the same situation in a configuration with V100-SXM2 GPUs. Even though there is a small difference in performance, numbers are close to each other. The conclusion of these benchmarks is that given the right network (InfiniBand) and software stack, scale-out and scale-up training performance may be close to each other.

III. LESSONS LEARNED

We have been using the DL Cookbook internally since the beginning of 2017. It was released under an open source license (Apache 2.0) in December of the same year. In this chapter, we present our experience and the lessons that we have learned, in particular, we discuss the following topics: ease of use, reproducibility, benchmarks history, modular architecture and advanced benchmarks. This chapter also sheds light on certain aspects that are critical and important for a tool like DL Cookbook to be successful and useful in an enterprise environment similar to ours.

A. Ease of use.

Ease of use is essential for adoption by both ML/DL experts and performance engineers. The following features were important to achieve this goal.

Containers. Containers [15] provide a technology to package applications with their complete runtime. Two popular container runtimes are Docker and Singularity. The former has become a de-facto standard for packaging AI applications while the latter is used for HPC applications due to a better security model. From the start it proved advantageous to make Docker/Singularity containers our main runtime environment. In the beginning, the HPE internal benchmarking and test infrastructure was not ML/DL friendly and was mostly tuned for running HPC benchmarks. In such environments, it was hard and extremely time consuming to build or install various DL frameworks largely, but not entirely, due to restricted user permissions. We immediately realized the potential of containers and quickly added a collection of Docker files (container specifications) into DL Cookbook that became our main runtime environment. Later, when NVIDIA announced and started to provide optimized Docker containers via NVIDIA GPU Cloud (NGC), we started to use them as our default runtime environment for NVIDIA accelerators. For other devices (Intel CPUs and AMD GPUs) we continue to provide reference Docker files.

Using containers resulted in various improvements: 1) we achieved a better reproducibility by providing standard runtimes on various machines; 2) the DL Cookbook can easily switch between different runtimes benchmarking various versions of the same DL framework, supporting libraries etc.; 3) there is substantial time saving in setting up a new server; 4) non ML/DL experts can just run a bash script that pulls or builds a Docker container for them; 5) we support containers from accelerator manufactures that achieve a state-of-the-art performance result.

In addition to vendor specific and optimized containers, we find it very convenient to provide reference Docker files to establish a baseline performance; to have frameworks compiled with support for particular, non-standard functionality; to better understand impact of software optimizations coming from HW vendors; and to have step-by-step instructions on how to compile and build frameworks.

Challenges. At the moment, only NVIDIA provides a good collection of Docker containers optimized for their accelerators. We would like to see something similar from other vendors as well. Hosting an NGC type service may require substantial investment, however, providing well-documented Docker files is a good alternative option.

Unified command line interface. Users of the DL Cookbook do not need to interact with DL frameworks and/or specific DL scripts. Rather, they interact with the DLBS that runs benchmarks for them. Everything is parametrized (framework, runtime, neural network model…) and users request to run a certain workload or workloads by configuring the DLBS either on a command line or by providing a JSON configuration file. The DL Cookbook unifies parameters across various frameworks and benchmark back ends as much as possible, so switching between them is relatively easy. We have a collection of pre-defined configurations suitable for various servers and use cases, so running benchmarks and tests for standard scenarios is straightforward.

Challenges. Configuring benchmarks may be non-intuitive due to a large number of parameters. Any benchmark tool needs good configuration handling, particularly it needs to provide: 1) advanced configuration validation and checking –it needs to catch and report configuration errors and incompatible settings; 2) it needs an intuitive way to configure multiple benchmarks and simple ways to override default values for parameters.

B. Reproducibility.

Reproducibility is critical for users. The DL Cookbook achieves reproducibility by using standard runtimes (containers); by versioning code using GitHub hash tags and; by using standard configuration files that define benchmark settings.

Variety of implementations. As described in section 2, the DLBS implements a core functionality that runs benchmarks using benchmark back ends that represent separate projects. At the level of these benchmark back ends, reproducibility is enforced by using a specific version of a benchmark back end, a GitHub hash tag. This tag can be specified by a user to use particular version of a code.

Challenges. We see a need to support at least two ways to run benchmarks. The first one is when benchmark code is located in host OS and containers provide DL runtime. The

second one is when benchmark code is placed inside containers what enables running benchmarks in environments managed by such tools as Kubernetes.

Neural network models. To enforce apple-to-apple comparison across frameworks, we re-implemented NN models that are supported by DLBS from scratch in each benchmark back end. Thus, ResNet50 training performance with TensorFlow is truly comparable to MXNET or PyTorch.

Challenges. Implementing the same model in different frameworks requires time and human efforts. Also, it requires support such as code updates due to frameworks' API change. Originally, we were going to store models in one format and then convert them to a framework-specific representation using specialized converters. We experimented with several converters and found out that performance of converted models was not as good as the performance of models implemented using native framework API. With the recent development of ONNX format, we are considering it as a unified representation across all frameworks.

C. *Benchmark history.*

It is important to keep benchmark results along with details on HW configuration. This enables various types of analytics to be conducted, for instance, in order to determine optimally performing HW/SW configurations. Additionally, lack of knowledge of the specific HW configuration that was used in a benchmark may prevent users from replicating our results. It is also not only extremely useful, but often critical, that we retain the output of monitoring tools that record such parameters as CPU/GPU utilization, memory usage, power draw etc.

In order to meet the aforementioned requirements and goals, we implemented several modules. Firstly, we implemented and deployed a web based application, the Deep Learning Performance Guide, as described previously. We also added a reference implementation of a resource monitor module that records hardware usage statistics in benchmark log files and added a number of scripts that collect system information such as CPU and GPU models and their parameters and many others.

Challenges. Tools like DLPG have proven useful for providing high level performance results, however they do not provide a complete view of HW/SW infrastructure. They are suitable for promotional and marketing purposes, for quick overview of results, but not for serious technical analysis. An embedded resource monitor can provide some insights, but in general there is a need for a better monitoring tool, ideally using some of the existing open source tools, such as CollectD. Additionally, performance engineers need a structured and consistent approach to collect system information so that each aspect of the software and hardware environment is preserved for further analysis.

D. *Modular architecture.*

From the very beginning the DLBS was designed to be modular i.e. to support various benchmark back ends (fig. 6). In some sense, the DLBS core is a benchmark controller. The very first benchmark back end that became part of the DLBS is Google's TF CNN benchmarks project. Later, other open source projects were integrated: NVIDIA's nvcnn and Google's Tensor2Tensor. Since the DLBS provides common set of core parameters for all benchmark back ends (model name, batch size etc.), it's relatively easy for users to switch between different benchmark back ends, including those that were not developed by us. Benchmark back ends can run on different runtimes – various Docker containers or bare metal We follow a very simple policy to integrate third-party projects – when we realize that we need to run particular benchmark on a regular basis, we integrate that into DLBS. There are several benefits for doing so: 1) we take advantage of community efforts and integrate well-known benchmarks and 2) having multiple benchmark back ends for the same framework, we can use the one that implements best practices and provides best performance. For instance, when HPE released a beta version of a server with eight NVLINK GPUs, TF CNN benchmarks did not scale well to all 8 GPUs, but NVCNN did. So we quickly switched to a new TensorFlow back end and continued to test hardware without any delays.

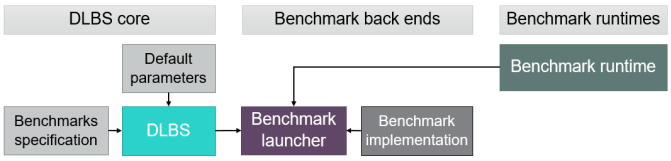


Fig. 6. Architecture of the Deep Learning Benchmarking Suite.

Challenges. TensorFlow is the only framework for which DLBS does not have a native benchmark back end – mostly because since the start of the project, Google's and later NVIDIA's implementations of benchmark projects have been available. For all other frameworks we use our own implementations that are part of DLBS. This has resulted in unbalanced implementations of benchmark projects that have their own pros and cons.

The benchmark back ends that we have created in DLBS (MXNET, PyTorch, Caffe, Caffe2 and TensorRT), while providing minimal functionality, are very easy to adjust when APIs change. Conversely, the Google TF CNN Benchmark back end is quite a large project and it usually takes much more time to do so, for instance, to integrate a new version in DLBS or to adjust to a new container version. This used to be the case for the NVIDIA NVCNN TensorFlow back end but is no longer quite the case as it has been modularized.

Another consideration is that community driven benchmark projects are often fully fledged implementations that support much more functionality, including end to end metrics, especially time to accuracy. Our implementations do not currently support advanced metrics. We are actively working on expanding the range of supported benchmark back ends to cover additional methods, models and applications as well as expanding the types of metrics and tests that we can support, such as time to accuracy and multi-user throughput.

E. *Advanced benchmarks.*

Single-metric tool. The DLBS was designed to run a large number of various DL workloads to qualify, validate and benchmark hardware, software and firmware on a regular basis in diverse environments. For these purposes, the only metric that matters is raw performance – like images per second or throughput per watt. Achieving good results with end to end

metrics such as time to accuracy requires excellent understanding of deep learning theory and best practices and does not introduce significantly more valuable information to our testing routines. For almost a year and a half that we have been using the DLBS, only a small fraction of workloads required end-to-end results. Typical examples are marketing events and publications and customer-related tests (responding to their RFPs).

Challenges. We anticipate that end-to-end benchmarks will play more significant role in our day to day work, given the popularity of MLPerf community initiative and the emphasis place on them by luminaries in the field such as Andrew Ng. One use case for long end-to-end benchmarks is the demonstration that as training progresses, there's no degradation in raw performance due to, for instance, overheating. Currently support for end-to-end metrics is something that DLBS is missing. However, integrating new performance metrics, such as time to accuracy or performance per watt, will not require significant re-design. Moreover, for certain benchmark back ends such as existing Google's TF CNN benchmarks or Tensor2Tensor, NVIDIA's nvcnn or future MLPerf benchmark back ends, support for end-to-end metrics is already there and need only be exposed by the DLBS configuration schema.

Single-focus tool. The DLBS is a tool that does one thing and it does it very well – running a performance benchmark or benchmarks. These benchmarks stress various components of hardware under the test and assume a single-user working environment. Also, some of the use cases that the DLBS assumes, for instance, when running inference tests, are only a subset of real world use cases. In particular, in the inference benchmark regime, the DLBS assumes single-stream large scale model validation scenario which is similar to MLPerf's [16] *offline* inference scenario.

Challenges. As we expand the areas where we use DL Cookbook, and as more and more colleagues and customers get familiar with the tool, new use cases emerge that are not currently supported by DLBS. In particular, the following quite important workloads are out of scope and are not supported.

- *Multi-tenant benchmarks*. Extremely important cloud/data center model training use cases that help to better size AI/DL infrastructure.
- *Application benchmarks*. For workloads such as video analytics (object detection inference), understanding that a system can process N frames per second is not enough and does not offer customers much information. A new set of metrics is required, such as number of streams a system can process in real time.
- *Additional inference use cases*. Such use cases as multi-stream inference are not supported by DLBS now.

Variety of neural network models. From the first day, the DL Cookbook has been supporting a significant number of convolutional neural networks. We had to spend some time re-implementing these models in every benchmark backed for each supported framework. We also added several fully connected models, one RNN model for speech recognition and one LSTM autoencoder.

Challenges. One of the goals of the DL Cookbook is to be able to characterize DL workloads and to recommend an optimal HW/SW stack for running each of them. Another goal is to be able to put enough stress on various components of a system – storage, CPU, accelerator, interconnect, etc., to understand potential system bottlenecks. In order to do so, we cannot afford to limit the tool to supporting for one particular neural network architecture, such as CNNs. Rather, we need a diverse collection of neural network architectures each introducing a unique computation/communication pattern. The DLBS is currently deficient in support for RNN models, object detection models and newly emerged architectures like Transformer and BERT. We are actively engaged in ameliorating this situation, leveraging current open source implementations where possible.

IV. Related work

Researchers have been considering multiple approaches for characterizing DL workloads. From a high level point of view, these efforts can be split into two categories – prediction models for estimating the performance of DL workloads and benchmark tools for measuring that performance.

There are several approaches to building prediction models. In the first approach [17], researchers have built *machine learning-based* models for various neural network functions like activations, convolution, and matrix multiply etc. for predicting performance of individual operations. In the second approach, *analytical models* have been employed for estimating performance of various phases like computations in neural network layers or prediction performance of a communication (aggregation) phase [2], [18], [19], [20]. From an historical perspective, the paper by Qi, Sparks, Talwalkar on Paleo [20] and the visualization tool that they developed inspired us to develop DLPG [9].

In their prediction model, the overall performance is the sum of performance of individual operations. Even though the reported numbers are acceptable in most cases, there are several challenges associated with only using prediction models. The models become ever more complex as we introduce hardware properties, such as frequencies and BIOS settings. Furthermore, a number of challenges is introduced by the software stack as various BLAS libraries, driver versions etc. impact performance. The other source of variation is the DL framework itself. As noted in [10], frameworks perform a number of optimizations and select the best suitable compute kernel for a given hardware. BLAS libraries can also adjust and select specific kernel optimized, for instance, for dimensions of input matrices. No matter what prediction model is used, this all needs to be taken into account to accurately model the performance.

A more popular approach to characterizing DL workloads is to run benchmarks and measure the actual performance. A number of benchmarks and frameworks have been introduced within past two year focusing on various aspects of benchmarking process.

To put our work into a historical context, at the time we started the development of the DL Cookbook, Fathom [21] was probably the closest tool that introduced a set of reference DL workloads with code available on GitHub. Their focus was on TensorFlow and they did not support distributed execution;

which for us was an absolute requirement. Another paper that to some extent influenced our work is [22], whose authors extensively tested multiple DL frameworks: CNTK, Torch, Caffe, MXNet and TensorFlow. In their paper they presented tables with performance numbers for a variety of frameworks all running the same *standard and consistent* neural network models, e.g. ResNet50 in TensorFlow is the same as ResNet50 in Caffe2 and we decided to emulate this approach. In summary, Paleo [20], Fathom [21], [19] and requirements from HPE performance engineers formed the foundation for the DL Cookbook. Our year and half of experience in using the DL Cookbook has proven the right choice.

Since then the DL community and researchers in the field have proposed a number of standard benchmarks and frameworks. Notably, MLPerf [16], TBD [23] and the most recently Deep500 [24] are particularly relevant to our work.

MLPerf [16] is a community driven initiative supported by major companies to establish standard Deep Learning benchmarks by introducing strict rules and reference DL workloads. The MLPerf benchmark suite is not large, but it is diverse and represents well the various computation and communication patterns that are found in various NN models and verticals. Its focus is end-to-end performance i.e. time to accuracy and to some extent, it is a competition between researchers in the first place. Each benchmark, being training or inference, is documented and is accompanied by a launching script what makes it possible to easily integrate with the DL Cookbook. We plan to do so when we see the need to integrate the MLPerf benchmarks into our everyday performance evaluation workflows.

TBD, Training Benchmark for DNNs, [23] is another attempt to introduce a collection of DL reference workloads. Their workloads include image classification, machine translation, speech recognition and some others. Implementations exist for a number of frameworks, such as TensorFlow, MXNet and CNTK, albeit each workload is not implemented in all of the frameworks. It's another good source of model implementations and projects that may become DLBS benchmark back-ends at some time.

One concern regarding integrating various third-party benchmark back-ends is that there must be somebody to support them. Currently, our benchmark back-ends are small and are focused on achieving good raw performance (for instance, samples/sec). Basically, this requires implementing a model and a simple training loop. It's a simplified view, but makes it possible to quickly upgrade to new containers and framework versions - a fundamental requirement for our work. Furthermore, we require workload implementations for all supported frameworks - a requirement that is difficult to achieve with complex, end-to-end, training logic, e.g., for something like object detection or various sequence applications and models.

Deep500 [24], introduced recently, is another attempt to propose benchmark tools and infrastructure for DL workloads in HPC environments with workloads ranging from the testing of low level operators all the way to end-to-end training. This is a very similar project to DLBS in that introduces a unified Python API for evaluating datasets, frameworks and models. At this time DLBS does not have a good support for a programmable Python API, rather it relies on a command line interface that accepts an extensively configurable JSON file input.

V. Summary and Future work

DL has been proven to achieve and surpass state of the art results in many areas including image classification, speech recognition, language modelling and anomaly detection. This has made DL an important workload for various systems deployed in the cloud or on-premises and has challenged us to answer four questions:

1. What is the most suitable hardware and software stack for a given DL workload?
2. Where are the bottlenecks?
3. How to design next generation hardware to better address DL computation and communication patterns?
4. How to prove that a particular system can train various neural networks fast and without performance degradation?

To address all these challenges, we created the DL Cookbook. It provides a unified command line interface and a Python API for all supported DL frameworks and models. It exposes most of the commonly used DL parameters such as batch size and GPU count. It supports performance analysis and bottleneck identification by exploring various values of a single or multiple parameters.

We address the **first question** by running a comprehensive set of benchmarks with various DL frameworks, neural networks and hardware and software configurations. The results from these experiments provide insights into DL computation and communication patterns and enables us selecting the right configuration. We address the **second question** by isolating individual parameters such as batch size and running a large number of benchmarks. From this the DL Cookbook discovers configurations under which a certain HW components such as PCIe lanes, CPU, GPUs or interconnect become a bottleneck. We address the **third question** by analyzing a large number of workloads and their performance profiles to drive the software and hardware co-design for our next-generation computing systems. We address the **forth question** by running the most compute intensive configurations that stress various components (compute devices, network or storage).

We have been using the DL Cookbook internally and externally together with our customers for about 1.5 years. We have identified its strong and weak points that we plan to address in the future releases. In particular, we plan to 1) significantly improve and simplify multi-node performance tests; 2) add better monitoring tools to measure memory and device utilizations; 3) add new workloads that have emerged recently.

Acknowledgments

We thank the many colleagues in Hewlett Packard Enterprise who helped and supported us. In particular, Sorin-Cristian Cheran has been helping us from the very beginning; Al Amin has actively been promoting DL Cookbook inside HPE and; we thank Aalap Tripathy for great advice and numerous discussions. We also thank marketing teams for organizing DL Cookbook presentations and demos at various internal and external events.

REFERENCES

[1] K. M. Bresniker, S. Singhal, R. S. Williams, "Adapting to Thrive in a New Economy of Memory Abundance," *Computer*, vol.48, no. 12, pp. 44-53, 2015.

[2] A. Ulanov, A. Simanovsky, and M. Marwah, "Modeling scalability of distributed machine learning," in *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. IEEE, 2017, pp.1249-1254.

[3] "Deep Bench," 2016. [Online]. Available: https://github.com/baidu-research/DeepBench. [Accessed: 2019-05-20].

[4] "ConvNet Benchmarks". [Online]. Available: https://github.com/soumith/convnet-benchmarks. [Accessed May 20, 2019].

[5] "TensorFlow CNN benchmarks". [Online]. Available: https://github.com/tensorflow/benchmarks. [Accessed May 20, 2019].

[6] "HPE Deep Learning Cookbook," 2017. [Online]. Available: https://developer.hpe.com/platform/hpe-deep-learning-cookbook/home. [Accessed May 20, 2019].

[7] "HPE Deep Learning Benchmarking Suite," 2017. [Online]. Available: https://github.com/HewlettPackard/dlcookbook-dlbs. [Accessed May 20, 2019].

[8] "HPE Deep Learning Benchmarking Suite online documentation", 2018. [Online]. Available: https://hewlettpackard.github.io/dlcookbook-dlbs. [Accessed May 20, 2019].

[9] "HPE Deep Learning Performance Guide," 2018. [Online]. Available: https://dlpg.labs.hpe.com. [Accessed May 20, 2019].

[10] T. Ben-Nun and T. Hoefler, "Demystifying parallel and distributed deep learning: An in-depth concurrency analysis," *ACM Computing Surveys (CSUR)*, vol. 52, no. 4, p. 65, 2019.

[11] "NVIDIA Data Loading Library". [Online]. Available: https://developer.nvidia.com/DALI. [Accessed Sep. 3, 2019].

[12] "Scaling of the AlexNet neural network in a 8-GPU system". [Online]. Available: https://dlpg.labs.hpe.com/report=aba607b1993c426485ba83ed47bd1bf4. [Accessed May 05, 2019].

[13] Y. You, I. Gitman, and B. Ginsburg, "Large batch training of convolutional networks," *arXiv preprint arXiv:1708.03888*, 2017.

[14] Y. You, J. Li, J. Hseu, X. Song, J. Demmel, and C.-J. Hsieh, "Reducing bert pre-training time from 3 days to 76 minutes," *arXiv preprint arXiv:1904.00962*, 2019.

[15] D. Merkel, "Docker: Lightweight linux containers for consistent development and deployment," *Linux J.*, vol. 2014, no. 239, Mar. 2014. [Online].Available: http://dl.acm.org/citation.cfm?id=2600239.2600241. [Accessed May 20, 2019].

[16] "MLPerf. a broad ML benchmark suite". [Online]. Available: https://www.mlperf.org. [Accessed May 14, 2019].

[17] D. Justus, J. Brennan, S. Bonner, and A. S. McGough, "Predicting the computational cost of deep learning models," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 3873-3882.

[18] F. Yan, O. Ruwase, Y. He, and T. Chilimbi, "Performance modeling and scalability optimization of distributed deep learning systems," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '15. New York, NY, USA: ACM, 2015, pp. 1355–1364. [Online]. Available: http://doi.acm.org/10.1145/2783258.2783270. [Accessed May 20, 2019].

[19] S. Shi and X. Chu, "Performance modeling and evaluation of distributed deep learning frameworks on gpus," in *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*. IEEE, 2018, pp. 949-957.

[20] H. Qi, E. R. Sparks, and A. Talwalkar, "Paleo: A performance model for deep neural networks," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[21] R. Adolf, S. Rama, B. Reagen, G. Wei, and D. M. Brooks, "Fathom: Reference workloads for modern deep learning methods," in *2016 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2016, pp. 1-10.

[22] S. Shi, Q. Wang, P. Xu, and X. Chu, "Benchmarking state-of-the-art deep learning software tools," in *2016 7th International Conference on Cloud Computing and Big Data (CCBD)*. IEEE, 2016, pp. 99-104.

[23] H. Zhu, M. Akrout, B. Zheng, A. Pelegris, A. Phanishayee, B. Schroeder, and G. Pekhimenko, "TBD: benchmarking and analyzing deep neural network training," *arXiv preprint arXiv:1803.06905*, 2018.

[24] T. Ben-Nun, M. Besta, S. Huber, A. N. Ziogas, D. Peter, and T. Hoefler, "A modular benchmarking infrastructure for high-performance and reproducible deep learning," *arXiv preprint arXiv:1901.10183*, 2019.

Non-Volatile Memory Array Based Quantization- and Noise-Resilient LSTM Neural Networks

Wen Ma
Western Digital Research
Milpitas, CA USA
wen.ma@wdc.com

Pi-Feng Chiu
Western Digital Research
Milpitas, CA USA
pi-feng.chiu@wdc.com

Won Ho Choi
Western Digital Research
Milpitas, CA USA
won.ho.choi@wdc.com

Minghai Qin
Western Digital Research
Milpitas, CA USA
minghai.qin@wdc.com

Daniel Bedau
Western Digital Research
San Jose, CA USA
daniel.bedau@wdc.com

Martin Lueker-Boden
Western Digital Research
Milpitas, CA USA
martin.lueker-boden@wdc.com

***Abstract*—In cloud and edge computing models, it is important that compute devices at the edge be as power efficient as possible. Long short-term memory (LSTM) neural networks have been widely used for natural language processing, time series prediction and many other sequential data tasks. Thus, for these applications there is increasing need for low-power accelerators for LSTM model inference at the edge. In order to reduce power dissipation due to data transfers within inference devices, there has been significant interest in accelerating vector-matrix multiplication (VMM) operations using non-volatile memory (NVM) weight arrays. In NVM array-based hardware, reduced bit-widths also significantly increases the power efficiency. In this paper, we focus on the application of quantization-aware training algorithm to LSTM models, and the benefits these models bring in terms of resilience against both quantization error and analog device noise. We have shown that only 4-bit NVM weights and 4-bit ADC/DACs are needed to produce equivalent LSTM network performance as floating-point baseline. Reasonable levels of ADC quantization noise and weight noise can be naturally tolerated within our NVM-based quantized LSTM network. Benchmark analysis of our proposed LSTM accelerator for inference has shown at least 2.4× better computing efficiency and 40× higher area efficiency than traditional digital approaches (GPU, FPGA, and ASIC). Some other novel approaches based on NVM promise to deliver higher computing efficiency (up to ×4.7) but require larger arrays with potential higher error rates.**

Keywords—quantization, noise, LSTM, non-volatile memory, machine learning hardware

I. INTRODUCTION

Deep neural networks [1] have gained great popularity during the past several years and have become one of the most widely used machine learning technique nowadays. Deep neural networks can be broadly classified into two categories: feedforward neural networks and recurrent neural networks, depending on whether there are loops present inside the network topology. Unlike feedforward neural networks such as convolutional neural network (CNN) and multilayer perceptron (MLP) that are being used for static input problems like image recognition, object detection etc., recurrent neural networks such as long short-term memory (LSTM), gated recurrent unit (GRU), and echo state network (ESN) are suitable for non-static input tasks including natural language processing, time-series prediction etc. LSTM [2] is a special kind of recurrent neural networks that was first designed to avoid the exploding or vanishing gradient problems during backpropagation, and have now become the state-of-the-art approach for speech recognition. LSTM, combined with other types of neural networks like CNN, is used by Siri, Google voice, Alexa, etc. but is usually executed on cloud servers and data need to be transmitted between clients and servers through wire/wireless networks which may encounter instability or interruptions. It is desirable to have low-power embedded hardware to run deep neural networks directly on power restricted systems, such as mobile devices and self-driving cars.

Neuromorphic chips are regarded as promising technology to be integrated with mobile devices considering their great advantage in power efficiency and computing speed. They are usually based on CMOS VLSI circuits and attempt to mimic the human brain to perform computations by taking advantage of the massive parallelism when billions of neurons and trillions of synapses process and store information [3]. Some of the existing notable efforts on neuromorphic computing hardware systems include IBM's TrueNorth [4], Stanford's Neurogrid [5], EU's BrainScaleS [6], and more recently Intel's Loihi [7], etc. In addition to using CMOS based analog/digital circuits, non-volatile memory (NVM) devices can be integrated to accelerate neuromorphic computing or machine learning hardware, as they can be used directly as synaptic weights in artificial neural networks [8]. Some of the popular candidate NVM technologies for neuromorphic computing include ReRAM [8], PCRAM [9], MRAM [10] and floating gate transistors [11], which offer smaller footprint and lower power than SRAM or eDRAM that are the mainstream CMOS technologies to hold synaptic weights.

A lot of work has been done previously to investigate NVM based MLP [12]–[14] or CNN [15], [16] hardware accelerators, while only more recently, LSTM acceleration has been reported by using pure analog NVM arrays (ReRAM or PCRAM) [17]–[19]. In [17] the design was realized by using analog NVM, 7-

978-1-7281-5221-9/19/$31.00 ©2019 IEEE

bit input, and 9-bit ADC, while in [19] 8- or 16-bit number was used to improve the computing efficiency of their LSTM model. In this work, we explore the possibility of using ever lower bit-precision NVM weight array and periphery circuit component, while maintaining comparable performance with the software baseline (32-bit). This is the first work to our knowledge that explore the quantization effects on both NVM weights and ADC/DAC on the periphery for LSTM network, and we show that for different benchmark tasks, 4-bit weight along with 4-bit ADC/DAC is in general able to produce satisfactory result.

The highlights of this paper can be summarized as:

1) Quantization-aware training (potentially performed on CPUs or GPUs) was used for our LSTM network to achieve extreme low bit-precision (< 4 bits) requirements on the NVM array-based hardware for inference purposes, making our network quantization-resilient.

2) Detailed noise analysis on the circuit and device level was performed and it was shown that reasonable amount of ADC quantization noise and weight noise can be naturally tolerated within the quantized LSTM hardware, making our network noise-resilient.

3) Detailed benchmark analysis has shown that our NVM-based LSTM neural network quantized to 4-bit precision can offer 13× higher computing efficiency (throughput / power) and over 8000× higher area efficiency (throughput / area) than state-of-the-art GPU that focuses on edge applications, over 400× higher computing efficiency than FPGA-based LSTM accelerators, 2.4× better computing efficiency and 40× better area efficiency than traditional SRAM-based ASIC approach. Some other NVM-based approaches promise to deliver higher computing efficiency (up to ×4.7) and area efficiency (up to ×3.1) but require larger arrays with potential neural network performance degradation (e.g. higher error rates).

The rest of this paper is organized as follows. Section II discusses the concept of using NVM array to accelerate Vector-Matrix Multiplication (VMM) in machine learning directly using Ohm's law. Section III introduces the basic LSTM operations, how to use NVM arrays to accelerate LSTM operations, and where the quantization should be considered in the hardware implementation. In section IV, quantization bit-widths requirements on the hardware is explored using two major benchmark tasks without considering the analog hardware noise. Section V analyzes the effect of hardware noise on the performance of our quantized LSTM network, and in section VI, detailed benchmark analysis of our NVM-based quantized LSTM network is performed where computing efficiency, area efficiency etc. are compared with mainstream digital approaches (GPU, FPGA, ASIC) and other NVM approaches. Section VII concludes the paper.

II. Vector-Matrix Multiplication Accelerated by NVM Array

To accelerate machine learning algorithms, the NVM device cells can be constructed into a cross-point like array, as shown in Fig. 1. The cross-point structure is inspired by biology where each pre-synaptic neuron corresponds to each row and each post-synaptic neuron corresponds to each column, therefore each cross junction is one synapse, which is represented by one NVM cell [8]. When used in the read mode, i.e. the conductance values of the NVM weight cells are stationary, the NVM array can accelerate vector-matrix multiplication (VMM) operations directly in physics using Ohm's law [20]. The readout current from each column is the dot product of input voltage values from the rows (when encoded in the amplitude of the voltage pulse) and the stationary conductance values from the NVM cells on that column. Altogether, the readout currents from all the columns represent the VMM of the input voltage vector and the NVM weight array matrix. As VMM is heavily used in most machine learning algorithms, its acceleration efficiency is the most important figure-of-merit in the system.

Such analog VMM realized by using the analog weight array may possibly run into several challenges such as the available NVM cell conductance level is limited to a certain number of bits [21]. Even though ReRAM and PCRAM can have almost continuous incremental conductance change [8], [9], 32-bit precision of weight is hard to achieve, while MRAM and NOR Flash are mostly binary type memory cells [22]. In addition to the inherent limitations posed by the NVM devices, implementing high precision periphery circuits can be very costly in terms of area and power, especially for ADCs due to the number of comparators required for high resolution conversion. Previous studies have shown that ADCs as a classifier between the analog weight array and digital back-end circuits consume most of the power in the system [16]. Therefore, it is important to explore the possibility of using low bit-precision weight memory array and periphery circuit component, while maintaining comparable performance with the software baseline (32-bit).

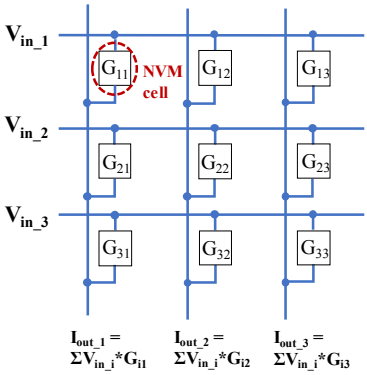


Fig. 1. Schematic of using NVM crosspoint array to accelerate VMM using Ohm's law.

III. Hardware Accelerated Quantized LSTM

There have been many research efforts from the algorithm point of view on binarizing or quantizing the feedforward neural networks like CNN and MLP [23], [24]. Binarizing LSTM is more challenging than binarizing the CNN or MLP as it is difficult to adopt the back-end techniques like batch normalization in a recurrent neural network [25]. Instead, quantized LSTM has been studied and it is revealed that low quantization bit-widths can be achieved by quantizing the weights and hidden state during forward propagation and using straight-through estimator (STE) to propagate the gradient for weight update [26], [27]. However, these quantized LSTM

studies are not based on considerations of the real hardware implementation. For example, hardware implementations based on the NVM weight array need quantization on more than just the weights and hidden state, as will be discussed in the following sections.

A. Basic LSTM Operations

The forward propagation operation of the LSTM unit (Fig. 2) contains 4 vector-matrix multiplications, 5 nonlinear activations, 3 element-wise multiplications and 1 element-wise addition [28]. As shown in Equation (1) - (4), the hidden state of the previous time step h_{t-1} is concatenated with the input of the current step x_t to form the total input vector being fed into the weight arrays W_f, W_i, W_o and W_c to perform the VMM. The VMM results will be passed into 4 nonlinear activation function units respectively to get the values of forget gate f_t, input gate i_t, output gate o_t and new candidate memory cell c_c_t. The new memory cell c_t is composed of the new information desired to be added by multiplying the new candidate memory c_c_t with input gate i_t, and the old information desired to be not forgotten by multiplying the old memory cell c_{t-1} and forget gate f_t, shown in Equation (5). The final hidden state h_t is calculated by multiplying the output gate o_t and the activation of the new memory cell c_t, shown in Equation (6). During backpropagation, the values of W_f, W_i, W_o and W_c are updated according to the training algorithm, usually based on the stochastic gradient descent.

$$f_t = \text{sigmoid}\left([x_t, h_{t-1}]\, W_f\right) \quad (1)$$

$$i_t = \text{sigmoid}\left([x_t, h_{t-1}]\, W_i\right) \quad (2)$$

$$o_t = \text{sigmoid}\left([x_t, h_{t-1}]\, W_o\right) \quad (3)$$

$$c\_c_t = \tanh\left([x_t, h_{t-1}]\, W_c\right) \quad (4)$$

$$c_t = f_t \bullet c_{t-1} + i_t \bullet c\_c_t \quad (5)$$

$$h_t = o_t \bullet \tanh\left(c_t\right) \quad (6)$$

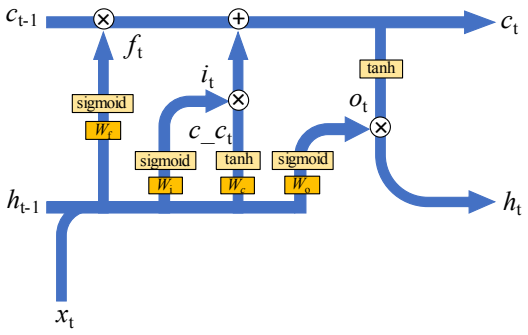


Fig. 2. LSTM basic operation flowchart

B. NVM Weight Array Accelerated LSTM Unit

The 4 vector-matrix multiplications to calculate the forget gate, input gate, output gate and new candidate memory cell can be accelerated by NVM weight arrays, as shown in Fig. 3. The 4 weight arrays representing W_f, W_i, W_o and W_c can be concatenated into a whole NVM array to calculate the VMM results in parallel. As the input x_t and previous hidden state h_{t-1} processed after the DACs are in the form of analog voltages, NVM weight arrays are resistive cross-point arrays, the VMM results are therefore in the form of analog currents that will pass the ADCs to be converted into digital values. Note that due to the relatively large area of ADCs, it is reasonable to perform time-multiplexing on the ADCs between different columns. Column multiplexers are used to choose which columns to be connected to the ADCs for each time step. After the ADCs, the digital values representing the VMM results will then be fed into different activation function units (either sigmoid or tanh) to get the final values of the forget gate f_t, input gate i_t, output gate o_t and new candidate memory cell c_c_t that can later be processed in other hardware components (elementwise multiplication and addition units) to generate the new hidden state h_t, which will then be fed into the DAC in the next cycle as part of the total input vector.

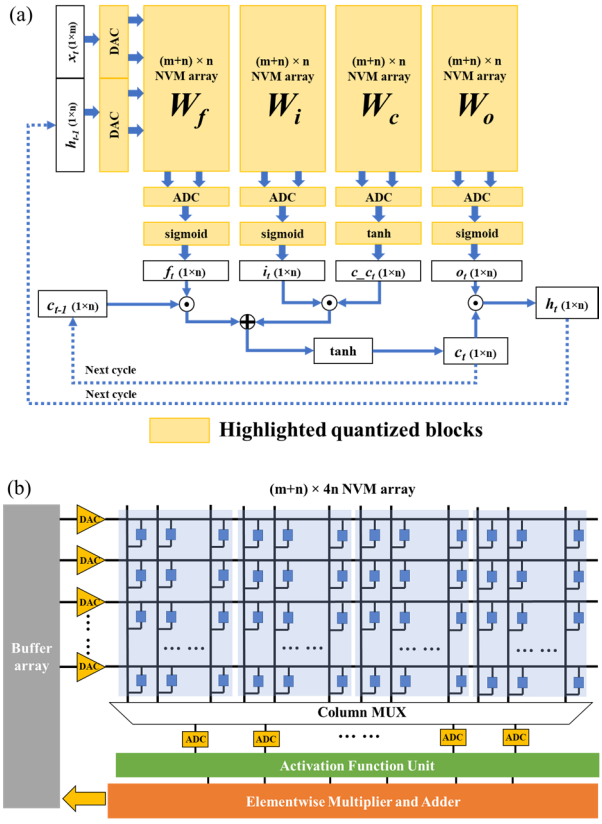


Fig. 3. Architecture of NVM weight array accelerated LSTM unit. (a) Operation flow indicated by block diagram. (b) Hardware design.

As mentioned before, due to the limitations on the available number of stable resistance states on a single NVM cell, it is worthwhile to try to lower the bit-precision of the weights. In addition, the lower bit-precision of the ADC or DAC will lead to lower cost and smaller area/power consumption. Therefore, we target to quantize the highlighted blocks in Fig. 3(a) to a lower bit-width than 32-bit floating-point baseline while we achieve a comparable performance in terms of accuracy. The memory cell values are stored at 32-bit without being quantized to lower bit-width. Note that the activation units (sigmoid or

tanh) are quantized naturally with the same bit-widths as the ADCs. Lower bit-width of the non-linear activation units can lead to area/power savings as well. For example, a 4-bit ADC would only require 16 entry look-up tables (LUTs) for sigmoid/tanh activation.

Our training approach is taking quantization into consideration, as the parameters targeted to be quantized, such as weights, inputs from DACs, outputs through ADCs, are already quantized during forward propagation for both training and inference. Straight-through estimator (STE) method is used to propagate the gradients for weight update during back propagation [26], [27]. This quantization-aware training approach is supposed to enhance the performance when the quantization bit-width is extremely low such as 1 or 2 bits. On the contrary, if the quantization is only done offline during inference, such that training is still performed with full precision numbers, it is very difficult to maintain satisfying inference performance with extremely low quantization bit-widths.

IV. Bit Precision Requirement on LSTM Weight Array and Circuit Components

To evaluate the performance of our quantized LSTM neural network based on the NVM array, two major tasks are performed: Human Activity Recognition (HAR) and Natural Language Processing (NLP). Different bit-precisions of the weights and ADC/DACs are explored to compare with high precision floating-point baselines. Note that the analog hardware noise is not considered in this section.

A. Human Activity Recognition (HAR)

The Human Activity Recognition database [29] was built from the recordings of 30 subjects wearing smartphones while performing six daily activities: walking, walking upstairs, walking downstairs, sitting, standing and laying. LSTM is used for this dataset to learn and recognize the type of activity the user is doing. The input vector size and hidden state size for the LSTM unit are both fixed at 32. Therefore, the NVM weight array size used is 64 × 128 (m = n = 32 in Fig. 3). Different bit precisions ranging from 1 to 16 bits are used for NVM weights and ADC/DACs and quantization-aware training algorithm using STE is applied.

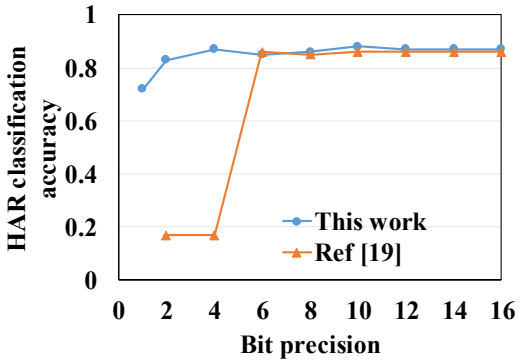


Fig. 4. Human Activity Recognition result. By using the quantization-aware training with straight-through estimator (STE) method, classification accuracy is improved significantly compared to [19] at lower quantization bit-widths such as 1-4 bits.

To compare this result with a similar work [19] that uses retraining-based iterative parameter quantization, where training is still carried out with full precision numbers and then followed by quantization and retraining for each iteration, we observe a significant improvement on the classification accuracy at lower quantization bit widths such as 1 – 4 bits (Fig. 4). It is important to point out that with 4-bit NVM weight precision and 4-bit ADC/DAC, the classification accuracy does not degrade compared to the 16-bit baseline.

B. Natural Language Processing (NLP)

Two subtasks are chosen for NLP. The first subtask - the Penn Treebank dataset [30] contains 10K unique words from Wall Street Journal material annotated in Treebank style. With the Penn Treebank corpus, the task is to predict the next word in the sentence based on previous words, and the performance is measured in perplexity per word (PPW). The perplexity is roughly the inverse of the probability of correct prediction. The input vector size and hidden state size are both fixed at 300. Therefore, the NVM weight array size is 600 × 1200 (m = n = 300 in Fig. 3). As shown in Fig. 5, as training progresses, the validation perplexity continues decreasing for the 32-bit FP baseline, 2-bit weight 2-bit ADC/DAC, and 4-bit weight 4-bit ADC/DAC cases. The 1-bit weight 2-bit ADC/DAC example case does not show a successful training as the validation perplexity does not converge, while the 4-bit weight 4-bit ADC/DAC case produces competitive training result with the FP baseline without noticeable degradation.

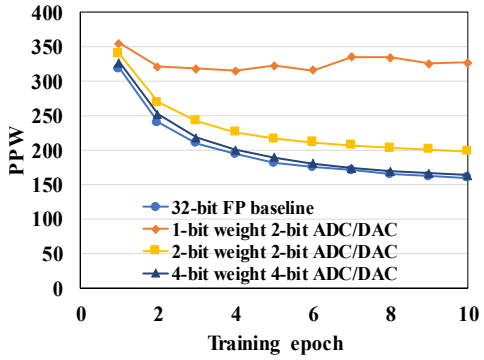


Fig. 5. Penn Treebank dataset result. Validation perplexity does not converge for the 1-bit weight 2-bit ADC/DAC case while the other bitwidth configurations produce successful training. 4-bit weight 4-bit ADC/DAC can generate close-to-equivalent training result with the FP.

To fully explore the bit-width requirement on the weights and ADC/DAC, all combinations of bit-precision ranging from 1 to 4 bits are tested as shown in Fig. 6, 4-bit weight along with at least 2 bits of ADC/DAC are required to achieve a comparable result with the floating-point baseline (less than 5% of perplexity increase). It clearly turns out that the high bit-precision of the weight plays a more important role than the high bit-precision of ADC/DAC for the general performance of the LSTM network, such as with 1-bit weight and 2-bit ADC/DAC the final PPW is 327, which is higher than 282, the PPW achieved with 2-bit weight and 1-bit ADC/DAC. Similar phenomenon was observed by comparing performances between the 2-bit weight 4-bit ADC/DAC case (PPW=182) and the 4-bit weight 2-bit ADC/DAC case (PPW=172). Therefore, we can conclude that

improving the resolution of the conductance levels of NVM cell weights is more important than increasing the precision of ADC/DAC.

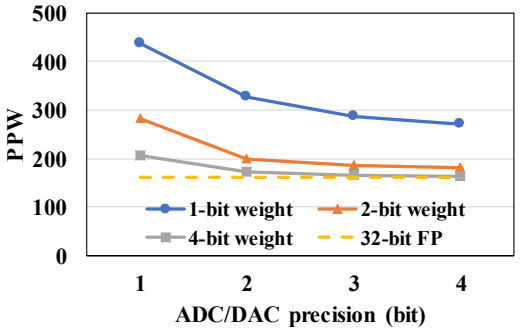


Fig. 6. Penn Treebank dataset result with full exploration on the bit-widths of weight and ADC/DAC. The PPW is measured as the validation perplexity after 10 epochs of training.

The second subtask is the national name prediction where the next character in a name is predicted instead of the next word. The perplexity metric here is for per character. The input vector size is 100 while the hidden state size is 256. So, the NVM weight array size used for this task is 356 × 1024 (m = 100, n = 256 in Fig. 3). After 8000 training iterations, the training perplexity and accuracy are measured. As shown in Table I, using 2-bit weight and 2-bit ADC/DAC is sufficient to produce result within 5% degradation compared to the floating point baseline case. Comparing with the result from the Penn Treebank, a lower bit-precision requirement on the weight and ADC/DAC is needed for the simpler character prediction task. To conclude from both NLP tasks, a 4-bit weight along with 4-bit ADC/DAC can ensure almost-zero degradation for LSTM network performance. Such bit-width requirements for training also ensure the inference performance as our training approach is quantization-aware. Quantization-aware model-based training ensures that the forward pass matches precision for both training and inference.

TABLE I. TRAINING PERPLEXITY AND ACCURACY AT DIFFERENT BIT-WIDTHS CONFIGURATIONS FOR CHARACTER PREDICTION

Character prediction result		
LSTM configuration	***Training accuracy (%)***	***Training perplexity (per character)***
32-bit floating point baseline	85.09	1.52
4-bit weight + 4-bit ADC/DAC	85	1.55
2-bit weight + 2-bit ADC/DAC	83.6	1.58
1-bit weight + 1-bit ADC/DAC	72.82	2.27

V. EFFECT OF DEVICE AND CIRCUIT NOISE

After we show the possibility of using lower bit-precision weights and peripheries for LSTM, non-idealities in the quantization precisions of NVM weight cells and ADC/DACs should be considered for their potential impact on network performance. Especially during forward propagation, the read noise can distort the VMM result and thus lead to inaccurate weight updates. The two main sources of noise are the analog digital conversion and the device itself: the resistance of the memory element is intrinsically fluctuating. Specifically, the types of noises being considered include:

1) Johnson-Nyquist (thermal) noise, which is caused by the random thermal motion of charge carriers. The effect is intrinsic to any resistor and puts a lower limit on how fast any memory element can be read.

2) Shot noise, which is the statistical fluctuation of the electric current due to its quantized nature, and manifests predominantly when the current traverses a barrier, like in many resistive memory cells.

3) Random telegraph noise (RTN), or burst noise, which is commonly caused by the random motion of charge carriers between trapping sites. The output signal typically appears to switch back and forth between different levels.

4) 1/f (flicker) noise, which has a 1/f power spectral density and can be the result of impurities, generation and recombination, and randomly distributed traps. Superposition of many RTN processes on different scales will have a 1/f spectrum.

5) Quantization noise, which is the error introduced by quantization in the analog-to-digital converter (ADC). It is the rounding error between the analog input voltage of the ADC and the output digitized value and is discussed as follows.

A. *Effect of ADC Noise*

The ADC noise consists of thermal as well as quantization noise. For relevant bandwidths, the quantization noise, $V_{\mathrm{QNoise}} = \Delta/\sqrt{12}$ [31], with Δ being the quantization step size of ADC, is larger than thermal and shot noise. For example, for a signal range of ±1 V and 2-bit resolution we can expect a quantization noise of $V_{\mathrm{QNoise}} = 1 / (2^1 \cdot \sqrt{12}) = 0.144\mathrm{V}$, which is dominant compared to the Johnson-Nyquist noise $V_{\mathrm{JNoise}} = \sqrt{k_B T \cdot R \cdot \mathrm{BW}}$ [32], which is around 4 nV / $\sqrt{\mathrm{Hz}}$ for a 1 k resistor. Therefore, the resistive thermal noise of the ADC can be neglected.

Another way to look at the noise is to calculate the effective number of bits from the noise power in relation to the signal power. Effective number of bits, or ENOB = (SNR – 1.76dB) / 6.02dB [31], with signal to noise ratio (SNR) defined as $P_{\mathrm{signal}}/P_{\mathrm{noise}}$. This is essentially the same as the equation for the root mean square value of $V_{\mathrm{QNoise}} = \Delta/\sqrt{12}$.

To simply model the ADC quantization noise, an additive noise term is added to the values at the forget gate, input gate, output gate and new candidate memory cell during ADC quantization and before the activation function units. The noise follows a Gaussian distribution with a standard deviation $\sigma = V_{\mathrm{QNoise}} = (V_{\max} - V_{\min}) / (2^N \cdot \sqrt{12})$, N is the ADC bit resolution. For example, at the forget gate:

$$f_t = sigmoid\,([x_t, h_{t-1}]\, W_f + Z_{\mathrm{ADC}}) \tag{7}$$

$$Z_{\mathrm{ADC}} \sim N(0, \sigma^2),\ \sigma = V_{\mathrm{QNoise}} \tag{8}$$

Where Z_{ADC} is the ADC quantization noise vector with the same dimension as $[x_t, h_{t-1}]$ W_f. It follows a Gaussian distribution with zero mean and a standard deviation σ that equals the ADC quantization noise root mean square value, which is V_{QNoise} by definition. As shown in Fig. 7, the influence of ADC quantization noise on the LSTM performance is negligible. The experiment is run on the Penn Treebank corpus measuring the validation perplexity after 10 epochs of training. Note that the noise is also present during the quantization-aware training stage as our LSTM network can be considered for online training as well.

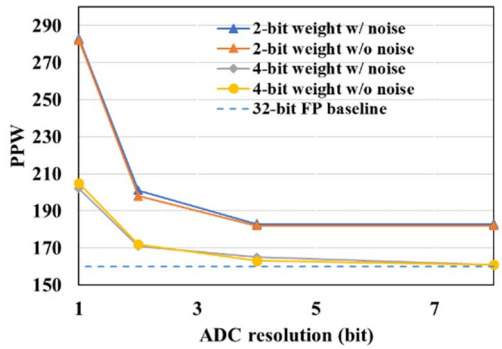


Fig. 7. Effect of ADC quantization noise on the Penn Treebank experiment. Weight resolution are fixed at either 2 bits or 4 bits. Validation perplexity is measued.

B. *Effect of Weight Noise*

As explained in the previous section, the resistive thermal (Johnson-Nyquist) noise will also be negligible for the memory cells, as well as the shot noise, which is given by $I_{SNoise} = \sqrt{2e \cdot I \cdot BW}$ [32]. Due to the atomistic nature of transport in resistive memory cells, $1/f$ noise and random telegraph noise (RTN) will be dominant [33]. From literature, we know that the level of RTN noise and $1/f$ noise is dependent on the resistance levels, such as σ_R/R is around 0.1 at high resistance (low conductance) states and around 0.01 at low resistance (high conductance) states.

To model the weight noise, an additive noise term is added to the values of the weight arrays. The noise follows a Gaussian distribution with a standard deviation proportional to the total weight range. For example, at the forget gate:

$$f_t = sigmoid\,([x_t, h_{t-1}]\,(W_f + Z_w)) \tag{9}$$

$$Z_w \sim N\,(0, \sigma^2),\ \sigma = \beta\,(w_{max} - w_{min}) \tag{10}$$

Where Z_w is the weight noise matrix with the same dimension as W_f. It follows a Gaussian distribution with zero mean and a standard deviation σ ranging from 0 to 20% of the total weight range w_{max} - w_{min}. We define the percentage of the weight range β as the weight noise ratio. Only exploring β from 0 to 20% is realistic with previously mentioned NVM device performance. Fig. 8 shows that even 20% weight noise have no harmful effect on the LSTM network performance with the same Penn Treebank experiment setup. From analyzing both the ADC quantization noise and weight noise, we can clearly say that our LSTM network is robust against reasonable levels of hardware noise.

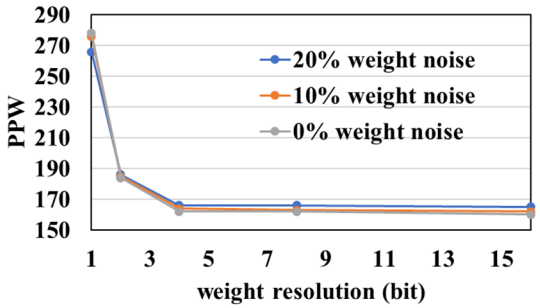


Fig. 8. Effect of weight noise on the Penn Treebank experiment. ADC/DAC resolution is fixed at 4 bits. Validation perplexity is measured after 10 epochs of training.

VI. Benchmark Analysis

Here we compare the inference performance of our NVM-based quantized LSTM network at 4-bit weight and 4-bit ADC/DAC configuration with other existing hardware platforms including GPU (Nvidia Jetson AGX Xavier [34]) that focuses on edge applications, FPGA-based LSTM accelerator [35] that uses DRAM for weight storage, ASIC-based neuromorphic engine [36] that uses SRAM for weight storage and other NVM-based LSTM accelerators [17], [19]. For GPU- and FPGA-based systems, the high throughput usually comes with high power and area consumption, so the overall computing and area efficiency are generally lower than ASIC- and NVM-based systems (Table II, Table III, and Fig. 9).

A. *Throughput Analysis*

The latency for NVM based vector-matrix multiplication is around 80 - 100 ns [17], [19]. Therefore, for an NVM array size of 356 × 1024 used in our paper for the character prediction task, the estimated throughput for vector-matrix multiplication is 3645 GOP/s (assuming 100 ns latency with 64 ADCs at sampling rate of 160 MS/s), which is comparable with GPU. In addition, we need to consider the latency for non-linear activation function, which is assumed to be 5 ns, and element-wise operations, which is assumed to be 1 ns. Communication overhead is not considered here. So, the overall throughput is calculated to be 3439 GOP/s. IBM estimates that their resistive processing unit (RPU) devices [17] accelerate the throughput to be 84 TOP/s and assume 3 resistive cross-point arrays with the size of 4096 × 4096 are used at the same time. In general, larger array size enables higher throughput, but the line resistance and parasitic capacitance of larger arrays can become non-negligible and will potentially harm the network accuracy. Note that in the RRAM-based Processing-In-Memory (PIM) architecture paper [19], each subarray size is only 128 × 128, thus the achieved throughput is only 108.4 GOP/s.

B. *Power Estimation*

ADCs can consume a relatively large amount of power in the overall architecture. For a 356 × 1024 NVM array it is unrealistic to use 1024 ADCs with full parallel readout for all the columns. Therefore, we locate 64 ADCs to perform time-multiplexing so

that every 16 columns are sharing one ADC (for the 356 × 1024 array). Suppose we need 100 ns latency for each column, so the sampling rate of ADC needs to be at least 160 MS/s. Energy consumption per sample in 4-bit ADC is assumed as 1 pJ [37]. For ADCs at lower resolutions, the energy performance seems to be independent of ENOB. For higher resolutions such as ENOB $\geq$ 9, the energy per sample quadruples for every additional bit of effective resolution in ADC. The state-of-the-art energy per sample vs. ENOB almost exactly follows the relationship $E = 2^{2(ENOB-9)}$ pJ for ENOB $\geq$ 9. Therefore, in our design with 4-bit ADC with 160 MS/s sampling rate, each ADC consumes 0.16 mW, and with 64 4-bit ADCs the estimated power is 0.01 W. On the other hand, if 12-bit ADCs are used instead, the energy consumption per sample will skyrocket to 64 pJ and the overall power consumption from 12-bit ADCs will be 0.64 W.

TABLE II. COMPARISON WITH MAINSTREAM DIGITAL APPROACHES

	GPU Nvidia Jetson [34]	ESE [35]	Tianjic [36]	This work
Technology	GPU	FPGA	ASIC	NVM
Precision (bit)	16	12	8	4
Throughput (GOP/s)	3,478	282	1,214	3,439
Power (W)	15	41	0.95	1.136
Computing efficiency (GOP/s/W)	231	6.88	1,278	3,027
Area (mm^2)	8700	N/A	14.44	1.031
Area efficiency (GOP/s/mm^2)	0.399	N/A	84	3,333

TABLE III. COMPARISON WITH OTHER NVM-BASED APPROACHES

	IBM's RPU [17]	RRAM PIM [19]	This work
Array size	4096 × 4096	128 × 128	356 × 1024
Precision (bit)	N/A	16	4
Throughput (GOP/s)	84,000	108.4	3,439
Power (W)	6	0.932	1.136
Computing efficiency (GOP/s/W)	14,166	116.3	3,027
Area (mm^2)	8.04	0.39	1.031
Area efficiency (GOP/s/mm^2)	10,477	277	3,333

For estimating read power consumption in NVM array while performing inference operation, the assumed read voltage for our NVM cell is 1 V, and the assumed average resistance of the NVM cells is 1 MΩ. Therefore, the power from the 356 × 1024 NVM array is 0.364 W.

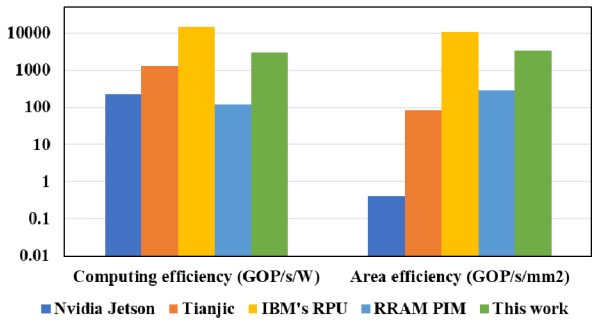


Fig. 9. Comparison of our NVM-based quantized LSTM neural network with other hardware platforms.

In addition to the power consumption by the ADCs and NVM array, we need to consider how much power the other peripherals including the DACs, multiplexers, data buffers, non-linear activation function units, and element-wise operations consume. We utilized relevant numbers from [19] and came to a total number of 1.136 W for power consumption of our design.

The computing efficiency of our NVM-based quantized LSTM (throughput divided by power) – around 3 TOP/s/W is higher than traditional digital approaches (Table II). FPGA platforms show relatively lower computing efficiencies because of their relatively higher power consumption (41 W) and lower throughput (282 GOP/s), while Nvidia Jetson GPU and ASIC-based approaches are in the middle range of performance. Compared with other NVM-based approaches, our computing efficiency is in the middle (Table III). IBM's RPU shows exceedingly high computing efficiency due to the enormous size of the array being used. Note that if the neural network precision in [19] is decreased to 4-bit, their computing efficiency is estimated to 2.7 TOP/s/W, which is very close to our result. Therefore, we see the obvious advantage of utilizing low bit-width to enhance computing efficiency.

C. *Area Estimation*

Similar to the power estimation, the large footprint of ADC in nature can be dominant in overall system area. The area of ADC increases by 2-2.2× for every additional bit of ENOB when ENOB $\geq$ 6 [38]. A 4-bit ADC consumes around 0.01 mm^2 [38] so that the estimated area of 64 4-bit ADCs on a 356 × 1024 array is 0.64 mm^2. In comparison, if 12-bit ADCs are used instead, the area consumption per ADC will increase to 1 mm^2. Therefore, the overall area consumption from 12-bit ADCs will be 64 mm^2, which is intolerable.

We assume that the NVM array itself has 400 nm pitch with 200 nm NVM device width and 200 nm spacing between devices. A 356 × 1024 array will therefore consume 0.058 mm^2, which is insignificant compared to the ADC area consumption. In addition to ADCs and NVM array, area consumption by the DACs, multiplexers, data buffers, non-linear activation function units, and element-wise operations cannot be neglected. Similarly, we utilized relevant numbers from [19] and came to a total of 1.031 mm^2 for area consumption of our design.

The area efficiency of our NVM-based quantized LSTM (throughput divided by area) – around 3 TOP/s/mm^2 is also among the highest in the benchmark analysis as shown in Table

II, Table III, and Fig. 9. By lowering the resolution of ADCs, significant amount of power and area can both be saved in our design. As for the NVM weights, the bit-width is not critical as much as ADC's for our current design in terms of power and area performance as the NVM memory cells are inherently multi-level cells.

VII. Conclusion

Quantizing the LSTM neural network becomes very important when it comes to the embedded hardware design and acceleration of state-of-the-art machine learning to lower the memory size and computation complexity. Specifically, NVM cross-point arrays can accelerate the VMM operations that are heavily used in most machine learning algorithms for artificial neural networks, including but not limited to LSTM, CNN and MLP. Previous literature has reported on using pure analog NVM arrays for LSTM without considering any quantization bit-width requirements on the weight memory cells or circuit components [17], [18]. To our best knowledge, this is the first work that explore the quantization effects on both NVM weights and ADC/DAC on the periphery for LSTM.

By utilizing a quantization-aware training approach, our NVM-based LSTM network is both quantization- and noise-resilient. We have found that 4-bit NVM weight cell along with 4-bit ADC/DAC in the LSTM unit can deliver comparable performance as the floating-point baseline in human activity recognition and natural language processing tasks. For a simple dataset for character level prediction, even 2-bit NVM weight cell along with 2-bit ADC/DAC does not show noticeable degradation in performance. In addition, ADC quantization noise and NVM weight noise can both be well tolerated for our quantized LSTM network as well, making it robust against realistic hardware noise.

Detailed benchmark analysis has shown that our NVM-based quantized LSTM neural network at 4-bit precision can offer at least 13× higher computing efficiency and 8000× times higher area efficiency than Nvidia Jetson GPU, over 400× times higher computing efficiency than the FPGA-based LSTM accelerator – ESE, 2.4× better computing efficiency and 40× better area efficiency than the neuromorphic chip – Tianjic. Some other novel approaches based on NVM, such as IBM's RPU, promise to deliver higher computing efficiency (up to ×4.7) but require larger arrays (4096 × 4096) with potential higher error rates caused by the non-negligible parasitic capacitance and line resistance. Even with competitive computing efficiency, traditional ASIC-based method still has relatively low area efficiency due to the larger area of SRAM devices, while NVM-based approaches in general already start to show area efficiency advantage.

Acknowledgment

The authors would like to thank Adel Dokhanchi, Tung Hoang and Haoyu Wu for useful discussions.

References

[1] Y. A. LeCun, Y. Bengio, and G. E. Hinton, "Deep learning," *Nature*, 2015.

[2] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[3] C. Mead, "Neuromorphic Electronic Systems," *Proc. IEEE*, vol. 78, no. 10, pp. 1629–1636, 1990.

[4] P. A. Merolla *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science (80-.).*, vol. 345, no. 6197, pp. 668–673, 2014.

[5] B. V. Benjamin *et al.*, "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proc. IEEE*, vol. 102, no. 5, pp. 699–716, May 2014.

[6] J. Schemmel, D. Brüderle, A. Grübl, M. Hock, K. Meier, and S. Millner, "A wafer-scale neuromorphic hardware system for large-scale neural modeling," in *ISCAS 2010 - 2010 IEEE International Symposium on Circuits and Systems: Nano-Bio Circuit Fabrics and Systems*, 2010, pp. 1947–1950.

[7] M. Davies *et al.*, "Loihi: A Neuromorphic Manycore Processor with On-Chip Learning," *IEEE Micro*, 2018.

[8] S. H. Jo, T. Chang, I. Ebong, B. B. Bhadviya, P. Mazumder, and W. Lu, "Nanoscale memristor device as synapse in neuromorphic systems," *Nano Lett.*, vol. 10, no. 4, pp. 1297–1301, Apr. 2010.

[9] D. Kuzum, R. G. D. Jeyasingh, B. Lee, and H. S. P. Wong, "Nanoelectronic programmable synapses based on phase change materials for brain-inspired computing," *Nano Lett.*, vol. 12, no. 5, pp. 2179–2186, May 2012.

[10] S. Lequeux *et al.*, "A magnetic synapse: Multilevel spin-torque memristor with perpendicular anisotropy," *Sci. Rep.*, 2016.

[11] X. Guo *et al.*, "Fast, energy-efficient, robust, and reproducible mixed-signal neuromorphic classifier based on embedded NOR flash memory technology," in *Technical Digest - International Electron Devices Meeting, IEDM*, 2018.

[12] P. Y. Chen, X. Peng, and S. Yu, "NeuroSim: A circuit-level macro model for benchmarking neuro-inspired architectures in online learning," *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, vol. 37, no. 12, pp. 3067–3080, 2018.

[13] P. Y. Chen, X. Peng, and S. Yu, "NeuroSim+: An integrated device-to-algorithm framework for benchmarking synaptic devices and array architectures," in *Technical Digest - International Electron Devices Meeting, IEDM*, 2018, p. 6.1.1-6.1.4.

[14] S. Agarwal *et al.*, "Achieving ideal accuracies in analog neuromorphic computing using periodic carry," in *Digest of Technical Papers - Symposium on VLSI Technology*, 2017, pp. T174–T175.

[15] P. Chi *et al.*, "PRIME: A Novel Processing-in-Memory Architecture for Neural Network Computation in ReRAM-Based Main Memory," in *Proceedings - 2016 43rd International Symposium on Computer Architecture, ISCA 2016*, 2016.

[16] A. Shafiee *et al.*, "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," in *Proceedings - 2016 43rd International Symposium on Computer Architecture, ISCA 2016*, 2016, pp. 14–26.

[17] T. Gokmen, M. Rasch, and W. Haensch, "Training LSTM Networks with Resistive Cross-Point Devices," *arXiv:1806.00166*, 2018.

[18] C. Li *et al.*, "Long short-term memory networks in memristor crossbars,"

arXiv:1805.11801, 2018.

[19] Y. Long, T. Na, and S. Mukhopadhyay, "ReRAM-Based Processing-in-Memory Architecture for Recurrent Neural Network Acceleration," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 26, no. 12, pp. 2781–2794, 2018.

[20] M. Hu *et al.*, "Dot-product engine for neuromorphic computing," in *Proceedings of the 53rd Annual Design Automation Conference on - DAC '16*, 2016, pp. 1–6.

[21] C. Sung, H. Hwang, and I. K. Yoo, "Perspective: A review on memristive hardware for neuromorphic computation," *J. Appl. Phys.*, vol. 124, no. 151903, 2018.

[22] A. F. Vincent *et al.*, "Spin-transfer torque magnetic memory as a stochastic memristive synapse for neuromorphic systems," *IEEE Trans. Biomed. Circuits Syst.*, 2015.

[23] I. Hubara, M. Courbariaux, D. Soudry, D. El-Yaniv, and B. Yoshua, "Binarized Neural Networks," *Adv. Neural Inf. Process. Syst.*, 2016.

[24] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-net: Imagenet classification using binary convolutional neural networks," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9908 LNCS, pp. 525–542, 2016.

[25] C. Laurent, G. Pereyra, P. Brakel, Y. Zhang, and Y. Bengio, "Batch normalized recurrent neural networks," in *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 2016.

[26] I. Hubara, M. Courbariaux, D. Soudry, E.-Y. Ran, and B. Yoshua, "Quantized Neural Networks : Training Neural Networks with Low Precision Weights and Activations," in *Journal of Machine Learning Research*, 2018, pp. 1–30.

[27] M. Z. Alom, A. T. Moody, N. Maruyama, B. C. Van Essen, and T. M. Taha, "Effective Quantization Approaches for Recurrent Neural Networks," in *Proceedings of the International Joint Conference on Neural Networks*, 2018.

[28] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," in *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, 2014.

[29] M. S. Parmar, "A Public Domain Dataset for Human Activity Recognition Using Smartphones," *Can J Cardiol*, 2013.

[30] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, "Building a large annotated corpus of English: the penn treebank," *Comput. Linguist.*, 1993.

[31] W. Kester, "Taking the Mystery out of the Infamous Formula," SNR= 6.02 N+ 1.76 dB," and Why You Should Care," *Analog Devices - MT-001 Tutor.*, 2005.

[32] K. H. Lundberg, "Noise sources in bulk CMOS," *Unpubl. Pap.*, 2002.

[33] S. Ambrogio, S. Balatti, V. McCaffrey, D. C. Wang, and D. Ielmini, "Noise-Induced Resistance Broadening in Resistive Switching Memory-Part II: Array Statistics," *IEEE Trans. Electron Devices*, vol. 62, no. 11, pp. 3812–3819, 2015.

[34] Nvidia, "Jetson AGX Xavier: Deep Learning Inference Benchmarks," 2018. [Online]. Available: https://developer.nvidia.com/embedded/jetson-agx-xavier-dl-inference-benchmarks.

[35] S. Han *et al.*, "ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA," *Proc. 2017 ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays - FPGA '17*, 2016.

[36] J. Pei *et al.*, "Towards artificial general intelligence with hybrid Tianjic chip architecture," *Nature*, 2019.

[37] B. E. Jonsson, "An empirical approach to finding energy efficient ADC architectures," in *International Workshop on ADC Modelling*, 2011, no. 1.

[38] B. E. Jonsson, "Area efficiency of ADC architectures," in *2011 20th European Conference on Circuit Theory and Design, ECCTD 2011*, 2011, pp. 560–563.

A Comparator Design Targeted Towards Neural Nets

David J. Mountain
Laboratory for Physical Sciences at Research Park
Department of Defense
Catonsville, MD USA
davidjmountain@ieee.org

***Abstract*—Theshold gates are a specific type of neural network that have been shown to be valuable for cybersecurity applications. These networks can be implemented using analog processing in memristive crossbar arrays. For these types of designs, the performance of the comparator circuit is a critical factor in the overall capabilities of the neural network. In this work a relatively simple comparator design is demonstrated to be compact, low-power, and fast. The design takes advantage of features inherent in the neural net architecture and memristor technology. This paper includes the basic design and specific enhancements to improve its capabilities, along with power, area, and timing estimates.**

***Index Terms*—artificial neural networks, nanotechnology, neural network hardware.**

I. Introduction

As machine learning and neural networks are applied to increasingly large data sets, interest in specialized hardware for these tasks increases. One technology proposed for several neural network processors is the *memristive crossbar*. These crossbars are well-suited for direct execution of neural nets [1]–[9] because they can compute a dot product in a power-efficient manner, can store their synaptic weights locally and persistently, and can be programmed in a straightforward fashion. Application of co-design techniques can be very useful for these novel computing approaches with emerging technologies.

A basic functional unit for neural nets is a combined multiply-accumulate operation, evaluated with a specific function (ReLU, sigmoid, tanh, etc.) The neuron we use is shown in Fig. 1, which is evaluated by using a threshold function. Neurons of this type implement threshold gate networks (TGN). TGN neurons have digital outputs, enabing the use of analog computing with digital communication. Our program has demonstrated a malware triage and analysis capability utilizing TGN, and mapped the TGN to a neuromorphic processor utilizing memristor arrays [8]. This paper provides a detailed look at how we use a co-design approach for our comparator, and its impact on the capabilities of the TGN.

Section II provides a high level overview of our memristive array architecture, along with the specific neuron design. Section III provides a detailed analysis of the comparator design, and Section IV discusses the beneficial impact of this design. Section V provides a summary and conclusions.

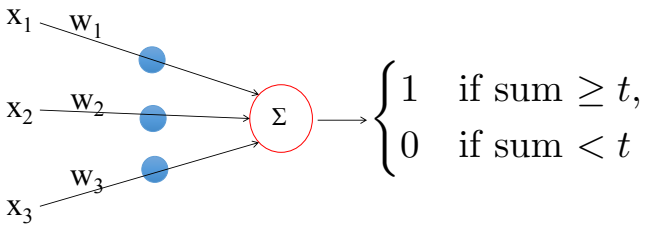


Fig. 1. Basic threshold gate network (TGN).

II. Background Information

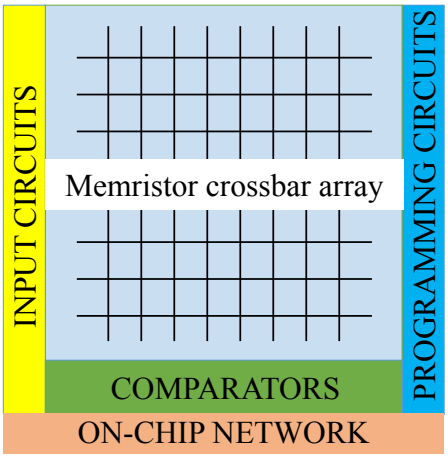
A. Architectural Definition

Fig. 2 is a block diagram of the basic architecture of our neural processor. We assume off-chip learning to enable algorithm flexibility, and incorporate 1T1M (one transistor, one memristor) array cells (see Fig. 3). The value of the 1T1M design is that the P-channel Field Effect Transistor (PFET) access device eliminates sneak path issues [10] and enables highly precise programming of memristor conductances [11]. We assume integer weight values up to 32 are possible (memristor conductance values are G = $\pm$ 1, 2, 3 ... G_{max} = 32 µS; 1 µS = 1 MΩ), based on the measured performance of Tantalum Oxide memristive devices [12]. The low conductance (or high resistance, as G = 1/R) values translate into lower power analog computing. These devices have been shown to have good endurance, and are built using Back-End-Of-Line (BEOL) fabrication processes compatible with CMOS technology. The precise programming of the conductances modestly relaxes the required sensitivity of the comparator, which must be able to distinguish a current difference equal to the smallest actual weight (conductance) difference. This will be ΔG_{min} = 1 µS minus 2X the programming precision. Comparators are used for the evaluation circuit to enable a digital communication network and digital input circuits. This building block can be replicated multiple times to create a full chip version for large neural processing applications. [9].

B. Neuron design

Fig. 4 shows a neuron using 1/High Z inputs (denoted as DZ). Fig. 5 shows the row driver circuit used to enable 1/High Z inputs. When the input is a logical low, such as in the case

978-1-7281-5221-9/19/$31.00 ©2019 IEEE



All inputs and all outputs are from/to the network

Fig. 2. Block diagram of the architecture showing the key components.

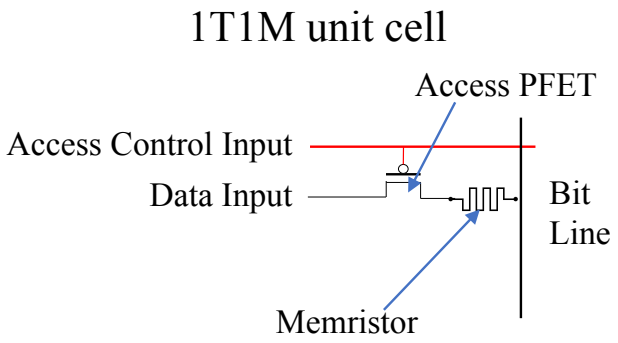


Fig. 3. Unit cell circuit schematic.

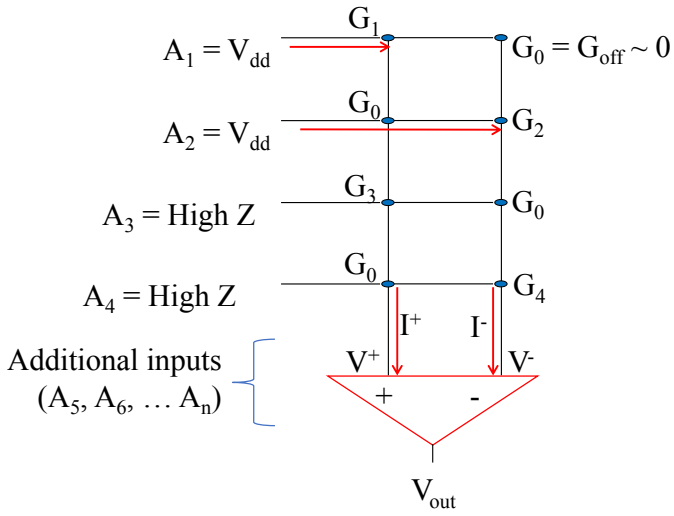


Fig. 4. Differential current architecture with 1/HighZ inputs (DZ neuron).

currents (I^+ and I^-) therefore represent the total positive and negative multiply-accumulate values for the neuron. The two currents are compared using a differential current comparator to form the threshold gate. If I^+ is greater, the output is high (V_{dd}), while if I^- is greater, the output is low (V_{ss}); thus the comparator performs the neuron threshold evaluation function and outputs a digital signal.

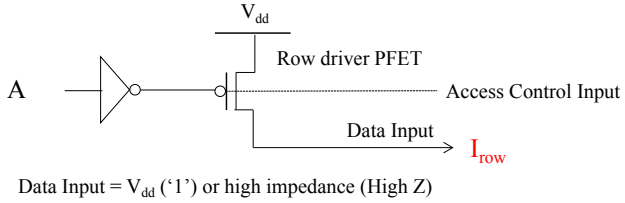


Fig. 5. Row driver circuit schematic that implements the 1/High Z input.

of A_3 and A_4, the output of the row driver is a high impedance (High Z), rather than V_{ss}. This means no current will flow into the memristor (power-efficient operation). When the input is a logical high, such as A_1 and A_2, the input is driven to V_{dd}. For the DZ neuron, every input is connected to both a positive column and a negative column of memristors. If the synaptic weight is positive, the memristor in the positive column is set to a conductance value directly corresponding to the desired integer weight value (such as G_1 and G_3). Similarly, if the weight is negative, the memristor in the negative column is set (G_2 and G_4). The memristor in the other column is set to its off conductance value ($G_0 = G_{off} = 0.01$ µS = 100 MΩ). Following from Ohms Law (I = G x V), the current through a memristor represents the weighted dot product of its input. The outputs from the memristive crossbar are two currents, I^+ and I^-, each summed separately by connecting the memristors to their respective column line. These two

III. Comparator Design and Analysis

The desired design will be compact, low-power, and fast. If this can be achieved, this circuit can be used for each neuron, instead of multiplexing a large, power-hungry Analog to Digital Converter (ADC) across many neurons [5]. Consider the memristor array itself. In 45 nm technology, the 1T1M unit cells require 995.3 µm^2 for a 128 x 32 array (128 inputs, 32 neurons); without the access transistor it needs only 66.4 µm^2. However, this 15X increase in area using the 1T1M unit cell only translates into a 1.8X increase in overall area, and enables our comparator design to be much smaller. Even with our compact comparator design (55.1 µm^2), the comparator circuit takes up almost 50% of the area in our architecture. A power analysis [8] indicates the unit cells in this size array would consume 276 µW, while our comparator consumes 431

μW (roughly 60% of the total). So even with our design, the comparator is still a very large factor in the overall area and power of the array. For directly connected neurons, the timing is dominated by the comparator speed, as the memristor array contributes only a few hundred ps to the overall delay (see Fig. 13, ΔV_{diode} vs. time graph). Its overall impact on timing can be less in other architectures, depending on the specific on-chip network used and the size of the neural network.

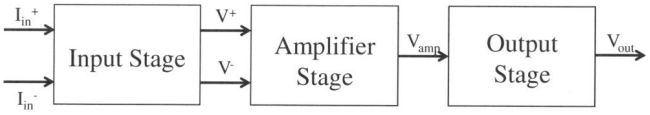


Fig. 6. Comparator Architecture.

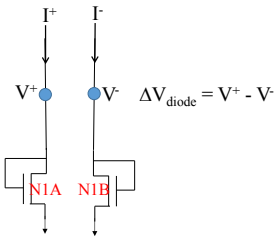


Fig. 7. Simplified view of the input stage of the comparator.

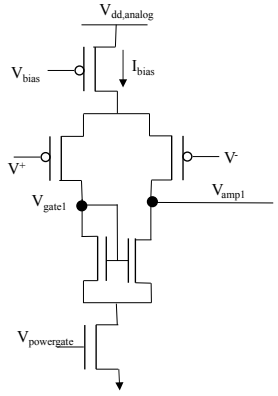


Fig. 8. Differential Amplifier.

Our comparator architecture is shown in Fig. 6. The input stage receives the two input currents (I^+ and I^-) from the memristor array, and individually transduces them into two voltages (V^+ and V^-). The differential voltage ($\Delta V_{diode} = V^+ - V^-$) is amplified by the amplifier stage to create a voltage signal (V_{amp}) that can be buffered and latched by the output stage (V_{out}) for driving the data onto the digital communication network, or to directly drive additional neurons. The basics of the design are easily described. The input stage is built using an FET with its drain and gate connected to create a diode-connected FET (one each for the positive and negative inputs); see Fig. 7 . In this configuration, as the two currents into each diode (I^+ and I^-) increase, their voltage outputs (V^+ and V^-) increase proportionally. These two output voltages become the inputs for the amplifier stage.

The amplifier stage is based on a simple differential voltage amplifier; see Fig. 8. The basic operation is as follows. The bias current (I_{bias}) created by V_{bias} is steered preferentially down one of the two parallel paths depending upon the differences in the input voltages. Small input voltage differences create large output voltage differences due to this steering. By tying V_{gate1} to the gates of the two NFETs, we keep this voltage relatively constant, and force V_{amp1} to swing over a wide range, since the FETs are in a saturated mode. V_{gate1} and V_{amp1} are the inputs into a second differential voltage amplifier. Two amplifiers in series will be used to provide enough overall gain so that the output voltage of the second amplifier (V_{amp2}) is driven close to V_{dd} or V_{ss}. $V_{powergate}$ is used as a power reducing control voltage that shuts off the bias current if the comparator is not being used.

The output stage is a simple latched buffer, with four inverters in series, each properly scaled in size to efficiently drive the output load of the on-chip network. Since this is a digital output, it can be used with any type of on-chip communication network, or can directly drive inputs to additional neurons residing in other memristor arrays in the neural network. In designing the comparator and analyzing its capabilities, we can use some specific properties to our advantage:

1. The weights in the neural network must be programmed, and are therefore known in advance; for any given set of weights, the comparator needs to operate correctly in only a subset of the total range required.
2. Memristors are programmable conductance devices that can be incorporated into the design to ensure correct operation even under device and parameter mismatch conditions.
3. Corner case operating conditions that create inaccurate functionality can be tolerated if they rarely occur; the overall neural net accuracy can still be very good (perhaps unaffected).

A. Design enhancements to the input stage

Fig. 10 shows the detailed schematic for the input stage. Since the comparator in a DZ neuron architecture is required to sink the currents from the array, it has to be capable of handling the total current while still being able to discriminate the minimum weight difference possible. Since the memristor conductances are precisely set to integer values, the minimum weight difference equals 1 (ΔG_{min} = 1 μS). The N1A and N1B FETs are wired to create a diode function, which is repeated multiple times in parallel to allow larger values of current to be passed into the comparator if necessary. A control voltage (V_{ctrl2} in the figure) enables individual pairs of diodes to be activated or de-activated. The total weight (maximum possible conductance) is known in advance since we assume off line

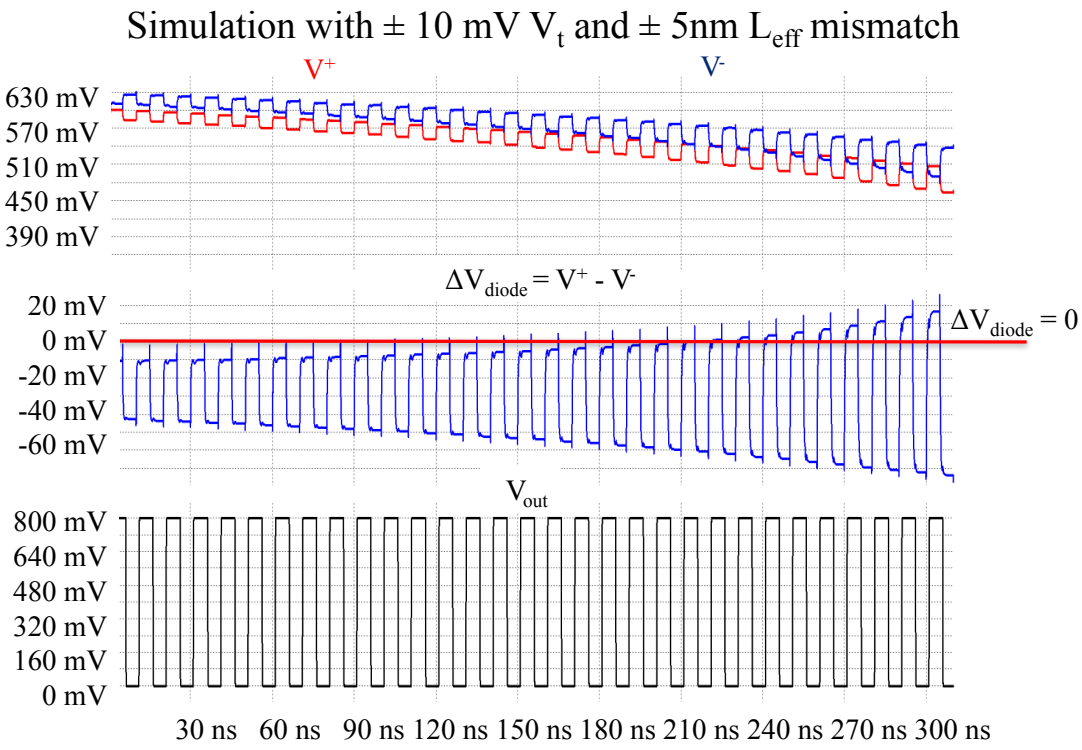


Fig. 9. Effect of calibration procedure for handling mismatch. Top plot shows V^+ and V^-, the middle plot is the differential voltage ($\Delta V_{diode} = V^+ - V^-$); note that it is almost always negative. The bottom plot is the comparator output, using a 5 ns clock. The output signals are 100% correct and very clean.

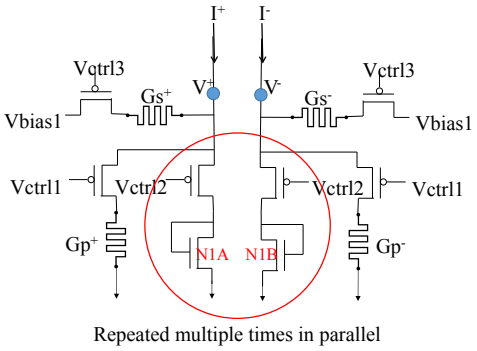


Fig. 10. Detailed input stage circuit. Only one pair of the parallel sets of FETs wired to form a diode (N1A, N1B) is shown.

training is used. By controlling the number of diodes activated, we controllably and evenly split the total current across the active sets of diodes, and the design can operate in its desired voltage range under most input conditions. We have found that using 4 sets of diodes provides a reasonable balance when trading off area, sensitivity, and control of the voltage range. Under the simulation conditions reported, we can sense of difference of 4 nA with 16 µA of input current.

Another enhancement is to include memristors in the design for calibration purposes. One set of two memristors (G_s^+, G_s^-) is connected to the bit lines (like a data input). The other set of two memristors (G_p^+, G_p^-) is in parallel with the diodes. These memristors can be programmed in a manner similar to the array weights, and enable fine-grained adjustment of the differential voltages (V^+, V^-) to compensate for device and parameter mismatches. The series memristors have greater effect when V^+ (or V^-) is low; the parallel memristors have greater effect for higher V^+ (or V^-). The proper memristor values can be found as part of a chip calibration procedure, which is done after chip fabrication but before setting the desired programming weights into the array. A suitable procedure is:

- Set the weights to create equal numbers of +1 and -1 values, and set all inputs high
- At each major clock cycle (10 ns is used in Fig. 9), cycle one +1 weight, and then one -1 weight by turning its input off, then on (each for 5 ns)
- After this, reduce both the total positive and negative weights by 1 by setting their inputs to High Z
- Repeat until all the inputs are at High Z
- Adjust the G_p and G_s devices based on output errors:
- "0" errors (output is high instead of low) that occur with many inputs on can be reduced by increasing G_p^+
- "1" errors that occur with few inputs on can be reduced by an increase in G_s^+
- G_p^- and G_s^- are adjusted for the opposite conditions
- Repeat this procedure until all errors are eliminated

Other calibration procedures can be used, depending on the

desire for fully operational comparators vs. allowance for incorrect operation under extremely rare conditions. The simulation results shown used LTspice [13], 45nm technology device parameters [14], [15], and the memristor models described in [16], [17]. Up to ± 10 mV (20 mV total) V_t mismatch and up to ± 5nm (10 nm total) dimensional mismatch can be tolerated. A simulation showing correct comparator operation for this level of mismatch is seen in Fig. 9.

B. Design enhancements to the amplifier stage

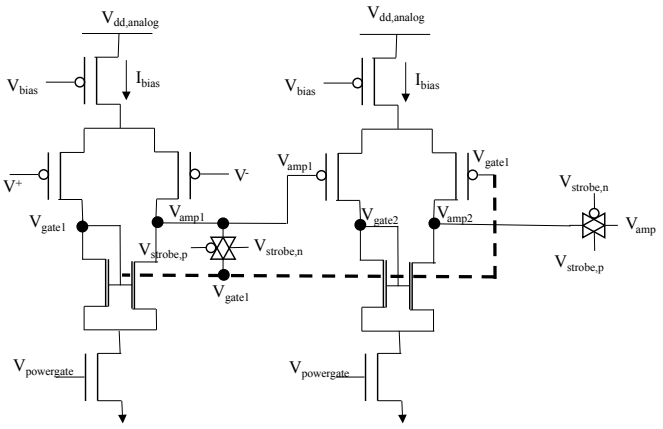


Fig. 11. Amplifier stage of the comparator with modifications.

The simple 2-stage differential amplifier can be modified to improve the comparator performance (see Fig. 11). Let us consider the basic operation. As the two currents I^+ and I^- flow into the diodes, they raise the drain voltages (V^+ and V^-). The small difference in the currents caused by the differing weighted sums creates a small voltage difference (ΔV_{diode}); this differential voltage is amplified to create a much large voltage difference between V_{gate1} and V_{amp1} (V_{gate1} stays relatively constant, while V_{amp1} swings over a large range). The second amplifier is used to drive its output (V_{amp2}) to V_{dd} and V_{ss}. The speed of this circuit is mainly dependent on the first amplifier, and is primarily determined by two factors:
1. How fast can the bias current swing V_{amp1}?
2. How much does the bias current need to swing V_{amp1}?
This can be expressed using the simple fact that V_{amp1} has a node capacitance: therefore

$$I = C * \Delta V/\Delta t \tag{1}$$

or

$$\Delta t = C * \Delta V/I \tag{2}$$

To reduce Δt, you have to increase I or reduce ΔV. The appropriate bias current to use is essentially a design optimization: higher bias currents can swing the output faster, but take more power and create a larger design (increased capacitance) that will slow down the amplifier. The ΔV value is input-pattern dependent. In a situation where the previous weighted sum is highly positive and the new weighted sum = -1, ΔV (V_{amp1} - V_{gate1}) is initially very large and positive, but the final V_{amp1} will be just slightly lower than V_{gate1}: the amplifier input voltage difference (ΔV_{diode}) is now very small, so the bias current therefore needs a relatively long time to switch V_{amp1} past V_{gate1}; only then will the second amplifier switch as well (see Fig. 12). This effect can be mostly mitigated by the amplifier design. By adding a transmission gate between V_{amp1} and V_{gate1}, we can controllably force V_{amp1} equal to V_{gate1}; this allows us to make ΔV_{amp1} very small and relatively constant under all input conditions. This is done by using a strobe signal that turns this T-gate on during the early part of the comparison operation, and turns it off during the later part. The memristor array inputs and input stage diodes are always on. While this T-gate is on, a second T-gate (connecting V_{amp2} to the output stage) is turned off to ensure a stable V_{out} signal from the latched buffer. The T-gates are turned off (or on, respectively) during the latter portion of the clock cycle.

The simulations in Fig. 12 and Fig. 13 are for the same array and inputs, with equal time scales. The comparator in the first plot does not have the T-gate/T_{strobe} enhancement. Under a worst-case condition the previous sum value is very positive and the new sum has a minimum value of -1 (ΔG = -1 µS); V_{amp1} must swing very far to cross V_{gate1} (from 900 mV to 400 mV). This takes a long time (ΔT = 6.865 ns). In the second plot, we see that the comparator with the T-gate/T_{strobe} included forces V_{amp1} to equal V_{gate1} almost immediately; this greatly reduces the delay (ΔT = 1.559 ns). Fig. 14 and Fig. 15 show how this affects the comparator speed for a single neuron across a large set of input test vectors. Without the T-gates (Fig. 14), the speed is dependent on both the previous inputs and the current inputs. With the new architecture (Fig. 15), V_{amp1} always starts very close to V_{gate1} and the worst-case time delay is drastically reduced; the comparator speed is now relatively independent of the inputs. The T_{strobe} time increases the delay for very fast transitions, but these do not define the comparator speed. Simulations indicate T_{strobe} = 1.1 ns is optimal and the sensitivity to timing variations is small (see Fig. 16); therefore, T_{strobe} = 1ns is a good, practical choice.

C. Power supply and temperature analysis

A simple test neuron, with 256 inputs and a weight distribution representative of an MNIST input layer, was used to examine the performance of the comparator over a range of supply voltages and temperatures.

Table I shows timing results for $V_{dd,digital}$ from 0.8 to 1.0 V, and $V_{dd,analog}$ from 1.0 to 1.2 V (V_{bias} = $V_{dd,analog}$/2). See Fig. 11 for reference to these voltages. Table II provides power consumption for the comparator for the same range of supply volatges. Table III shows how operating temperature affects the power of the comparator and speed of the neuron; the supply voltages used for these simulations were $V_{dd,digital}$ = 0.80 V, and $V_{dd,analog}$ = 1.00 V (operating voltages with the slowest timing).

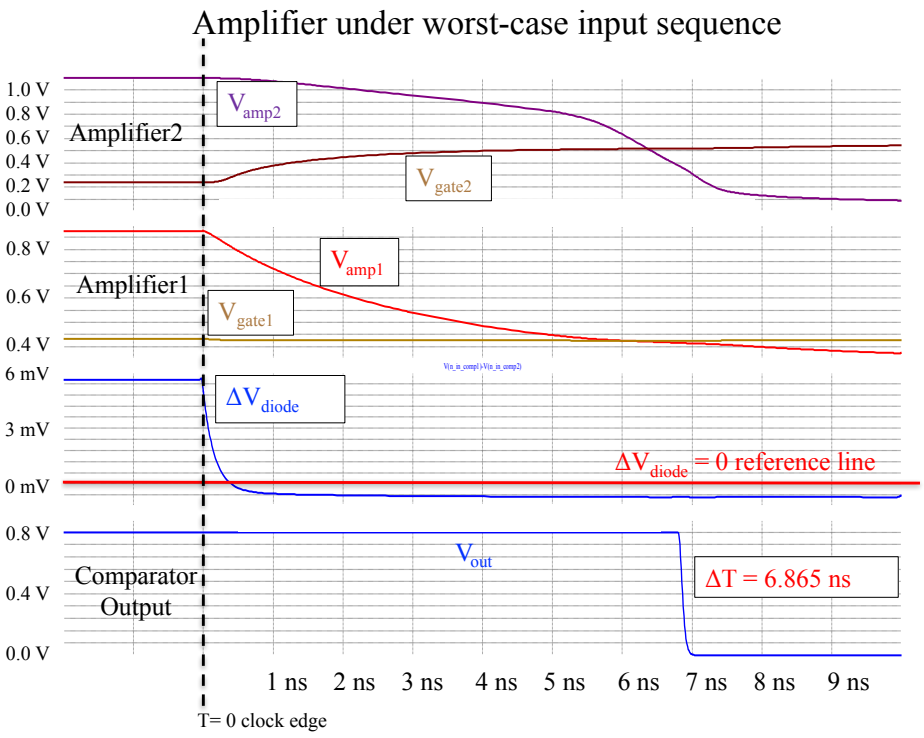


Fig. 12. Timing plot for comparator with no T_{strobe}.

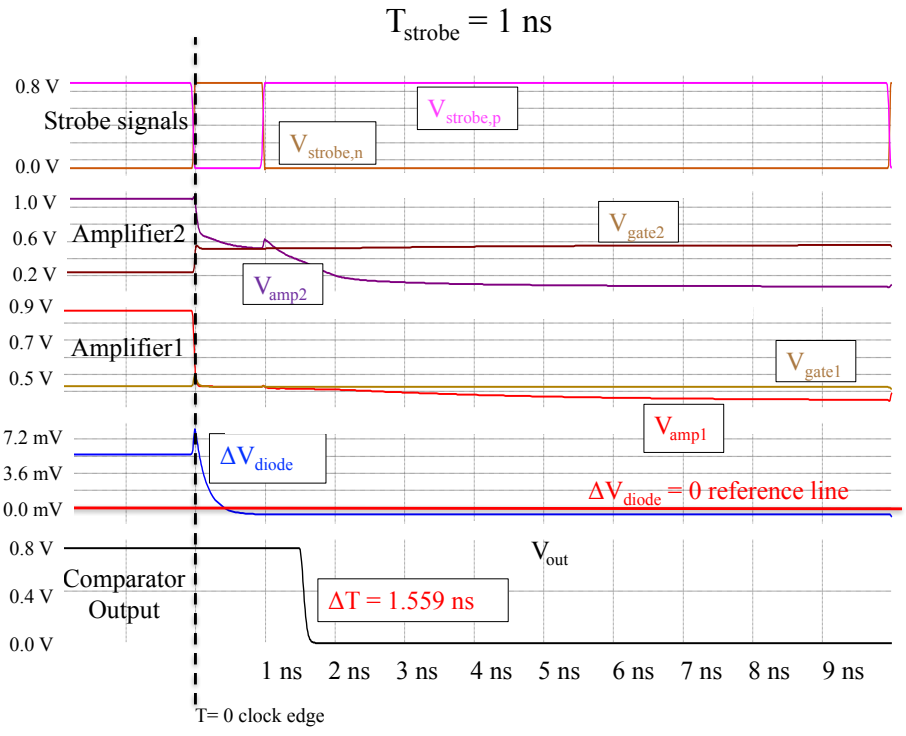


Fig. 13. Timing plot for comparator with T_{strobe} = 1 ns.

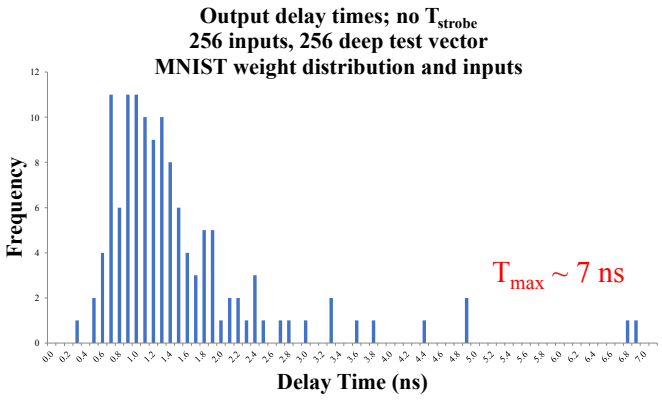


Fig. 14. Output delay times for a single neuron and no T_{strobe}. The distribution is wide, and includes very long delays.

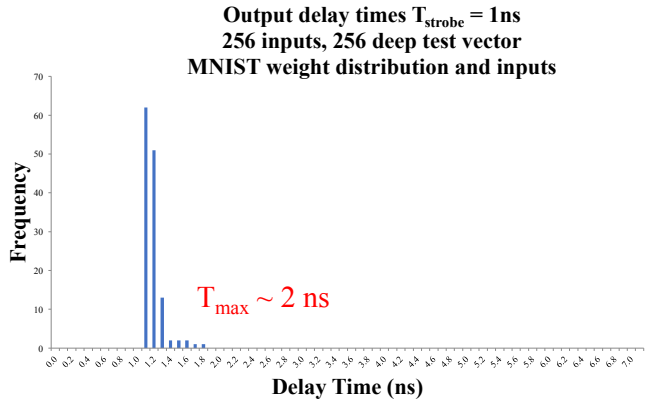


Fig. 15. Output delay times for a single neuron and T_{strobe} = 1 ns. The distribution is tight, and T_{max} is below 2.0 ns.

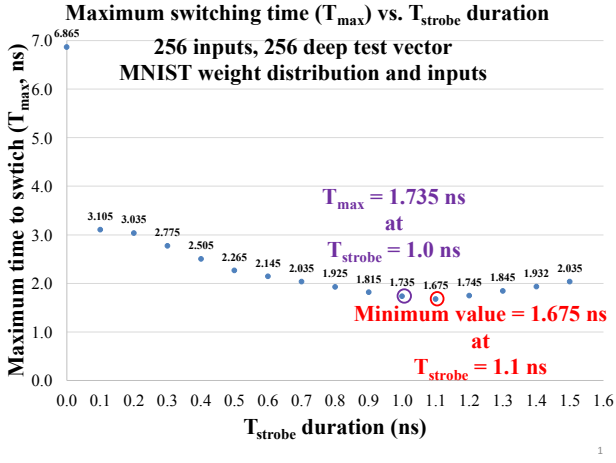


Fig. 16. Variation of comparator speed vs T_{strobe}.

TABLE I
NEURON TIMING (WORST-CASE DELAY IN NS AS A FUNCTION OF CHANGING SUPPLY VOLTAGES)

$V_{dd,digital}$		0.8	0.9	1.0
$V_{dd,analog}$	1.0	1.685	1.602	2.045
	1.1	1.555	1.375	1.408
	1.2	1.485	1.345	1.245

TABLE II
COMPARATOR POWER IN μW AS A FUNCTION OF SUPPLY VOLTAGES)

$V_{dd,digital}$		0.8	0.9	1.0
$V_{dd,analog}$	1.0	8.9	9.2	11.3
	1.1	15.3	14.4	13.8
	1.2	27.5	25.9	24.8

D. Final comparator results

The results indicate that an operating frequency of 250 MHz for neuron evaluation in the crossbar is practical, and 300 MHz is possible (only one T_{max} exceeded 3.333 ns across all the simulations presented). With a more realistic crossbar array architecture (256 inputs, 16 neurons), 300 MHz operation can be sustained if comparator errors at a rate of 0.05% are allowed. The comparator consumes 15.3 μW ($V_{dd,digital}$ = 0.8 V, and $V_{dd,analog}$ = 1.1V). At 250 MHZ, this represents 0.061 pJ per comparison operation. Higher speeds and energy efficiency can be gained by using higher supply voltages (see Table IV). Because the memristor array and amplifier are analog circuits and operating continuously, the energy used is proportional to

TABLE III
COMPARATOR POWER AND NEURON TIMING AS A FUNCTION OF TEMPERATURE ($V_{DD,DIGITAL}$ = 0.8 V, $V_{DD,ANALOG}$ = 1.0 V)

Temperature (C)	Power (μW)	T_{max} (ns)
0	11.3	1.435
25	9.0	1.665
50	7.8	2.065
75	7.0	2.685
100	6.6	3.535

TABLE IV
COMPARATOR POWER AND NEURON TIMING AS A FUNCTION OF TEMPERATURE ($V_{DD,DIGITAL}$ = 1.0 V, $V_{DD,ANALOG}$ = 1.2 V)

Temperature (C)	Power (μW)	T_{max} (ns)
25	25.3	1.291
100	14.7	2.235

the clock period; if the circuit speed increases at a faster rate than the power, a favorable energy efficiency trade-off can be made. At these higher supply voltages, an operating frequency of 400 MHz operation is practical; by tolerating comparator errors at the rate of about 1%, 500 MHz operation is possible. At 500 MHz, the energy efficiency is 0.050 pJ/comparison.

The area of the comparator is estimated at 55.1 μm^2, calculated using the following methodology:

- Calculate the gate area (W x L) of each transistor in the design
- Sum up the total gate area of the transistors and multiply this amount by 35X

The 35X factor was initially determined by reviewing actual SRAM sizing as a function of technology nodes [18]. It was then validated by looking at the physical size of layouts of comparators similar to the one described here, and from design information in [19].

By designing the comparator specifically for the TGN used, we have improved the power and area (at an equivalent operating frequency of 300 MHz) over the general-purpose ADC from [5] by 31x and 5x, respectively. This particular ADC comparison was chosen because that circuit was selected for use in a neural net architecture using memristor crossbar arrays, and it has complete power, area, and timing results.

As another point of comparison, consider the circuit in [22]. This is also a differential current comparator, and has a very similar architecture (input, amplifier, output stages) and similar circuits for the three stages. However, it was not designed to be used in a memristor-based crossbar architecture; it assumes symmetric pulsed inputs or sine wave inputs. This distinction will become important when comparing the speed to our design. The power for this comparator is 270 μW in 180 nm technology. Scaling to a 45 nm design suggests an equivalent technology power of 70-100 μW, depending upon scaling assumptions (4.5X - 6.5X higher than our design). The paper does not provide information on the area, but it uses an input stage design based upon a similar diode-connected configuration as ours, albeit with additional FETs. It utilizes an additional reference current source of similar magnitude to the input currents, which would require FETs of a similar size to the diodes. The architecture also assumes that two copies of their circuit are needed, one for each of two input currents. Since the input stage is the largest component of the design (around 70% for our comparator), it is likely this comparator is at least 3X larger than ours. The speed of the design is specified as 1 GHz, based on an average propagation delay of 0.87 ns. However, this is misleading when comparing against our design, for at least 3 reasons (2 minor, 1 major). The two minor reasons are:

- The stated delay does not include the time needed for the input values into a memristor crossbar to reach the comparator circuit, approximately 500 ps
- The delay reported is an average of the rise time and fall time delays, the worst-case delay specified in the paper is actually 1.13 ns

These two items indicate the propagation delay would be 1.6 ns or greater if used in our crossbar architecture, which is 2X the value stated in the paper. A much more significant issue is the fact that the input patterns used to analyze this design are very different than what are actually seen in a memristor crossbar. Since the referenced design also uses a differential voltage amplifier, it will experience a very similar input-pattern dependence on the worst-case delay. The results provided were based on either a sine wave input, or a pulsed input with large and symmetric positive and negative current swings relative to the reference. As shown in our paper, the input-pattern dependence has a huge impact on the worst-case delay. An initial analysis suggests this design would have a worst-case delay of 10-15 ns; which 3X - 5X slower than our design.

IV. Value of the Comparator Design

A crossbar approach can be quite efficient because it packs memristors into a tight, regular grid and shares input circuitry and on-chip network connections across many neurons. However, if a single array size is used for all layers and parts of a neural network, overall resource utilization can drop significantly. In the specific case of the cybersecurity application described in [8], we found that widely ranging requirements on crossbar size led to poor utilization. Some neurons required at least 250 inputs, while others needed as few as eight. When mapped into crossbars with 256 inputs and 128 neurons per crossbar, the neuron utilization per crossbar ranged from three to the full 128. Overall, we found that only 80% of the crossbar inputs, 2% of the synaptic memristors, and 61% of the neurons were used.

Creating crossbars that perfectly fit the needs of a given application would eliminate this resource underutilization, but would result in a custom chip that would be poorly suited or unusable for other applications. Using the approach of a single, fixed-sized crossbar allows a more general-purpose chip to be designed, but leads to underutilization. The use of the 1/High Z differential current architecture, and our comparator design, enables an important architectural option, which we call *crossbar tiles*. A tile is a small, but complete, memristive crossbar and its supporting circuitry. Each tile has additional circuitry to enable tiling and the fabric between tiles supports intertile connectivity. Tiles can function as a standalone crossbar or can be combined with other tiles to form larger crossbars with different input to neuron ratios. A wide variety of composite crossbars can be composed of tiles of a single uniform size, combined in different amounts both horizontally and vertically. The tile to crossbar assignment is configurable on chip after the neural network is defined, rather than fixed when the neural processor is designed and fabricated.

A. Tiling of Memristive Crossbars

The DZ architecture and our comparator enable the use of control PFETs to pass the differential current to a comparator in a different array. Keep in mind that the current being

passed represents the weighted sum of the inputs; therefore the function of the neural net is maintained. The second array is now evaluating its inputs *plus* the inputs from the first array; the two arrays are combined into a single neuron (or set of neurons). Figure 17 shows the basic concept; this feature means that smaller arrays (tiles) can be connected to form much larger arrays. For example, four 64 x 16 tiles can be connected together to make a 128 x 32 array (see Figure 18).

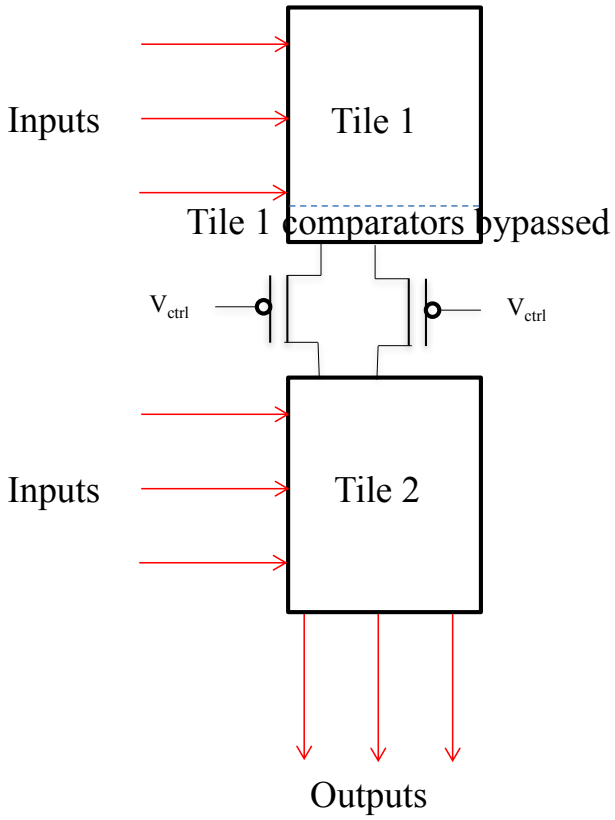


Fig. 17. Use of control FETs that enable two smaller arrays (tiles) to be combined into a single array.

Here the inputs go into Tile 1 and Tile 3; they are also sent across to Tile 2 and Tile 4. The comparators in Tile 1 and Tile 2 are shut down, and the current (the sum of the weighted inputs) is passed to the comparators in Tile 3 and Tile 4, which now sum up all of the weighted inputs to create the final outputs.

The tile concept is further enhanced by the ability to control the current (and therefore the power) in the unused portions of the tile (unit cells, comparators). Simulations of a 256 x 32 array show that the active power can be completely eliminated; the leakage power is an extremely small fraction (much less than 1%) of the total. There are drawbacks to using the tiles; the input circuits may need to send the input value across a tile to a neighboring tile, through another driver/latch circuit. This adds a small delay ($\approx$ 30 ps per tile); as long as the number of horizontal tiles connected is reasonable (10 or so), the effect on

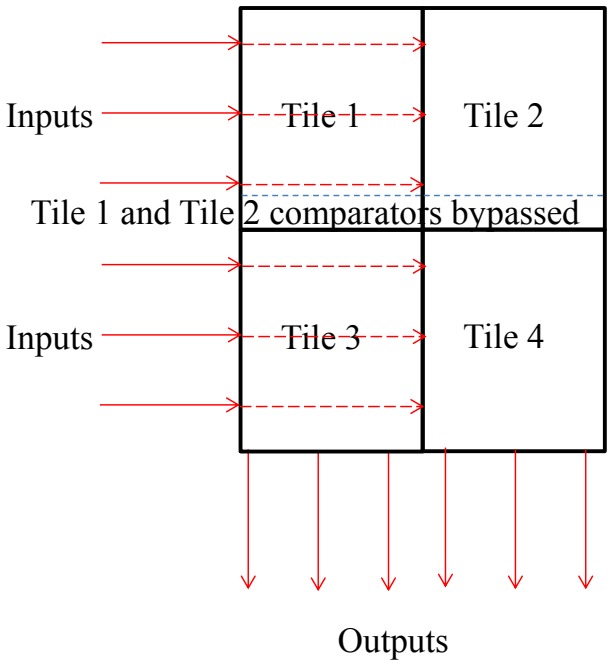


Fig. 18. Four 64 x 16 tiles combined to form a 128 x 32 array.

performance is modest. There is little additional delay incurred by adding tiles vertically (increasing the number of inputs), since the current flow through the memristors and into the comparator input stage happens within 300-500 ps, which is significantly shorter than the T_{strobe} time. Another point to be made is that the control PFETs added to the comparator design may need to pass a lot of current, and are therefore large (W/L = 1200/45 nm). This adds about 3.8 μm^2 in area to the comparator – we have included this extra area in our comparator analysis.

B. Chip Level Impact of Tiles

The use of tiles enables a more complete utilization of the chip resources, improving the capabilities of a general-purpose neural processor. The primary criteria for any design is performance, but it is usually better to evaluate options on a normalized basis, in terms of Performance/Watt or Performance/Cost [5], [20]. Since neural nets typically are used for streaming applications, the proper performance metric is application throughput, defined as the number of new inputs per second. While costs for a design can be estimated, using chip area is a reasonable proxy [20]. This provides us with two primary metrics:

- Throughput/Watt (T/W)
- Throughput/Area (T/A)

As a point of comparison, for MNIST, our architecture as described is expected to require roughly 32 pJ/classification [21]. A detailed analysis of a general-purpose neural processor executing inference on a variety of neural network based applications using the tile feature demonstrates clear advantages

in both T/W and T/A compared to one that does not use this feature [8]. In this analysis, optimal designs using tiles are compared to optimal designs without using tiles (arrays). As shown in Table V, designs using composable 128 x 32 tiles are $\geq$ 1.5X better in T/W, and $\geq$ 1.8X better in T/A, compared to non-composable designs using 512 x 32 arrays.

TABLE V
TILE VS. ARRAY COMPARISON

Design Type	T/W (Gbps/W)	T/A (Gbps/mm^2)
128 x 32 Tile	219.5	59.5
512 x 32 Array	147.9	33.2

V. SUMMARY AND CONCLUSIONS

A compact, power-efficient, and fast comparator design was presented and analyzed. Nominal power, area, and timing estimates were made, showing this targeted design is better than using a general-purpose ADC:

- Power = 15.3 μW; 133x improvement
- Area = 55.1 μm^2; 22x improvement
- Frequency = 300 MHz; 0.25x decrease

This compares the two designs directly, as standalone circuits, as opposed to at iso-speed, or as expected to be used in a specific architecture. Although our design is somewhat slower, it is significantly smaller and substantially more power-efficient. The speed can be improved by 2X at the cost of 60% higher power. This targeted design also enables the use of a tiled architecture to improve chip resource utilization. Using tiles, clear improvements in both T/W (1.5X) and T/A (1.8X) are seen when compared to our designs without the tile feature incorporated.

VI. ACKNOWLEDGMENTS

The author wishes to thank Al Scalo for his help in analyzing the comparator results.

REFERENCES

[1] W. Wilcke. The IBM cortical learning center project. *Neuro-Inspired Computational Elements (NICE)*, 2015.

[2] Chen et al. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 52(1):127–138, January 2017.

[3] Chen et al. Dadiannao: A machine-learning supercomputer. *Proceedings of MICRO-47*, pages 609–622, December 2014.

[4] B. Reagen et al. Minerva: Enabling low-power, high-accuracy deep neural network accelerators. *ACM/IEEE International Symposium on Computer Architecture*, June 2016.

[5] A. Shafiee et al. Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. *ACM SIGARCH Computer Architecture News*, pages 14–26, June 2016.

[6] M. Hu et al. Memristor crossbar-based neuromorphic computing system: A case study. *IEEE Transactions on Neural Networks and Learning Systems*, 25(10):1864–1878, October 2014.

[7] R. Hasan and T. Taha. Enabling back propagation training of memristor crossbar neuromorphic processors. *International Joint Conference on Neural Networks*, pages 21–28, 2014.

[8] D. Mountain et al. Memristor Crossbar Tiles in a Flexible, General Purpose Neural Processor. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 8(1):137-145, March 2018.

[9] R. Hasan et al. High throughput neural network based embedded streaming multicore processors. *IEEE International Conference on Rebooting Computing*, October 2016.

[10] A. Chen Nonlinearity and Asymmetry for Device Selection in Cross-Bar Memory Arrays. *IEEE Transactions on Electron Devices*, 62(9):2857-2864, September 2015.

[11] F. Alibart et al. High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm. *Nanotechnology*, 23:1-7, January 2012.

[12] M. Hu et al. Dot-Product Engine for Neuromorphic Computing: Programming 1T1M Crossbar to Accelerate Matrix-Vector Multiplication. *Design Automation Conference*, June 2016.

[13] LTspice IV can be downloaded from the Linear Technologies web site at http://www.linear.com/designtools/software/

[14] D. Mountain. Technology considerations for neuromorphic computing. *IEEE International Conference on Rebooting Computing*, October 2016.

[15] High performance models incorporating high-k/metal gate and stress effects; 45nm PTM HP model:V2.1, found at http://ptm.asu.edu/latest.html

[16] C. Yakopcic et al. Memristor SPICE Model and Crossbar Simulation Based on Devices with Nanosecond Switching Time. *International Joint Conference on Neural Networks*, August 2013.

[17] C. Yakopcic et al. Memristor Model Optimization Based on Parameter Extraction from Device Characterization Data. *to be published in the IEEE Transactions on Computer Aided Design*, 2019.

[18] K. Zhang. Memory trends. *International Solid State Circuits Conference (ISSCC)*, 2013-2015.

[19] P. Merolla et al. A Million Spiking-Neuron Integrated Circuit with a Scalable Communication Network and Interface. *Science*, 345(6197):668-673, August 2014.

[20] D. Mountain. Analyzing the value of using three-dimensional electronics for a high-performance computational system. *IEEE Transactions on Advanced Packaging*, 31(1):107–117, February 2008.

[21] C. Krieger et al. Relative Efficiency of Memristive and Digital Neuromorphic Crossbars. *Proceedings of the International Conference on Neuromorphic Systems*, July 2018.

[22] S. Patro et al. 1 GHz High Sensitivity Differential Current Comparator for High Speed ADC. *Journal of Digital Integrated Circuits in Electrical Devices*, 2(1):7-12, January 2017.

FPGA demonstrator of a Programmable Ultra-efficient Memristor-based Machine Learning Inference Accelerator

Martin Foltin
Silicon Design Lab
Hewlett Packard Enterprise
Fort Collins, Colorado, USA
martin.foltin@hpe.com

Craig Warner
Silicon Design Lab
Hewlett Packard Enterprise
Plano, Texas, USA
cwarner@hpe.com

Eddie Lee
Silicon Design Lab
Hewlett Packard Enterprise
Plano, Texas, USA
eun.sub.lee@hpe.com

Sai Rahul Chalamalasetti
Hewlett Packard Labs
Hewlett Packard Enterprise
Palo Alto, California, USA
sairahul.chalamalasetti@hpe.com

Chris Brueggen
Silicon Design Lab
Hewlett Packard Enterprise
Plano, Texas, USA
chris.brueggen@hpe.com

Charles Williams
Silicon Design Lab
Hewlett Packard Enterprise
Plano, Texas, USA
charles.hen.williams@hpe.com

Nathaniel Jansen
Silicon Design Lab
Hewlett Packard Enterprise
Houston, Texas, USA
nathaniel.w.jansen@hpe.com

Felipe Saenz
Silicon Design Lab
Hewlett Packard Enterprise
Heredia, Costa Rica
felipe.saenz@hpe.com

Luis Federico Li
Silicon Design Lab
Hewlett Packard Enterprise
Heredia, Costa Rica
luis.li@hpe.com

Glaucimar Aguiar
Brazil Labs
Hewlett Packard Enterprise
Barueri, Brazil
glaucimar@hpe.com

Rodrigo Antunes
Brazil Labs
Hewlett Packard Enterprise
Porto Alegre, Brazil
rodrigo.antunes@hpe.com

Plinio Silveira
Brazil Labs
Hewlett Packard Enterprise
Porto Alegre, Brazil
plinio.silveira@hpe.com

Gustavo Knuppe
Brazil Labs
Hewlett Packard Enterprise
Porto Alegre, Brazil
gustavo.knuppe@hpe.com

Joao Ambrosi
Brazil Labs
Hewlett Packard Enterprise
Porto Alegre, Brazil
joao.ambrosi@hpe.com

Soumitra Chatterjee
SSTO RnD
Hewlett Packard Enterprise
Bangalore, India
soumitra@hpe.com

Jitendra Onkar Kolhe
SSTO RnD
Hewlett Packard Enterprise
Bangalore, India
jitendra.kolhe@hpe.com

Sunil Lakshiminarashimha
Composable Engineering
Hewlett Packard Enterprise
Bangalore, India
sunil.vl@hpe.com

Dejan Milojicic
Hewlett Packard Labs
Hewlett Packard Enterprise
Palo Alto, California, USA
dejan.milojicic@hpe.com

John Paul Strachan
Hewlett Packard Labs
Hewlett Packard Enterprise
Palo Alto, California, USA
john-paul.strachan@hpe.com

Amit Sharma
Silicon Design Lab
Hewlett Packard Enterprise
Palo Alto, California, USA
amit.sharma@hpe.com

***Abstract*— Hybrid analog-digital neuromorphic accelerators show promise for significant increase in performance per watt of deep learning inference and training as compared with conventional technologies. In this work we present an FPGA demonstrator of a programmable hybrid inferencing accelerator, with memristor analog dot product engines emulated by digital matrix-vector multiplication units employing FPGA SRAM memory for in-situ weight storage. The full-chip demonstrator interfaced to a host by PCIe interface serves as a software development platform and a vehicle for further hardware microarchitecture improvements. Implementation of compute cores, tiles, network on a chip, and the host interface is discussed. New pipelining scheme is introduced to achieve high utilization of matrix-vector multiplication units while reducing tile data memory size requirements for neural network layer activations. The data flow orchestration between the tiles is described, controlled by a RISC-V core. Inferencing accuracy analysis is presented for an example RNN and CNN models. The demonstrator is instrumented with hardware monitors to enable performance measurements and tuning. Performance projections for future memristor-based ASIC are also discussed.**

Keywords—Deep Neural Network Inference, Neural Network Acceleration, Memristor

I. Introduction

Deep Learning is showing a promise to change the computing landscape thanks to the ability to extract new insights from exponentially growing volume of data based on learned patterns. Hybrid digital-analog deep learning inference accelerators exhibit three characteristics that have the potential for significant increase of Deep Learning Performance and performance per Watt over conventional architectures. First, neural network weights are stored in situ on silicon die, eliminating communication energy and time overheads incurred by moving the weights from an external DRAM memory. Second, the weights are stored in next generation embedded nonvolatile memory that has significantly higher density and no leakage power as compared with SRAM, enabling higher computation density at a lower power. Third, the critical computations are carried out in an analog domain at the precision of the output activations, reducing energy consumed in the digital domain where calculations would

978-1-7281-5221-9/19/$31.00 ©2019 IEEE

normally need to be performed at a higher precision (and therefore higher energy) before truncating them to the output accuracy.

The benefits of neural network weights stored on-die have been demonstrated first in purely digital architectures – for example, DaDianNao [1] with weights stored in an embedded DRAM (eDRAM) showed the potential for two orders of magnitude higher performance over leading GPU architecture in 2014. The ISAAC [2] architecture introduced an in-memory analog computing - in memristor resistive memory (ReRAM) cross-bars, reporting two orders of magnitude higher performance per Watt over DaDianNao for Convolutional Neural Networks. The ISAAC energy and area efficiency has been further improved in the Newton architecture [3] by an adaptive ADC (Analog to Digital Conversion), better sharing of ADC converters between cross-bar tiles, and other optimizations. The PUMA [4] architecture replaced ISAAC in-situ multiply-accumulate (IMA) units with simple programmable compute Cores, enabling more flexible mapping of neural network models on hardware architecture. Instead of static mapping of the neural network model with synchronous execution of IMA units, the PUMA cores are independently running and synchronized by memory valid attributes (data access count fields) stored in the shared Tile Memory along with the data. This enables mapping of a broad range of neural network models to the hardware, using in-house developed compiler that takes advantage of the flexible PUMA Core and Tile Instruction Set Architecture (ISA) [4].

In this work we present a further evolution of the PUMA architecture, advancing it from research to implementation by developing an FPGA demonstrator to serve as a software development platform and a vehicle for further microarchitecture optimizations. In process of developing a Register Transfer Logic implementation of the PUMA architecture [4], additional micro-architectural details needed to be addressed at compute Core, Tile, and Full Chip level that

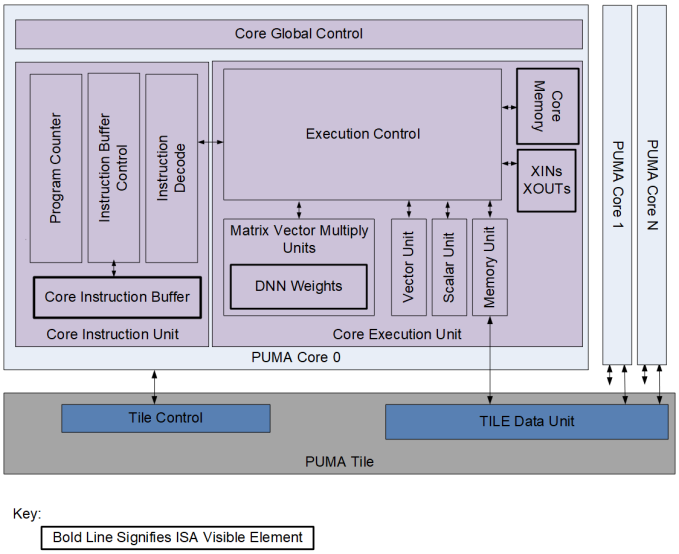


Figure 1: PUMA Core Architecture

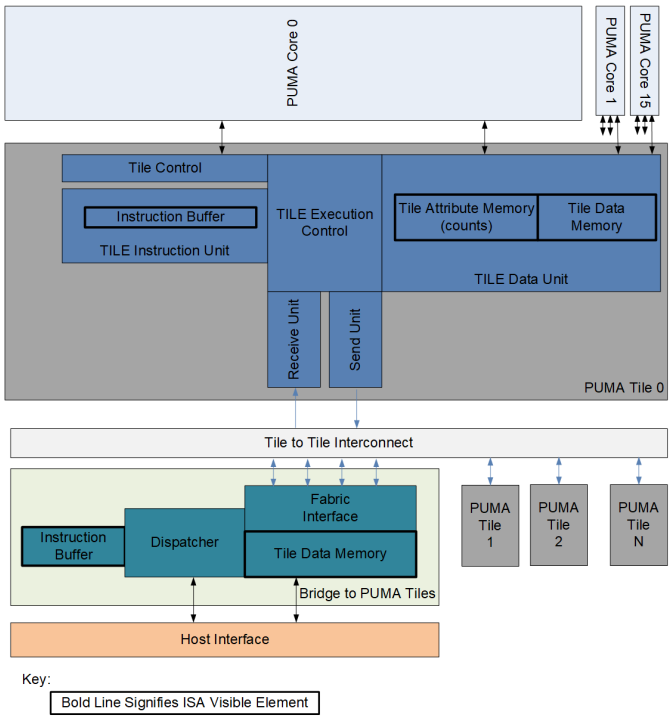


Figure 2: PUMA Tile and Bridge Architecture

were not considered before and are described in this work. We adopted a staged implementation approach supporting neural network models with up to ~20 million weights in the first stage, fitting on a single mainstream FPGA (Xilinx Ultrascale Plus XCVU9P). In the Future Direction section, we discuss extending this to larger models in the future, by model spanning over multiple FPGAs.

The PUMA architecture is organized in a three-tier hierarchy: Cores, Tiles, and Nodes [4]. A Core has its own program, Memory and Execution Units. The Core Execution section includes Matrix-Vector Multiply Units, a Vector Unit, a Scalar Unit, and a Core Memory Unit. A Tile is multiple cores connected to a Tile Memory (see *Figure 1 and Figure 2*).

The Tile Memory can be accessed from any of the cores which reside in the Tile. Tiles communicate with one another by sending Datagram packets to other tiles. The Tile memory has a unique feature for managing flow control – each element in the tile memory has a count field which is decremented by reads and set by writes. A Node is multiple tiles connected via an on-chip network.

We have extended the PUMA Architecture to include the concept of a Bridge. PUMA Bridges are much like PUMA Tiles. They have a PUMA Tile Instruction buffer and a PUMA Tile Data Memory both of which are visible to the PUMA compiler, but unlike tiles they have many fabric connections to enable fast parallel execution, and no count memory since Bridges have no PUMA cores.

II. Core Architecture

Key component of our architecture is an area efficient, single threaded, programmable computing core that enables concurrent matrix-vector multiplication in up to 4 Matrix-Vector Multiplication Units (MVMUs). The core employs a three-stage in-order instruction execution pipeline (fetch, decode, execute) and fully supports the PUMA Instruction Set Architecture (ISA) [4]. Besides MVMUs, the core includes a Vector Unit supporting non-linear activation functions (ReLU, sigmoid, tanh, etc.), pooling, and other arithmetic and logic vector operations. Central element of the core is the Core Memory Unit and associated Datapath.

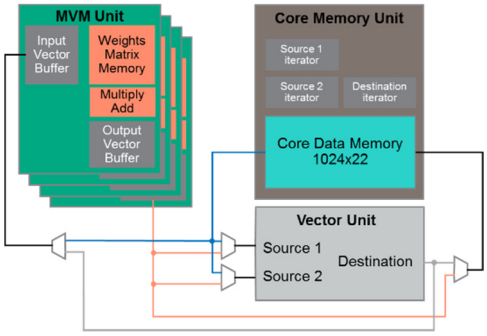


Figure 3: PUMA Core Datapath Architecture

To achieve high performance and power efficiency, output activations can flow from MVMU output buffer directly to Vector Unit without an intermediate hop through the Core Memory (see *Figure 2*). Similarly, Vector Unit outputs can be steered directly to MVMU input buffers. The load and store instructions moving data between Core and Tile memories can also target the MVMU buffers directly.

In the current implementation, the FPGA demonstrator supports input activations and weights at 8-bit precision in MVMUs, and the activation outputs are truncated to 16-bit precision supported by the Vector Unit, then to 8-bit precision for the next layer inputs. It has been shown that state of the art quantization algorithms retains the classification accuracy at 8-bit for most Convolutional Neural Networks without re-training [5], while some models require minimal re-training [6]. Our own work on model quantization supports these conclusions. We will be extending the implementation in the future to support per-layer configurable precisions of 16, 12, 8, 6 and 4-bit. We observed need for higher than 8-bit precision for some Long Short-Term Memory models (LSTM), however, the GRU model discussed in this work retains the accuracy well at 8-bit.

We considered two different implementations of MVMU on the FPGA – a direct digital model of memristor analog MVMU for the future ASIC, and an FPGA-optimized MVMU allowing to store large neural network models on a single FPGA. An analog MVMU employs activation input bit streaming and weight bit slicing to support 8- and 16-bit precisions [4]. The electric current summation along each column of an analog cross-bar can be emulated by a digital adder and register, where the register value is incremented by a weight value when the row input is 1 for a given input row bit, as the row address counter iterates over all weight matrix rows. However, due to large area penalty of FPGA logic gates, we decided to implement an FPGA optimized MVMU that leverages multipliers and adders available in FPGA hardened DSP engines. Importantly, we selected the number of multipliers used in each MVMU such that the ratio of MVM instruction execution times between the FPGA and the future ASIC is the same like corresponding ratios for the other instructions (vector operations, load, store, etc.). This enables using a constant scaling factor to project future ASIC performance from the measured FPGA performance. We optimized the implementation to allow storing up to ~20 million weights in-situ in an FPGA SRAM. The MVMU matrix size is configurable between 128 x 128 and 256 x 256 and the number of 8-bit x 8-bit multipliers per MVMU is configurable between 16 and 32. The former is used to emulate future embedded accelerator ASIC architectures that are less performance sensitive and more silicon area constrained, since most neural network models map more efficiently to 128 x 128 weight matrices. The latter is used to emulate high performance accelerators that benefit from higher degree of parallelism per core. Note that in the future analog ASIC implementation, each 256 x 256 matrix will be divided to 4 memristor cross-bars (times the number of bit slices) to keep ADC precision requirements low, consistently with design space optimizations [4].

III. Tile Architecture

A. Count Values Tile Memory

The PUMA tile memory has both a data and an attribute (i.e. count) field. The count field is used to perform synchronization across the PUMA cores. PUMA core loads and PUMA tile sends decrement the count. PUMA core stores and PUMA tile send require the count field to be zero.

We have extended the PUMA Architecture to easy handle data which want to be read many times, the maximum count value is special. A location with a count value of P_{MAX} indicates "infinite read" location. The count value for these Tile memory locations never decrements. To gracefully allow "infinite read" locations to be overwritten, stores and Tile Sends can overwrite these locations.

The *Table 1* below is a summary of how Core Reads and Core Writes are affected by the Count values and how Tiles Send operations are affected by Count values:

Table 1: Use of memory count attribute to synchronize multiple cores

Operation	Count	Behavior
Core Read	0	Read stalled until Count > 0
	$0 < x < P_{MAX}$	Read allowed; Count decremented
	P_{MAX}	Read allowed; Count NOT decremented
Core Write	0	Write allowed; Count Updated
	$0 < x < P_{MAX}$	Write stalled until count is zero
	P_{MAX}	Tile Error; Write is stalled indefinitely;
Tile to Tile (or Host) Send (Source Tile)	0	Send Stall Until Count is updated
	$0 < x < P_{MAX}$	Send Packet Sent; Count is decremented

	P_{MAX}	Send Packet Sent; Count is NOT decremented.
Tile (or Host) to Tile Send (Destination Tile)	0	Send Packet is Received and Stored into Tile memory; Count is updated
	$0 < x < P_{MAX}$	Tile Error; Write is performed; Count updated
	P_{MAX}	Tile Error; Send Packet is Received and Stored into Tile memory; Count is updated.
Host to Tile Write	0	Write Packet is received; Data and Count values are written to tile memory.
	$0 < x < P_{MAX}$	
	P_{MAX}	

B. *Send only Architecuture*

Another significant change we made to the PUMA architecture was the removal of the receive instructions. This made the tile to tile fabric considerably easier and more efficient but required the Send instruction to have a destination address. The new send instruction is of the following form:

Send(*destination_tile_address[15:0],*
source_tile_address[15:0],
count[10:0],
target[15:0],
words_to_send[13:0]);

The *target[15:0*] field in the Tile Instruction is used for Tile to Tile packet routing as well as all Bridge to/from Tile packet routing. To simplify the compiler's job we have the PUMA compiler treat all Tiles as virtual tiles. The Module Loader software alters the instructions to map virtual Tile Number Physical Chip ID and Physical Tile ID.

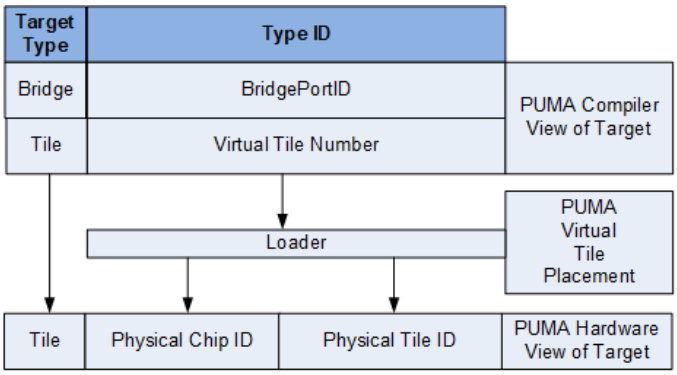


Figure 4: PUMA Target Field Use

The limited address space of the 16-bit tile address, was too limiting for our desired use cases, so we added a facility and tile instruction for controlling a source tile segment and a destination tile segment. We chose the segment register approach over extending the tile instruction register width to save area. (Less than 5% of the Tile instructions are segment register setting instructions for the DNN modules we have written) With the segment register scheme tile memory can be 2^{32} words. The large tile address space support is particularly useful for PUMA Bridges which need larger PUMA Tile Data Memory buffer to deal with Host Interface latencies.

C. *Flow Control with Send Only Architecture*

To enable flow control, we invented a barrier/ready-for-data mechanism which enables to tile software to stall the Tiles assigned to work on previous DNN levels or the Bridge interface. This extension of the PUMA ISA required the addition three new Tile Instructions: a PUMA Tile barrier instruction (*tbarrier*) and a PUMA Core "request-for-data" instruction (*rfd*). The barrier instruction has an argument specifying the number of request-for-data packet which must be received before the Tile Instruction Program Counter can proceed.

To enable higher performance the request-for-data core instruction does not stall the core. This allows for higher core efficiency, but with more complexity for the PUMA compiler. For easier adoption we also support a slow version of the ready-for-data which stalls the core (*rfds*).

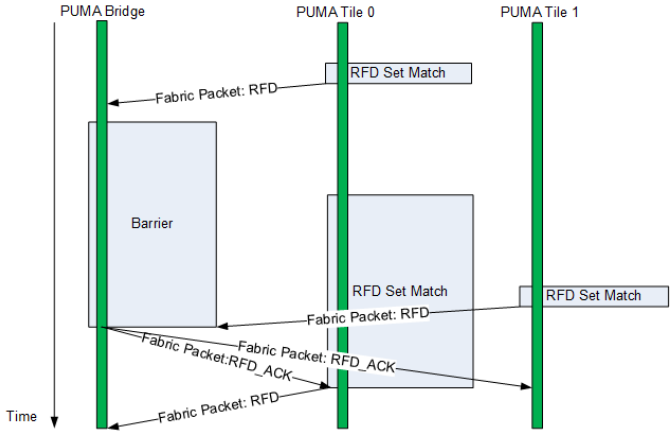


Figure 5: Tile Barrier Flow Control

Core Instructions: *rfd or rfds*

The rfd or rfds instruction is how a core tells the PUMA Tile it's ready for more data. To allow for flexibility each PUMA Tile implements several RFDs sets. Each RFD set configuration defines which cores need to execute rfd instructions in order to declare the Tile "ready-for-data." The RFD set configuration also defines which targets should be informed of the Tile's RFD declaration.

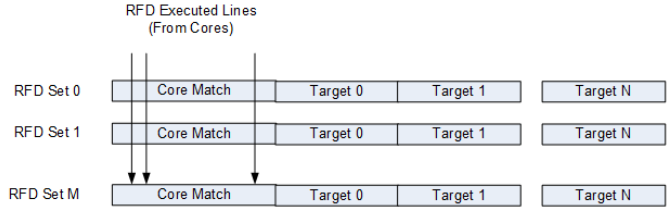


Figure 6: RFD Set Configuration

We found that two RFD sets with two targets each were sufficient for running YOLO-Tiny inference.

Core Pseudo Code for *rfd/rfds*:

```
while (TileRFDRecord[core_number] == 1) {
  // Stall
```

```
}
TileRFDRecord[core_number] = 1
if (Opcode == rfds) {
  while (TileRecord[core_number] == 1) {
    // Stall
  }
}
```

Tile Pseudo Code Code for rfd/rfds:

```
for(i=0; i<NUM_RFD_SETS; i++ ) {
  if((TileRFDRecord & PT_RFD_CFG[i].core_set) ==
      RFD_MATCH_SET_CFG[i].core_set) {
    for (j=0;j<NUM_RFD_TARGETS);j++) {
      Issue Fabric packet:RFD to
            RFD_NOTIFY_CFG[i].target[j]
    }
    TileRFDRecord = TileRFDRecord &
            ~PT_RFD_CFG[i].core_set
  }
}
```

Tile Instruction *tbarrier(count)*

The *tbarrier* instruction is used stall a tile from sending data too fast. The tbarrier instruction stalls the instruction program counter from advancing until the PUMA Tile receives Fabric RFD packets equal to the number specified in the count field of the instruction.

IV. Deep Learning Fabric

We have invented the Deep Learning Inference (DLI) protocol. The DLI protocol was optimized for sending arrays of low precision numbers for tile to tile and to and from a host interface. Unlike the fabric outlined in the PUMA architecture, the DLI protocol requires destinations to always accept packets from the fabric. This design approach eliminated fabric overheads associated with retries.

The protocol has 2 Virtual Channels (VC). Functionally only one VC is required, but the second VC allows a mechanism by which the request for data can be quickly delivered.

We built a 16-port switch using credit-based flow control, a rather narrow (32-bit wide) interfaces as the fundamental building block for our designs. The use of a 2-VC protocol was made possible by having higher level protocols designed in a way that ensures that fabric stalling is infrequent. The protocol supports large packets so high efficiencies can be supported. The large ID space (16 bits) supports many chips controlled by one host interface. Finally, by requiring only simple ordering rules, the protocol can be extended to multiple chips. Also, little effort is required to layer the protocol on top of another protocol, such as Ethernet, for server to server communication.

The fabric architecture we developed has packets for setting up weights, code, constants. It also has packets for communicating error information to the host. Finally, it has packets for pipelining inferences (see *Table 2*).

Table 2: DLI Fabric Packet Types

DLI Fabric Packet	Use
Send	Used to Send data from Tiles to Tiles and Tiles to the Bridge with PUMA. The PUMA Tile *send* initiates these packets.
RFD	Read for Data Indication. These packets are used to indicate to a sending Tile or Bridge that a Tile is ready for more data.
RFD Ack	Indicates that the barrier has been crossed, and the Tile is allows to issue another RFD packet.
Kickstart	Cause the PUMA cores and PUMA tiles to start executing PUMA instructions.
Kickstart Ack	Indication the Kickstart was received by a Tile.
Error	Used for sending error indications from the Tiles to the Bridge.
Performance Monitoring	Used for sending performance data to the Bridge.

V. Host Interface

A. Programmable Host Interface

Our host interface controller is programmable and communicates to a bridge which in turn communicates with an interconnected tile fabric (see *Figure 7*).

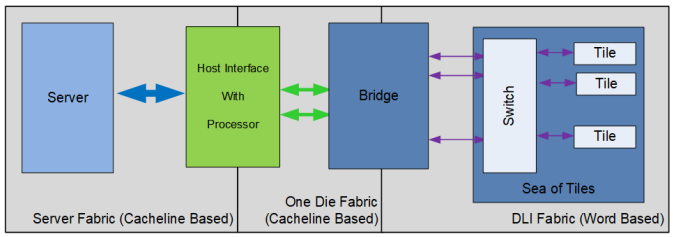


Figure 7: The host interface architecture

Our host interface controller is based on the HPE-RISC-V processor [8]. The HPE-RISC-V processor IP can be synthesized for almost any IC process or FPGA and has the advantage of utilizing open source software tools. We are writing all our HPE-RISC-V software in ANSI-C and reusing much of the code already developed. The HPE-RISC-V processor is a very low power processor, but at the same time it can move data very quickly thanks to its support for many outstanding requests. The Host controller uses a double buffered pipeline strategy, so server to accelerator transfers can be overlapped with output transfers as well as computations. The Host interface is designed with buffers large enough to hold entire video frames.

The Bridge block has hardware functionality for distributing input data to many different tiles, gathering output and performance monitoring data, and switching from processing one image to processing the next.

B. Work Queue / Completion Queue Interface

Our Host Interface is architected to use a Work Queue / Competition Queue style interface. When a device driver wants to request the DNN inference accelerator to perform a task, it writes a Work Queue Entry into the tail location of the work queue (a pre-registered section of main memory), writes a doorbell (a MMIO address associated with the work queue),

then polls on the head of the completion queue to receive the results of the work request. The Work Queue and the Completion queues are each 64-element long, enabling multiple work requests to the in-flight at the same time. With our accelerator, there are only two major types of Work Queue Entries: Load a DNN Model and Perform Inference (*Table 3*).

Table 3: Work Request Entries

Work Queue Entry	Inputs with Work Request	Completion Queue Data
Perform Inference	Input data for inference	Success or Failure and Inference Results
Load DNN Model	Core, Tile, and Bridge: Instructions and settings.	Success or Failure

Typically, a user will load a DNN Model into the accelerator, then perform inference until a new model is ready to be deployed.

C. Pipelining

The PUMA architecture does not describe how the pipelining of multiple inferences would occur. Our approach is to have a host interface with sufficiently large buffer to handle input data sample (e.g., an image) at a logical level, and have a bridge capable of sending segments of the input data to tiles processing the first layer of neural network model at the rates they can process. Once the first layer tiles processing specific segment of input data are freed-up, the corresponding segment from the subsequent input sample is sent to these tiles, while other tiles are computing deeper layers on the previous input sample.

Figure 8 shows how a multi-tile accelerator coordinates the DMA of images from the host memory, the inference, and returning results.

D. Virtualization Support

The HPE-RISC-V has an architected interface which enables the HPE-RISC-V device to look like a Gen-Z interface accelerator. From a software perspective the accelerator can be fully controlled via a Work Queue / Completion Queue Interface so that each accelerator can be shared by tens of independent containers or virtual machines.

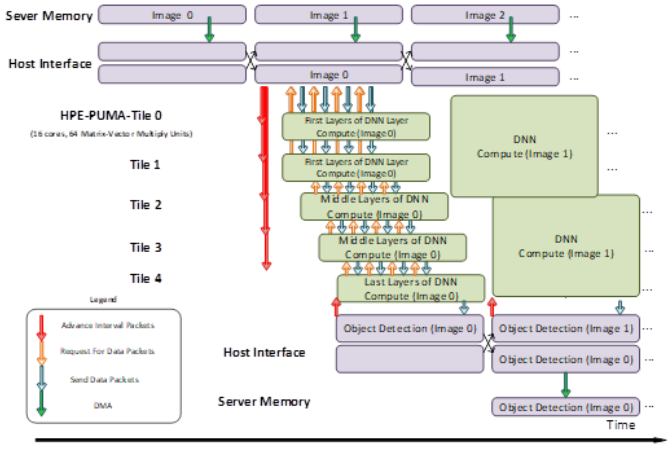


Figure 8: Pipelining of a stream of multiple inputs through the accelerators

VI. Results

A. Evaluation Setup

To evaluate our FPGA demonstrator with real-world ML applications we plugged in the FPGA card into an HPE Edgeline EL1000 server [10] (see *Figure 10*), which is used for edge/IOT based use-cases. Our FPGA demonstrator is implemented on an Advantech Vega 4000 FPGA card [11] that has a Xilinx Ultrascale+ XCVU9P FPGA. *Figure 9* shows the floorplanning of our proposed architecture on the FPGA with an annotation of different tiles and other auxiliary host interface logic such as PCIE port, and RISC-V core, etc. The proposed design occupies 78% of LUTs, 44% of BRAM, 77% of URAM, and 75% of DSPs. Due to space constraints for LUTs and URAM's on the XCVU9P device we were able to map up to 5 Tiles, 80 cores, and 320 crossbars to the FPGA.

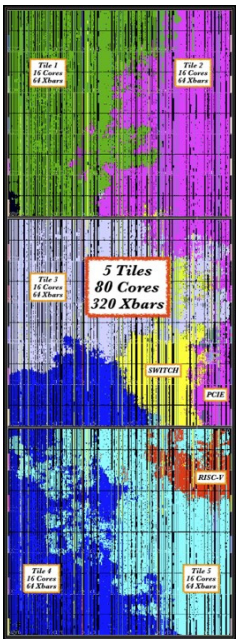


Figure 9: XCVU9P FPGA Floorplanning mapping of the PUMA design

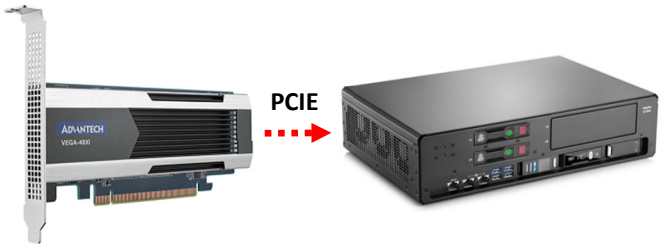


Figure 10: HPE Edgeline (EL1000) server[10] and Advantech Vega4000 FPGA card[11]

This setup focuses on full-chip architecture evaluation on real world neural network models and also serves as a platform for software development. As one of the future steps, we are planning to extend the setup to enable injection of errors to emulate noise in memristor device and analog circuits.

B. Neural Network Inference Results

We have demonstrated the capability of the PUMA architecture to handle the compute patterns of two different neural network models, each representing a major category of the deep neural networks.

For the *Recurrent Neural Network*, we have incorporated Aruba's device identification solution which employs the Gated Recurrent Unit. It detects the type and operating system of an IoT device connected to a wireless network. This model has

~180,000 trainable parameters and is able to identify 900 devices per second using total of 9 cores, at 125MHz. Although we have proven the capability of the PUMA architecture to handle recurrence with this model, we have focused our analysis more on a bigger model which is described next.

For the *Convolutional Neural Network*, we have incorporated YOLO (You Only Look Once)-Tiny version 2 network which is a real-time object detection system [9]. YOLO-Tiny has ~11 Million trainable parameters and our current FPGA demonstrator can keep up with a video stream rate of ~20 frames per second (fps) without batching, at 125 MHz clock frequency. We are planning additional microarchitecture optimizations to increase this to over 80 fps at the same clock frequency: reduce the number of clock cycles per Vector Unit single element operation from 2 to 1, enable concurrent MVM and vector operations (multi-threading), enable concurrent load from Tile data memory to all 4 MVMUs in a Core, etc. Extrapolating to 1 GHz clock frequency, an ASIC with 32 tiles will support more than 6000 fps for this model at batch size one, in silicon area less than 200 mm^2 at 22nm process. *Table 4* shows the utilization of the MVMU crossbars for the weight mapping of YOLO-Tiny on the demonstrator. In order to increase the latency and throughput of the inference, the weights of the early layers have been duplicated when mapped to crossbars, resulting the total number of mapped weights to be ~13 Million and occupying 63.1% of the total crossbar capacity. It is noteworthy that the crossbar utilization is low when the weight kernel size is small whereas the utilization is very high when the weight kernel size is bigger than the size of a single crossbar (256x256 for the current demonstrator). YOLO-Tiny has much fewer layers than other state-of-art networks and half of its layers have small weight kernel sizes, which leads to low crossbar utilization. We expect the crossbar utilization to be much higher for bigger networks.

Table 4: Crossbar utilization for YOLO-Tiny mapping

Tile	0	1	2	3	4	Total
Crossbar Utilization	15.5%	15.5%	85.6%	100%	99.0%	**63.1%**

The pictures from *Figure 11* are snapshots from a real-world use-case demonstration using our FPGA demonstrator with a surveillance camera input stream. They show the bounding boxes with classification probabilities overlaid on the input image. As expected, the post-processed results from both quantized software model and the demonstrator are very similar. Yet we are continuing to investigate the ways to close the gap. We are continuing to improve the quantization flow to consider all the hardware restrictions, which will be discussed more in detail in the following section, in order to precisely assess the hardware and software implementations and to minimize accuracy loss.

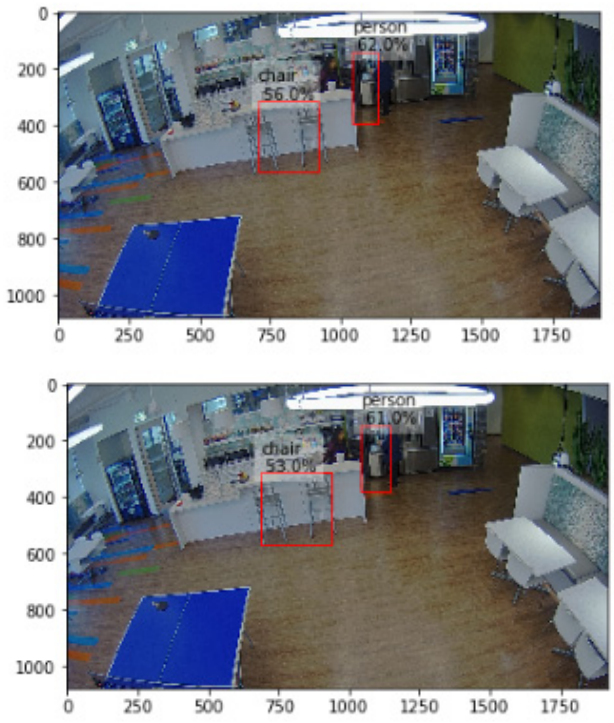


Figure 11: Sample object detection results using the quantized software model (top) and FPGA demonstrator (bottom)

Table 5 compares compute density of our MVM units against Tensor Cores implemented in state-of-the art GPU inferencing accelerator NVidia Turing T4 [10]. Extrapolating from FPGA results, a single MVM unit in an ASIC performs less operations per clock cycle than a Tensor Core, but the number of MVM units on a chip will be much larger, leading to ~3x higher performance in ~3x smaller chip area at a lower cost process node. We are not showing performance per Watt numbers here because they cannot be directly extrapolated from FPGA power consumption. However, an independent analysis projects ~10x performance per Watt advantage, thanks to in-situ weight storage (and therefore no movement of weights), efficient activation data movement, and good ADC efficiency at low resolutions.

Table 5: ASIC operations/clock cycle projections compared against a GPU

	NVidia T4 tensor core	PUMA MVMU
Max operations/clock cycle – per unit	256	128
Number of units per ASIC	320	2048
Max operations/clock cycle – per chip	81920	262144
Chip area (mm^2)	~545 (@12 nm)	~170 (@28 nm)

C. Accuracy and correctness analysis

a) Recurrent Neural Network (Gated Recurrent Unit):

We have developed a quantized software model using a mix of 8-bit and 6-bit precisions for the GRU and it retained the classification accuracy well. With the test set of *18,455* samples, the original floating-point model achieved *95.10%* classification accuracy and the quantized software model achieved *94.89%*. The FPGA demonstrator was able to exactly reproduce the quantized software model, resulting the same classification accuracy and proving the correct implementation.

b) Convolutional Neural Network (YOLO-Tiny v2):

As *Table 6* shows, YOLO-Tiny network has inferior detection rate compared to those of the state-of-art networks, and it is unfriendly to quantization. Furthermore, the PUMA architecture has additional challenges for retaining accuracy due to the split-mapping of the weight matrices, and our current quantization software does not consider such factor. For example, *Figure 12* shows the mapping of the convolutional layer 7 of YOLO-Tiny using 18 PUMA cores and 72 MVMU crossbars. In this case, 18 results from the MVMU operations need to be added horizontally to form the resulting vectors. During this process, the partial results are continuously rescaled to fit into the 16-bit variables, resulting data loss. Therefore, for the assessment of the YOLO-Tiny inference on the FPGA demonstrator, we do not focus on the accuracy, but more on the implementation correctness; we will focus more on the accuracy and quantization strategies for the benchmarks with other state-of-art networks. Also, we have determined that the average precision is not a proper metric to assess the correctness of the implementation of the demonstrator hardware and software stack. Therefore, we have instead compared the raw outputs from the quantized software model and the demonstrator hardware to determine that they are approximately equal and that our implementation is correct.

Table 6: Average Precision (AP) metrics for YOLO-Tiny inference with different quantization schemes

Model	AP	$AP_{IoU=0.50}$	$AP_{IoU=0.75}$
YOLO-Tiny original	0.9	2.6	0.3
YOLO-Tiny 8-bit quantized	0.7	2.0	0.2
YOLO-Full [8]	21.6	44.0	19.2

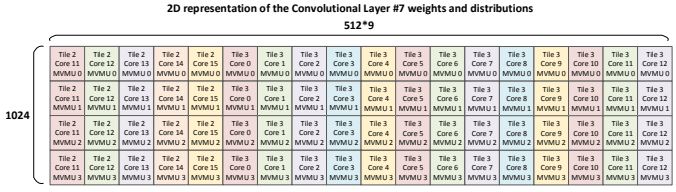


Figure 12: Weight mapping of YOLO-Tiny Convolutional Layer #7 using 18 PUMA cores (72 MVMU crossbars)

In order to justify the approximation, we have assessed two areas: 1) the percentage differences of the high activations from the last layer, 2) the distribution of all differences of activations from the last layer in the 3D space. For the *assessment #1*, we

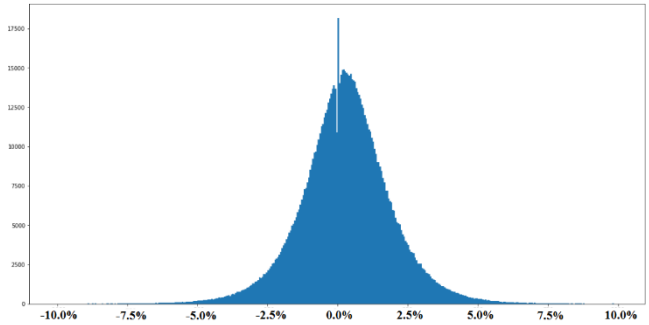


Figure 13: Histograms of the percentage differences of high activations from the last layer of YOLO-Tiny, between the quantized software model and demonstrator hardware outputs

have collected the activations of the last layer of YOLO-Tiny from the software model and the demonstrator hardware using 100 randomly-selected COCO [12] validation images. Then, we have filtered for high numbers which are bigger than the standard deviation. Finally, we have plotted the percentage differences between each pair from the quantized software and demonstrator hardware models, as shown in *Figure 13*. The percentage differences are ranged from -28.0% to 22.1%, with standard deviation of 1.66%. Again, these differences are the artifacts of the split-mapping of the weights, and the numbers and their distribution seem reasonable.

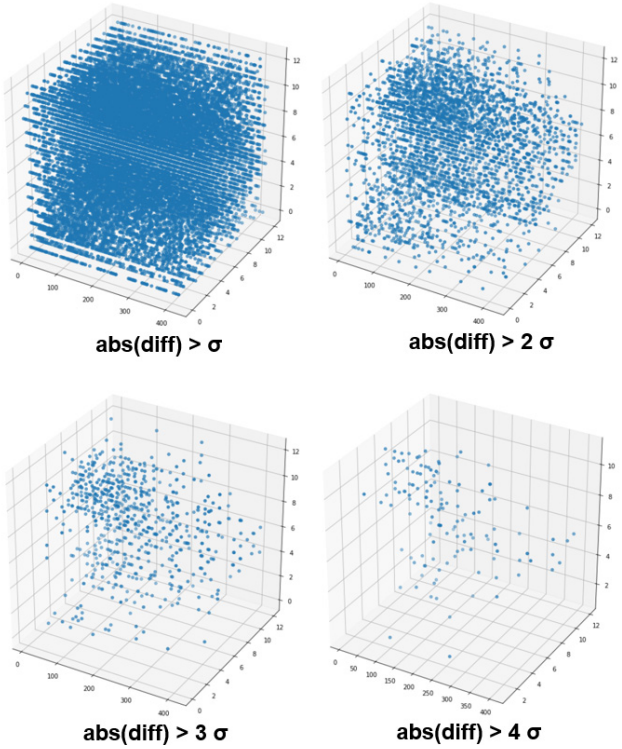


Figure 14: Distributions of the differences at different thresholds

In addition to the *assessment #1*, we have developed a software emulator of our demonstrator to run the same assembly code, and the results from the software emulator and the hardware demonstrator matched exactly, strengthening our claim for the implementation correctness.
The *assessment #2* is an interesting technique that we used to verify mainly the assembly code that runs on the demonstrator. As *Figure 14* shows, the distributions for all the activation differences between the quantized software model and the demonstrator are plotted in the 3D space with the dimensions of the output of the YOLO-Tiny network at various thresholds. We have realized that any cluster formed in a small region in this analysis indicates systematic errors in the software and/or hardware logic. With our final solution, we have not found any signature of clusters at various thresholds.
Based on the assessments described above, we have concluded that the quantized software model and the FPGA demonstrator model are approximately the same, and thus, our hardware and software implementations are systematically correct.

D. Performance monitoring instrumentation

The architecture implemented on the demonstrator includes many performance monitors embedded in the Tiles and Cores of the processing units. The Tiles are designed to automatically tag and forward all performance data to the bridge at the end of every inference interval (i.e. one iteration of the tile and core code executions). The performance counters then automatically reset and collect new data for the next inference interval. The performance monitoring feature allows a programmer to profile how often the MVMUs are in use, how often the vector units are in use, how often processors access core and tile memory, and more interestingly, how often processors wait on Tile memory count values – the PUMA approach to semaphores. *Figure 15* shows a high-level view of the performance metrics collected from the YOLO-Tiny inference on the demonstrator. Based on this information, we are improving the architecture further to enhance performance.

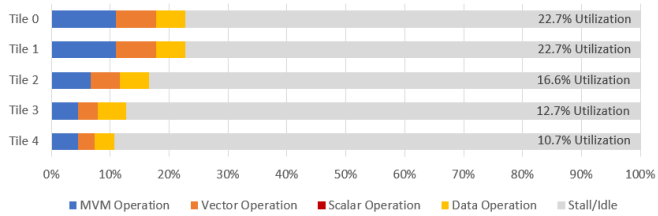


Figure 15: Utilization analysis per operation type for YOLO-Tiny inference with FPGA emulation

The performance monitoring is also available for the fabric connections for all the tiles and the bridge. *Figure 16* shows the fabric utilizations collected from the YOLO-Tiny inference on the demonstrator.

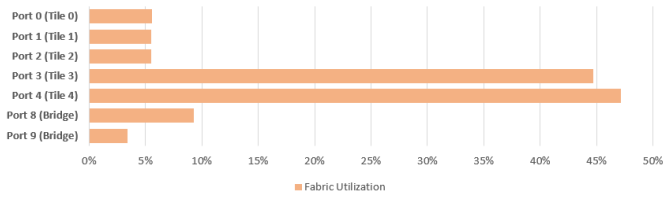


Figure 16: Fabric utilization for YOLO-Tiny inference with FPGA emulation

VII. Future Directions

A. Support for Multiple Chips

We have extended the PUMA architecture so that DNN inference can be performed using topologies of PUMA-devices connected by several high-speed links. The In-Situ weight approach enables great performance per Watt but puts a restrictive limit on the number and fidelity of the weights. By enabling multi-chip solutions, this limitation is removed.

Our approach is to build a solution which does not impact the higher-level machine learning software framework. DNN software must be written with the concept of Virtual Tiles in mind. As a part of the DNN loading step, the virtual tiles are mapped to physical tiles, enabling the hardware to know how to route Tile to Tile and Tile to bridge. As *Figure 17* shows, the

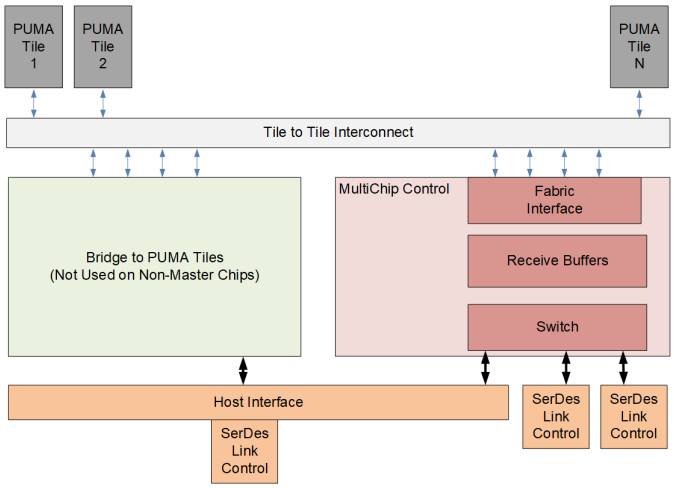


Figure 17: Multi-Chip PUMA architecture

multi-chip controller aggregates and disaggregates many narrow DLI fabric links so data can be efficiently sent over existing server interconnects like PCI Express. With this approach, still only one bridge and one host interface are needed.

B. Quantization

We are working on adaptive precision quantization (different precision for different model layers and feature maps) and quantization optimized for analog circuits.

VIII. Summary

We developed an FPGA demonstrator of the Deep Learning Inference Programmable Ultra-efficient Memristor Accelerator (PUMA [4]) that emulates analog memristor-based Matrix-

Vector Multiplication Units (MVMUs) with SRAM and enables performance projections, software development, and further microarchitecture optimizations. The demonstrator includes large number of small, independently programmable cores that enable high compute efficiency thanks to low latency data movement between MVMUs, Vector Units, Core and Tile Memory. We improved the Core-to-Core synchronization and developed a new Tile-to-Tile synchronization mechanism that enables efficient pipelining of input data stream through the accelerator. We also performed accuracy validation on one RNN and one CNN model example, and discuss future extensions enabling architecture validation on larger neural network models by model spanning across multiple FPGAs without software impact.

REFERENCES

[1] Y. Chen et. al., "DaDianNao: A machine-learning supercomputer", *In Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture 2014, p. 609-622*

[2] A. Shafiee et. al., "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars", *In Proceedings of the 43rd International Symposium on Computer Architecture 2016, p. 14-26*

[3] A. Nag et al., "Newton: Gravitating Towards the Physical Limits of Crossbar Acceleration", *arXiv 1803.06913*, 2018

[4] A. Ankit et. al., "PUMA: A Programmable Ultra-efficient Memristor-based Accelerator for Machine Learning Inference", *In Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems 2019, p. 715-731*

[5] S. Migacz, "8-bit Inference with Tensor-RT", *NVidia GPU Tech Conference 2017*

[6] R.J. Sambhav et. al., "Trained Uniform Quantization for Accurate and Efficient Neural Network Inference on Fixed-Point Hardware", *arXiv 1903.08066, 2019*

[7] RISC-V Foundation, *https://riscv.org/*

[8] J. Redmon, A. Farhadi, "YOLO9000: Better, Faster, Stronger", *ArXiv 1612.08242*

[9] NVidia Turing GPU Architecture, *https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf, page 17 and Table 5. on page 60*

[10] *HPE EL1000,* https://buy.hpe.com/us/en/servers/edgeline-systems/edgeline-systems/edgeline-converged-edge-systems/hpe-edgeline-el1000-converged-edge-system/p/1008670396

[11] *Advantech Vega4000 FPGA card,* https://www.advantech.com/products/43d339fb-73b4-4282-92bb-420aea6aa03d/vega-4000/mod_a08e9cfc-e2c0-48ee-9bdc-23b76519d733

[12] T. Lin et. al., "Microsoft COCO: Common Objects in Context", *ArXiv 1405.0312*

An Improved Implementation Approach for Quantum Phase Estimation on Quantum Computers

Hamed Mohammadbagherpoor[1], Young-Hyun Oh[2*], Patrick Dreher[3], Anand Singh[2], Xianqing Yu[2], Andy J. Rindos[2]
[1,3]Department of ECE & CSC,
North Carolina State University, Raleigh, NC 27606, USA
[2]Cloud and Cognitive Software, IBM, Durham, NC 27703, USA
Email: ohy@us.ibm.com*

***Abstract*—Quantum phase estimation (QPE) is one of the core algorithms for quantum computing. It has been extensively studied and applied in a variety of quantum applications such as the Shor's factoring algorithm, quantum sampling algorithms and the calculation of the eigenvalues of unitary matrices. The QPE algorithm has been combined with Kitaev's algorithm and the inverse quantum Fourier transform (IQFT) which are utilized as a fundamental component of such quantum algorithms. In this paper, we explore the computational challenges of implementing QPE algorithms on noisy intermediate-scale quantum (NISQ) machines using the IBM Q Experience (e.g., the IBMQX4, 5-qubit quantum computing hardware platform). Our experimental results indicate that the accuracy of finding the phase using these QPE algorithms is severely constrained by the NISQ computer's physical characteristics such as coherence time and error rates. To mitigate these physical limitations, we propose implementing a modified solution by reducing the number of controlled rotation gates and phase shift operations, thereby increasing the accuracy of the finding phase in near-term quantum computers.**

I. INTRODUCTION

The first generation of noisy intermediate-scale quantum (NISQ) [1] computers now provides a framework for re-formulating algorithms originally optimized for digital computers into a form suitable for the new quantum computing hardware platforms. These re-formulations hold the promise of potentially being able to solve particular problems exponentially faster than classical computers and also to explore regions that are inaccessible using even the most powerful digital high performance computers.

A key difference between an algorithm that is formulated for a digital computer versus a quantum computer is that digital computations are modeled on a Load-Run-Read cycle while quantum computers operate on a Prepare-Evolve-Measure cycle. The information flow for digital algorithms assumes that input data in digital bit format is inserted into the system, the program runs and then the output of the program is read. However, in quantum computers the qubit states are prepared as the input, manipulation of the input states is done using the operators and then the results are measured [2]. As part of the design for information flow is a quantum computer developer will also incorporate the quantum mechanical properties of both of superposition and entanglement of the qubits [3], [4] in order to strive for a quantum advantage over their digital counterparts.

Quantum phase estimation (QPE) is the critical building block for various quantum algorithms. In QPE the main objective of quantum phase estimation is to determine the eigenvalues of an unitary matrix with an unchanged eigenvector. This technique was described by Kitaev [5]. This procedure is a critical component in QC algorithm development for quantitative finance as well as mathematics such as Shor's algorithm for factoring the prime numbers, Grover's algorithm to search [6]–[12], cryptography, physics and quantum chemistry. Today there is an active research program to approximate, parallelize, and decompose quantum phase estimation related to algorithms [13]–[15].

However, implementing quantum algorithms on near-term quantum computers are severely constrained by low number of qubits, low reliability and high variability of quantum computers' physical characteristics. For example, the largest number factored by actual quantum computer is the number 143 which was implemented on a dipolar coupling NMR System by applying adiabatic quantum computation [16]. In addition, Shor's algorithm for factoring 15 (i.e., 3*5) on a nuclear magnetic resonance (NMR) computer is presented in [17] and the number 21 is factored by implementing qubit recycling in a photonic circuit [18]. Although the number of qubits is small, these experimental approaches will be considerably valuable when we can take full advantage of quantum supremacy in near future.

There are two main approaches that are used to implement quantum phase estimation. The first approach is to extract the phase information by applying the classical post processing computation after utilizing quantum gate operations as known as Kitaev's algorithm [19], [20]. Because Kitaev's algorithm requires some classical post processing after performing Hadamard operations, it is necessary to run a minimal number of trials to obtain the phase k^{th}-bit position with constant success probability. The second approach is to find the phase information in which the phase is estimated by applying inverse quantum Fourier transform (IQFT) [21]–[23]. However, IQFT approach requires a large number of rotation gates for precision digits to obtain more accurate phase information. Without loss of generality, more rotation gates can cause more readout errors from implementation of IQFT algorithms on near-term quantum computers. Thus, it is critical to minimize depth and controlled-rotation gates to

978-1-7281-5221-9/19/$31.00 ©2019 IEEE

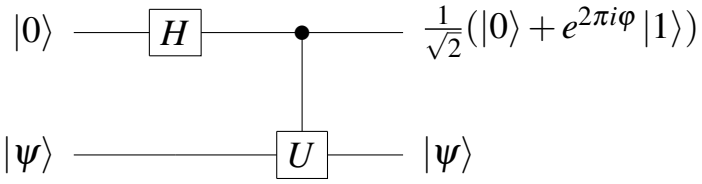


Fig. 1: Quantum circuit for transforming the states

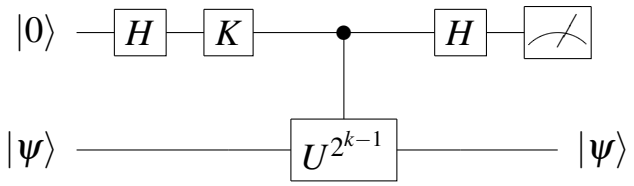


Fig. 2: Controlled U circuit

increase the accuracy of finding the phase information.

There have been several experimental hardware platforms constructed to test some of these QPE implementations. An experimental phase estimation based on quantum Fourier transform was implemented on a three-bit nuclear magnetic resonance (NMR) processor [24] but it only used to estimate the eigenvalues of one-bit Grover operators. An implementation of phase estimation algorithm on an ion-trapped quantum computer was proposed to find the eigenstates of the system [25]. Lloyd et al. have shown that quantum computers can speed up of some linear algebraic based machine learning algorithms by applying quantum phase estimation technique such as principle component analysis (PCA), support vector machine (SVM), and K-means algorithms [26], [27].

In this paper, we have implemented various quantum phase estimation algorithms using both the Qiskit Aer quantum simulator [28] for the theoretical results and the IBM Q Experience [29] for experimental results from the perspective of NISQ physical limitations. The experimental results show that the accuracy of finding the correct phase decreases as the number of qubits and quantum operations increase. To mitigate the problem, we propose modified solutions of these QPE algorithms by minimizing the number of control gates and phase shift operators.

This paper is categorized as follows. Section II describes the basic quantum operations and various phase estimation algorithms such as Kitaev's algorithm, the iterative algorithm to estimate the phase, Lloyd algorithm for phase estimation based on inverse quantum Fourier transform (IQFT), and the constant precision algorithm. In section III, the simulation and experimental results for each method are provided and compared. Finally, Section V summarizes the results and conclusions from this work.

II. Phase estimation

Phase estimation is a technique that is used to estimate the eigenvalues $|\lambda\rangle$ of a unitary matrix U with its known eigenvector $|\psi_\lambda\rangle$ [3],

$$U|\psi_\lambda\rangle = \lambda|\psi_\lambda\rangle, \tag{1}$$

where the eigenvalues of the unitary matrix are $\lambda = e^{2\pi i\varphi_n}$. The phase of the unitary matrix can be written as $\varphi_n = 0.x_1x_2x_3...x_n$ where n is the number of qubits used for phase estimation. The estimated variable $(\hat{\varphi})$ can be expressed as a binary representation,

$$\hat{\varphi} = \frac{x_1}{2^1} + \frac{x_2}{2^2} + \frac{x_3}{2^3} + \cdots + \frac{x_n}{2^n} \tag{2}$$

Fig.1 illustrates a quantum computing circuit that incorporates this problem of phase determination. The circuit consists of one qubit and an eigenstate, a Hadamard gate (H) and a rotation gate (U). The output of the circuit contains the phase $\frac{1}{\sqrt{2}}\left(|0\rangle + e^{i2\pi\varphi}|1\rangle\right)$ described in the top of the Fig.1.

The goal of phase estimation is to find the eigenvalues of an unitary matrix and then apply these eigenvalues to estimate the unknown phase of the unitary operator. However, with only this information available from the circuit output it is impossible to find the correct phase due to the superposition state on the value.

There are different methods used to calculate the phase for an unitary matrix expressed in this circuit construction. Section II-A describes sequential post processing techniques to calculate the unknown phase. Section II-B introduces an iterative technique to implement the Kiteav's algorithm with higher accuracy. Section II-C describes applying the inverse quantum Fourier transform (IQFT) to derive the unknown phase information and Section II-D discusses the arbitrary precision QPE that reduces the number of shift operators for phase estimation thereby decreasing the depth of the quantum circuits. These techniques along with their simulated and experimentally implemented results will be discussed in detail below.

A. Kitaev's algorithm

Kitaev's algorithm is the first algorithm that was introduced to estimate the phase of an unitary matrix. In this technique a set of Hadamard gates are applied to the input qubits. The outputs of the Hadamard gates connected with the controlled-$U^{2^{k-1}}$ result in an output represented by a phase shift operator. Applying a controlled-U operator k times transforms the control qubit to $\frac{1}{\sqrt{2}}\left(|0\rangle + e^{-i2\pi\varphi_k 2^{k-1}}|1\rangle\right)$. At each test phase $\varphi_k = 2^{k-1}\varphi$ can be calculated. By doing the test k times and measuring the output of each test the set of values $\varphi, 2\varphi, \cdots, 2^{k-1}\varphi$ can be achieved. These measurements are used to estimate the phase of the unitary matrix.

Fig.2 shows the circuit that performs the phase estimation. The operation K can be used to manipulate the qubit phase and provides more information about the phase of the system. Considering a $2x2$ identity matrix I_2 and setting,

$$K = I_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{3}$$

the mathematical manipulations of the qubits and the introduction of the phases can be seen in Eq. 4.

$$\begin{aligned} |0\rangle|\psi_\lambda\rangle &\xrightarrow{H\otimes I} \tfrac{1}{\sqrt{2}}(|0\rangle+|1\rangle)|\psi_\lambda\rangle \\ &\xrightarrow{C-U_k} \tfrac{1}{\sqrt{2}}(|0\rangle+|1\rangle)\otimes U_k|\psi_\lambda\rangle \\ &= \tfrac{1}{\sqrt{2}}(|0\rangle|\psi_\lambda\rangle + e^{2\pi i\varphi_k}|1\rangle|\psi_\lambda\rangle) \\ \xrightarrow{H\otimes I} & \tfrac{1}{\sqrt{2}}\frac{\left(|0\rangle+|1\rangle\right)}{\sqrt{2}}|\psi_\lambda\rangle + \frac{e^{2\pi i\varphi_k}}{\sqrt{2}}\frac{\left(|0\rangle-|1\rangle\right)}{\sqrt{2}}|\psi_\lambda\rangle \\ &= \tfrac{1}{2}\Big(\left(1+e^{2\pi i\varphi_k}\right)|0\rangle + \left(1-e^{2\pi i\varphi_k}\right)|1\rangle\Big)|\psi_\lambda\rangle \end{aligned} \tag{4}$$

Based on the calculations from Eq. 4, the probability of measuring $|0\rangle$ and $|1\rangle$ will be,

$$P(0|k) = \frac{1+cos(2\pi\varphi_k)}{2}, \qquad P(1|k) = \frac{1-cos(2\pi\varphi_k)}{2} \tag{5}$$

The quantity φ_k can be obtained more precisely by applying more trials. However, based on the data from Eq. 4 we cannot distinguish between φ_k and $-\varphi_k$. Another circuit is required to provide more information about the phase of the unitary matrix to distinguish between φ_k and $-\varphi_k$.

By considering the combination of the results from $K = I_2$ and $K = S$ the actual value of the phase can be determined.

$$K = S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \tag{6}$$

Eq. 7 illustrates that quantum circuit provides the following transformation if the $K = S$ gate is applied to the circuit.

$$\begin{aligned} |0\rangle|\psi_\lambda\rangle &\xrightarrow{H\otimes I} \tfrac{1}{\sqrt{2}}(|0\rangle+|1\rangle)|\psi_\lambda\rangle \\ &\xrightarrow{S} \tfrac{1}{\sqrt{2}}(|0\rangle+i|1\rangle)|\psi_\lambda\rangle \\ &\xrightarrow{C-U_k} \tfrac{1}{\sqrt{2}}(|0\rangle+i|1\rangle)\otimes U_k|\psi_\lambda\rangle \\ &= \tfrac{1}{\sqrt{2}}(|0\rangle|\psi_\lambda\rangle + ie^{2\pi i\varphi_k}|1\rangle|\psi_\lambda\rangle) \\ \xrightarrow{H\otimes I} & \tfrac{1}{\sqrt{2}}\frac{\left(|0\rangle+i|1\rangle\right)}{\sqrt{2}}|\psi_\lambda\rangle + i\frac{e^{2\pi i\varphi_k}}{\sqrt{2}}\frac{\left(|0\rangle-i|1\rangle\right)}{\sqrt{2}}|\psi_\lambda\rangle \\ &= \tfrac{1}{2}\Big(\left(1+ie^{2\pi i\varphi_k}\right)|0\rangle + \left(1-ie^{2\pi i\varphi_k}\right)|1\rangle\Big)|\psi_\lambda\rangle \\ &= \tfrac{1}{2}\Big(\left(1+e^{2\pi i\varphi_k+\frac{\pi}{2}}\right)|0\rangle + \left(1-ie^{2\pi i\varphi_k+\frac{\pi}{2}}\right)|1\rangle\Big)|\psi_\lambda\rangle \end{aligned} \tag{7}$$

Based on the calculations from Eq. 7, the probability of measuring $|0\rangle$ and $|1\rangle$ will be,

$$P(0|k) = \frac{1-sin(2\pi\varphi_k)}{2}, \qquad P(1|k) = \frac{1+sin(2\pi\varphi_k)}{2} \tag{8}$$

Eq.8 provides the additional information needed to determine the correct phase of the unitary matrix. In each test the probabilities of being zero or one in t trials are measured. By using the results from Eq.5 and Eq.8 the estimation of $cos(2\pi\varphi_k)$, $sin(2\pi\varphi_k)$, and the phase $(\hat{\varphi})$ can be calculated by

$$\hat{\varphi_k} = \frac{1}{2\pi}\, tan^{-1}\left(\frac{C_k}{S_k}\right) \tag{9}$$

where C_k and S_k are the estimation of $cos(2\pi\varphi_k)$ and $sin(2\pi\varphi_k)$ respectively.

In Kitaev's algorithm post processing calculation is required to estimate the value of the phase. Estimating of the phase within m bits of accuracy requires to increase the number of trials. $O\left(\frac{log(1-\delta)}{\varepsilon}\right)$ samples are required to estimate within ε with probability of $1-\delta$.

B. Iterative quantum phase estimation

In our experimental implementation, increasing the number of gates to estimate the phase with higher accuracy increases the convergence error so that the ability to approach the correct answer degrades. This section describes the iterative technique composed of the Kitaev's algorithm as a main component that estimates the phase with high accuracy in finite set of iterations. In order to estimate the phase, however, the iterative Kitaev's algorithm requires not only to run with sufficient number of shots and measurements but also to conduct post-processing calculation to determine the phase. Svore et al. introduces a fast phase estimation algorithm which considers interference across multiple qubits and asymptotically improve in runtime with less number of measurements and lower circuit width and depth [31]. Another approach has been discussed in [32] in which an adaptive algorithm based on Bayes' rule is provided to estimate the uncertainty of the phase using the experimental data. The probability distribution is updated by Bayes' rule by analyzing the previous set of experiment.

Table I shows the general iterative Kitaev's algorithm that helps to find the unknown phase of the system with m bits of accuracy. One Hadamard gate is used to perform the superposition. A controlled-U gate is then applied to the output

Iterative Phase estimation
1: for i = 1 to m do
2: Choose Controlled-U gate.
3: Perform basic Kitaev's circuit and measure the output
4: Calculate ith bit (φ_k) of the phase by applying the information from previous iteration result (φ_{k-1})
5: Update Controlled-U gate.
6: end for
7: Return the estimation of the phase.

TABLE I: Iterative quantum phase estimation

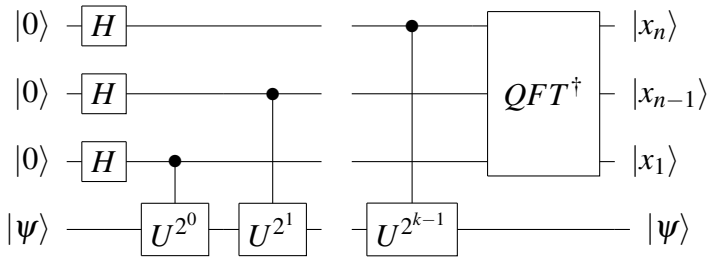


Fig. 3: Quantum phase estimation based on inverse quantum Fourier transform

of the Hadamard. The next step applies another Hadamard gate and then performs the final measurement for that iteration. In the next iteration the order of controlled-U gates is updated and the result from the previous measurement is applied to the circuit to estimate the new bit. This technique is repeated m times to estimate the phase with m bits of accuracy. In this method, each iteration of information from the previous iterations is used to estimate the next bit of the phase.

C. Phase estimation based on inverse QFT

One of the common methods used to implement the QPE algorithm is based on inverse QFT. The general view of this method has been shown in Fig.3. In this method two stages are required for phase estimation. The first stage starts with n-qubits initialized at $|0\rangle$ and prepares the state $|\psi\rangle$. The second stage uses inverse quantum Fourier transform operation to estimate the binary digits of the phase.

The mathematical equations of the first stage are given by Eq. 10.

$$\begin{aligned} &\tfrac{1}{\sqrt{2}}\big(|0\rangle + e^{2\pi i 2^{n-1}\varphi}|1\rangle\big)\tfrac{1}{\sqrt{2}}\big(|0\rangle + e^{2\pi i 2^{n-2}\varphi}|1\rangle\big)\cdots \\ &\tfrac{1}{\sqrt{2}}\big(|0\rangle + e^{2\pi i\varphi}|1\rangle\big) = \tfrac{1}{2^{n/2}}\textstyle\sum_{k=0}^{2^n-1} e^{2\pi i\frac{\varphi k}{2^n}}|k\rangle \end{aligned} \tag{10}$$

Considering $\varphi = x/2^n$ where $x = \sum_{i=0}^{n-1} 2^i x_i$ produces the Eq. 11

$$\begin{aligned} &\tfrac{1}{\sqrt{2}}\big(|0\rangle + e^{2\pi i 0.x_n}|1\rangle\big)\tfrac{1}{\sqrt{2}}\big(|0\rangle + e^{2\pi i 0.x_{n-1}x_n\varphi}|1\rangle\big)\cdots \\ &\tfrac{1}{\sqrt{2}}\big(|0\rangle + e^{2\pi i 0.x_1x_2\ldots x_n}|1\rangle\big) = \tfrac{1}{2^{n/2}}\textstyle\sum_{k=0}^{2^n-1} e^{2\pi i\frac{\varphi k}{2^n}}|k\rangle \end{aligned} \tag{11}$$

As can be seen from Fig.3 and Eq. 10 the outputs from the first stage (phase kick-back) are the input of inverse QFT. By applying controlled-$U^{2^{n-1}}$ there will phase kick back to prepare the states. Also, the output of the first stage is exactly the quantum Fourier transform of φ. By applying the inverse QFT we can recover the unknown phase. In order to analyze this method two different phase estimation circuits with different accuracy have been considered.

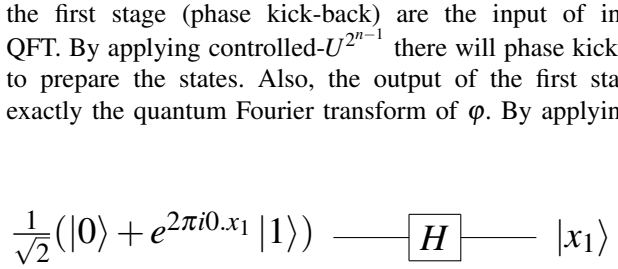


Fig. 4: One bit phase estimation quantum circuit

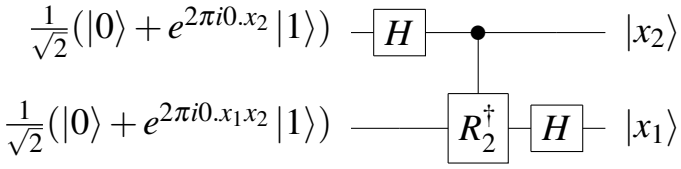


Fig. 5: 3-Qubits inverse quantum Fourier Transform (IQFT)

Case1: Starting with $\varphi = 0.x_1$, as shown in the circuit in Fig. 4 and applying Hadamard gate to the initial state $|0\rangle$ produces the Eq. 12

$$\begin{aligned} |0\rangle \xrightarrow{H} \quad & \tfrac{1}{\sqrt{2}}\big(|0\rangle + |1\rangle\big) \\ \xrightarrow{U} \quad & \tfrac{1}{\sqrt{2}}\big(|0\rangle + e^{2\pi i\varphi}|1\rangle\big) \\ \xrightarrow{H} \quad & \tfrac{1}{2}\big(1+e^{2\pi i\varphi}\big)|0\rangle + \big(1-e^{2\pi i\varphi}\big)|1\rangle \\ & = \tfrac{1}{2}\big(1+e^{2\pi i 0.x_1}\big)|0\rangle + \big(1-e^{2\pi i 0.x_1}\big)|1\rangle \end{aligned} \tag{12}$$

Calculating the probability from Eq. 12 produces Eq. 13.

$$P(|0\rangle) = \tfrac{1+cos(2\pi 0.x_1)}{2}, \qquad P(|1\rangle) = \tfrac{1-cos(2\pi 0.x_1)}{2} \tag{13}$$

Based on the result from Eq. 13, if $x_1 = 0$, then the probability of $|0\rangle$ is 1 [i.e. $P(|0\rangle) = 1$] and if $x_1 = 1$, then the probability of $|1\rangle$ is 1 [i.e. $P(|1\rangle) = 1$]. The conclusion that is inferred from this case is that the phase is considered as one bit and only one Hadamard gate is required to extract x_1.

Case2: starting with $\varphi = 0.x_1x_2$, as shown in the circuit in Fig.5 and applying inverse QFT, the unknown phase can be derived. The second digit (x_2) can be extracted by applying one Hadarmard gate, the same as the Case 1 described above. In order to extract the first digit (x_1), a controlled-rotation gate

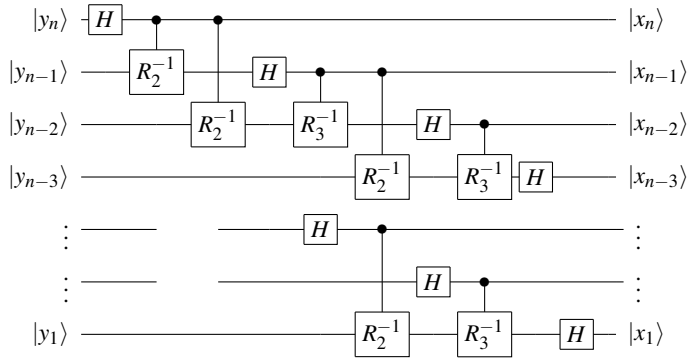


Fig. 6: QPE with arbitrary constant precision phase shift operators

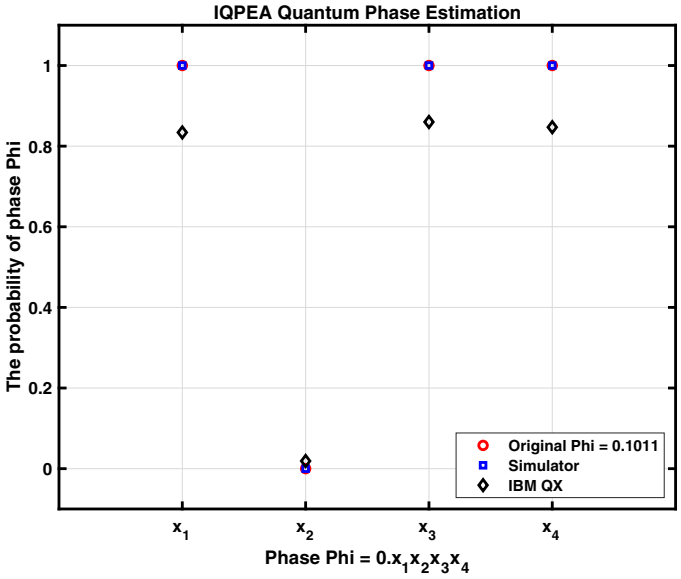


Fig. 7: Iterative quantum phase estimation algorithm on Aer Simulator and IBMQX4

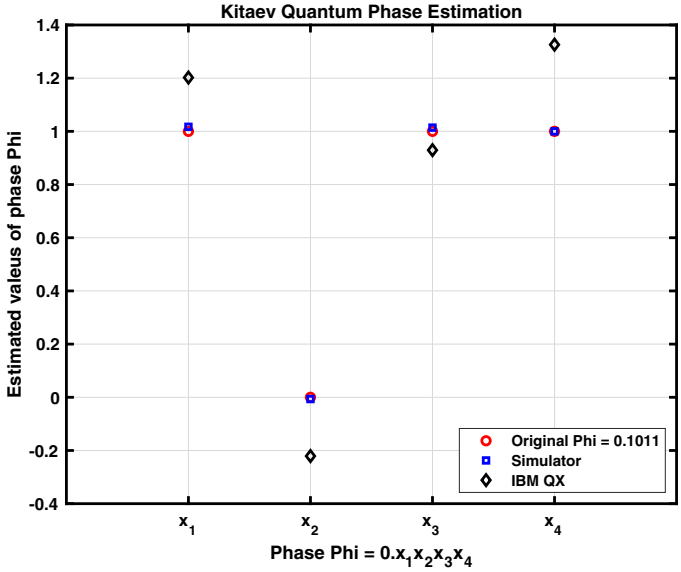


Fig. 8: Kitaev quantum phase estimation algorithm on Aer simulator and IBMQX4

R_2 is required to remove the impact of the x_2. This operation converts the result to case 1 and with the insertion of one Hadamard gate to estimate x_1, as Eq.14

$$\begin{aligned}
&\tfrac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i 0.x_1x_2}|1\rangle\right) \xrightarrow{C-R^*_2} \\
&\qquad \tfrac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i x_1 * 2^{-1} + x_2 * 2^{-2} - x_2 * 2^{-2}}|1\rangle\right) \\
&\qquad = \tfrac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i x_1 * 2^{-1}}|1\rangle\right) \\
&\xrightarrow{H} \tfrac{1}{\sqrt{2}}\left(1 + e^{2\pi i 0.x_1}\right)|0\rangle + \left(1 - e^{2\pi i 0.x_1}\right)|1\rangle
\end{aligned} \tag{14}$$

Calculating the probability from Eq. 14 we have,

$$P(|0\rangle) = \tfrac{1+cos(2\pi 0.x_1)}{2}, \qquad P(|1\rangle) = \tfrac{1-cos(2\pi 0.x_1)}{2} \tag{15}$$

The rotation gate R_2 is defined as Eq. 16 where $k = 2$.

$$R_k = \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^k} \end{pmatrix} \tag{16}$$

D. Arbitrary constant precision phase estimation

This section follows the work describing an arbitrary precision QPE [30]. This approach reduces the number of shift operators for phase estimation and as a result decreases the depth of the quantum circuit. In this approach only the information from the two previous qubits are used to estimate the phase with constant precision. Controlled phase shift rotation R_2 and R_3 are applied to extract the information about the phase with arbitrary success probability. Fig.6 illustrates the circuit diagram for this arbitrary precision QPE approach.

The first stage of this approach is similar to QPE based on QFT. By applying the controlled gate U^{2k} to the phase $\varphi = 0.x_1x_2x_3...$, the state $|\psi\rangle$ will be given in Eq. 17.

$$|\psi_k\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i * 2^k \varphi}|1\rangle\right) \tag{17}$$

$$|y_i\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i (0.x_i \ldots x_n)}|1\rangle\right) \tag{18}$$

By applying controlled rotation R_2^{-1} and R_3^{-1} to the qubits and using the information from the two previous qubits we have,

$$|\hat{\psi}_k\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i * 2^k \hat{\varphi}}|1\rangle\right) \tag{19}$$

where

$$\hat{\varphi} = 0.x_{k+1}00x_{k+4} \tag{20}$$

Applying controlled rotation R_2^{-1} and R_3^{-1} will remove the effect of x_{k+2} and x_{k+3} so, the precision in this case will be,

$$|\varphi - 0.x_{k+1}| = \theta < \frac{1}{8} \tag{21}$$

Hence,

$$|\hat{\psi}_k\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i * 2^k (0.x_{k+1} + \theta)}|1\rangle\right) \tag{22}$$

The post measurement probability based on the value of θ will be,

$$\begin{aligned}
P(0|k) &= cos^2(\pi\theta) \geq cos^2(\tfrac{\pi}{8}) \approx 0.85, \\
P(1|k) &= sin^2(\pi\theta) \leq sin^2(\tfrac{\pi}{8}) \approx 0.15
\end{aligned} \tag{23}$$

As it can be seen only controlled rotation R_2^{-1} and R_3^{-1} are used in each stage to extract the estimated phase with

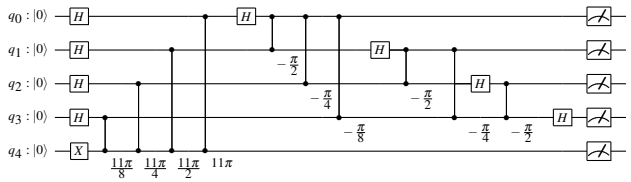


Fig. 9: Lloyd QPE algorithm gate with 1 ancillary qubit

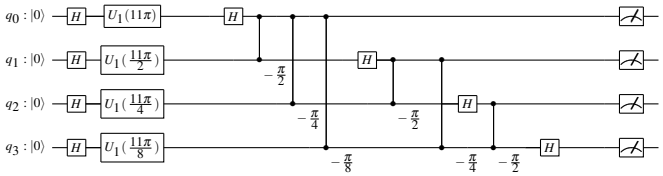


Fig. 11: Modified Lloyd QPE algorithm gate without 1 ancillary qubit

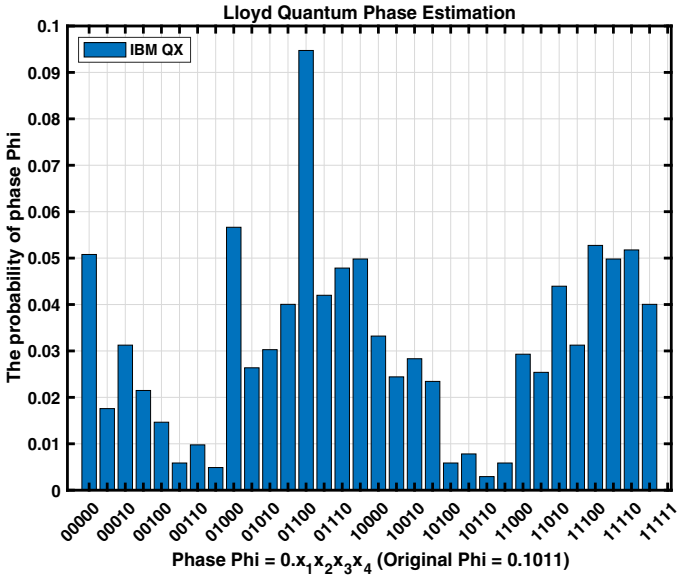


Fig. 10: Lloyd QPE algorithm on IBMQX4

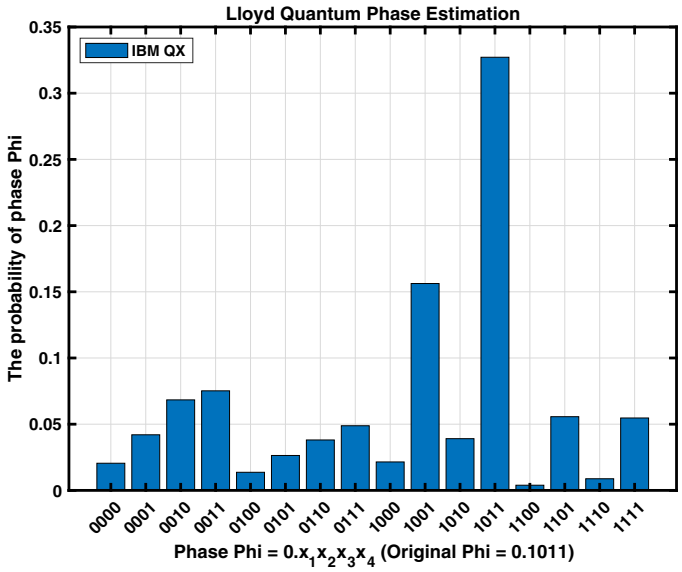


Fig. 12: Modified Lloyd QPE algorithm on IBMQX4

0.85 success probability. Applying only two controlled rotation gates will reduce the number of operating gates and as a result will decrease the depth of the circuit. This improvement will help to implement the circuit using actual quantum computers and extract the phase with higher probability and less noise which reduces the estimation probability of the correct phase.

III. Simulation and Experimental Results

In this section, we discuss the implementation of various quantum phase estimation (QPE) algorithms on both the Qiskit Aer simulator [28] and the IBMQX4, IBM Q Experience 5-qubit quantum computing hardware platform [29]. We obtained the theoretical results using the quantum simulator (Qiskit Aer) and then compared them with actual implementation on the QC hardware platform (IBMQX4). It should be noted that the actual hardware measurements include all environmental errors within the system such as readout errors, gate errors and environmental noise. The inclusion of noise models in the simulators are beyond the specific work addressed here and will be investigated in future research and is address in the Section IV.

This work examined single qubit performance. The single qubit in IBM Q Experience has good fidelity on most quantum operations but the fidelity will be quickly degrade as the number of control qubits increases. Our results confirm that the accuracy of experimental results is significantly reduced as the number of qubits increases. To mitigate the problem, the modified solutions of these QPE algorithms were implemented in order to increase the accuracy of the phase that is being experimentally measured. The experimental procedures take advantages of the capability of classical computers to store intermediate results and then feed these values into the next quantum operation when appropriate.

In our experiments, we set the phase $\varphi = 0.x_1x_2x_3x_4$, where the number of phase bit positions is 4 ($n = 4$). We defined $\varphi = 1/2+1/8+1/16$ which represents $\varphi = 0.1011$ as a binary value. For each QPE algorithm, we ran the default $1,024$ shots for both simulator and the IBMQX4.

First, we implemented Kitaev's algorithm to find the phase φ on both the Qiskit Aer simulator and IBMQX4 quantum computer. Fig.8 shows that the estimated $\hat{\varphi}$ values of simulator results are almost the same as the original φ values. The estimated φ values from the IBM hardware platform are slightly different than the original φ due to the lack of full quantum computing error correction capabilities today.

Nevertheless, we can estimate the correct binary values of bit positions by converting the estimated φ values. Because the noise can be attributed to various factors among different quantum computers, it is critical to find the hardware error rates in order to increase the accuracy of the φ estimation in Kitaev's algorithm. The accuracy can be increased by adjusting proper error rates for each quantum computer during the computation process from the estimated φ into the binary bit position.

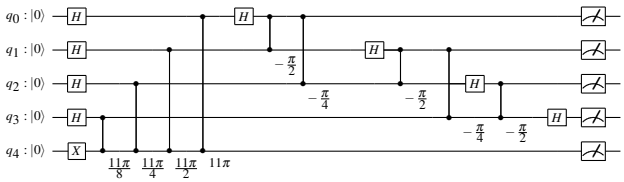


Fig. 13: Arbitrary constant precision QPE gate with 1 ancillary qubit

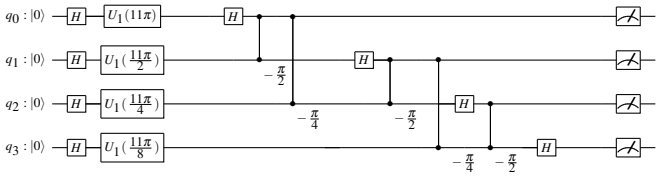


Fig. 15: Arbitrary constant precision QPE gate without 1 ancillary qubit

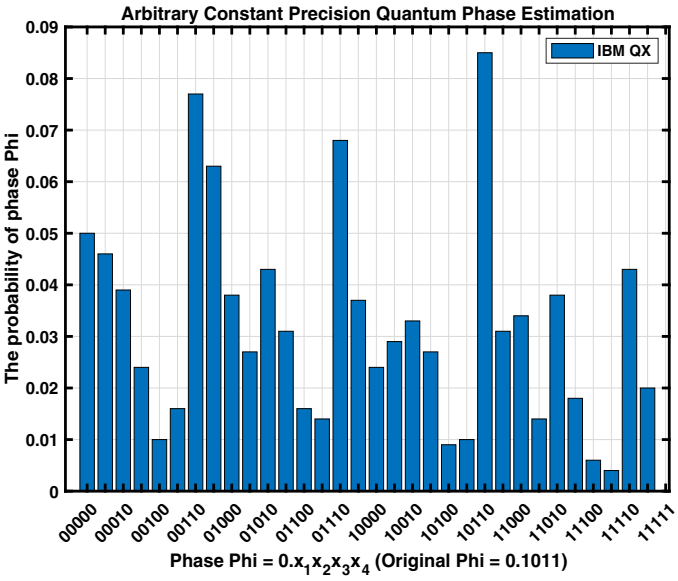


Fig. 14: Arbitrary constant precision QPE on IBMQX4

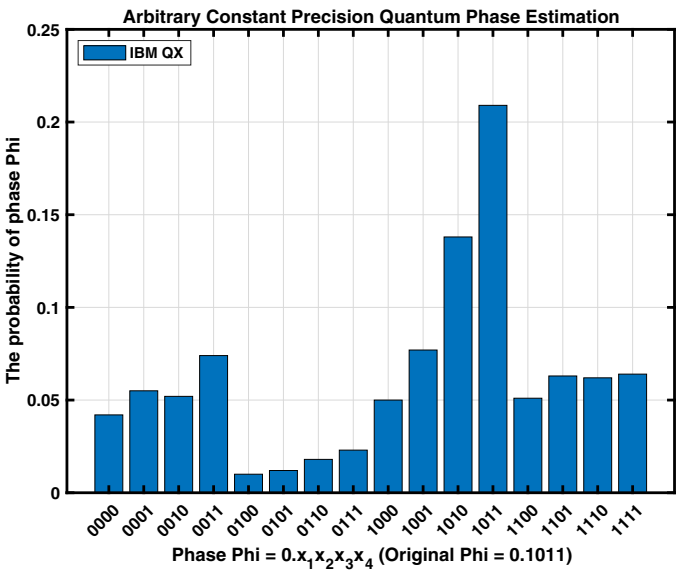


Fig. 16: Modified arbitrary constant precision QPE on IBMQX4

Second, we implemented iterative quantum phase estimation algorithm (IQPEA) to find the phase φ on both the Qiskit Aer simulator and IBMQX4 quantum computer. Fig.7 shows that the probability of finding φ value from the simulation results are exactly the same as the original φ. The experiment results are slightly different than the original φ but we can estimate the correct binary values the same way as Kitaev's algorithm.

Third, we implemented QPE algorithms using the inverse quantum Fourier transform technique to find the phase φ on both the Qiskit Aer simulator and the IBMQX4 quantum computer. Fig.10 only shows the probability of finding φ values from the IBMQX4 experiments because the probability of finding φ value from the simulation results are exactly the same as the original φ. However, the highest probability of the phase φ from the experimental results is when the φ is 0.01100 instead of the correct $\varphi = 0.1011$.

The main reasons for these inaccurate results are caused by the lack of error correction capabilities, short longitudinal, and transverse coherence time for qubits and ancillary qubits respectively. Moreover, as described in Fig.9, the number of controlled phase rotation gates on qubits can increase the readout errors.

To solve this problem and increase the accuracy of experimental results, we remove the ancillary control qubit and replace the unnecessary controlled-rotation gates with unitary rotation gates for each qubit as described in Fig.11. Our experimental results Fig.12 shows that our solution can find the correct phase φ and even the probability (i.e., 0.335%) is completely distinguished from other estimated φ values.

Finally, we implemented the arbitrary constant precision (ACP) QPE algorithm based on the inverse quantum Fourier transform technique to find the phase φ on both the Qiskit Aer simulator and the IBMQX4 quantum computer. Fig.15 only shows the probability of finding φ values from the IBMQX4 experiments because the probability of finding φ value from the simulation results are exactly the same as the original φ. However, the highest probability of the phase φ from the experimental results is when the φ is 0.10110 instead of the correct $\varphi = 0.1011$.

To increase the accuracy, the ancillary control qubit was removed and the unnecessary controlled-rotation gates were replaced with unitary rotation gates for each qubit as described in Fig.16. The experimental results shows that our solution can find the correct phase φ and even the probability (i.e., 0.209%) is completely distinguished from other estimated φ values. However, the average accuracy of i^{th} digit on ACP QPE algorithm is around 95% so that the experimental results may vary with each experimental run.

IV. Discussion

This paper investigates methods to increase the accuracy of implementing different QPE algorithms on actual quantum computers. Although the paper presents several approaches to the QPE, this work mainly focuses on addressing practical

challenges of implementing QPE based on the inverse QFT. The theoretical results of such QPE algorithms from the simulator estimate the phase with almost 100 % accuracy. However, it is not feasible to estimate the correct phase for QPE with inverse QFT from the NISQ quantum computers due to the noise of the system.

The Kitaev approach can estimate the phase of an actual system using two qubits. This approach provides a high fidelity and low error rate. The disadvantage is that post-processing is required and there must be a relatively large number of measurements performed relative to the other methods investigated here for the determinations of the phase to be measured.

The inverse QFT method does not require post processing. In addition, the binary digits of phase can be estimated separately. However, the inverse QFT method requires a large number of rotation gates to achieve a precise solution. The more gates in the system, the higher the level of noise. Higher noise levels decrease the accuracy of finding the correct phase.

The constant phase approach has the same set of advantages and disadvantages as the inverse QFT approach. However, one of the relative merits of this approach is that the number of required rotation gates of the original inverse QFT can be decreased by sacrificing some amount of the overall accuracy of the phase determination.

This paper showed that it was possible to remove the ancilla qubit from the iterative QFT without the loss of its functionality, thereby removing the unnecessary controlled-rotation gates replacing them with unitary rotation gates for each qubit. Making this change did reduce the number of controlled rotation gates required for a given level of accuracy.

All of these approaches were implemented on NISQ computers. One of the properties of these machines is that there are multiple sources of measurement errors that do occur and can be attributed to the physical system and the overall environment. For superconducting qubits coupled to readout cavities the state of the qubit is determined by measurement the response of a microwave tone incident on the readout cavity. Quantum computing hardware platforms do contain classical sources of noise that lead to readout errors of the qubit state. In addition it can also happen that not only the relaxation time T_1 decays the state of qubit during the measurement but also crosstalk between resonantors on chip and on the lines changes the probability distribution of the qubit states. These types of errors can be addressed by various techniques such as measurement calibration and error mitigation.

In this paper, the proposed approach was tested and analyzed using IBMQX4 which contains 5 qubits. The experimental results showed that using the proposed method the phase can be correctly estimated with reasonably distinct probability to other probabilities. The proposed approach can be easily applied to the large number of qubit systems to estimate the unknown phase of the complicated inputs. However, errors in the system described in this section can also increase as the number of qubits increases. Thus, it is critical that higher fidelity, longer coherence time, and lower readout errors should be followed by increasing the number of qubits to take full advantage of the presented technique.

V. CONCLUSIONS

This paper demonstrates how to implement existing quantum phase estimation (QPE) algorithms on the state-of-the-art IBM quantum computers. Our work also has documented the challenges of implementing QPE algorithms on real quantum processors.

We have proposed modified solutions of these algorithms by minimizing the number of controlled-rotation gates and by utilizing the digital computer's capabilities. Our experimental results can guide researchers to consider these challenges when they implement their quantum algorithms on noisy intermediate-scale quantum (NISQ) computers. Using these methodologies, substantial progress has been achieved applying QPE in various subject domains.

The experimental results show that our solutions significantly increase the accuracy for finding correct phase. Researchers can now implement these techniques using publicly available NISQ quantum computers such as the IBMQX4 [2] and Rigetti QPU [29] in order to take better advantage of existing NISQ machines and advance toward the longer term goals of quantum advantage and ultimately quantum supremacy.

REFERENCES

[1] John Preskill, Quantum Computing in the NISQ era and beyond, arXiv:1801.00862.

[2] Rigetti, https://www.rigetti.com/.

[3] M. A. Nielsen and I. L. Chuang, Quantum Computation and Quantum Information (Cambridge University Press, Cambridge, 2000).

[4] C. P. Williams, Explorations in Quantum Computing. Springer London, 2011.

[5] A. Kitaev, Quantum Measurements and the Abelian Stabilizer Problem. Technical Report, arXiv:quant-ph/9511026, 1995.

[6] A. A. Aspuru-Guzik, Simulated Quantum Computation of Molecular Energies, Science, vol. 309, no. 5741, pp. 17041707, Sep. 2005.

[7] B. P. Lanyon et al., Towards quantum chemistry on a quantum computer, Nature Chemistry, vol. 2, no. 2, pp. 106111, Jan. 2010.

[8] P. W. Shor, Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer, SIAM Journal on Computing, vol. 26, no. 5, pp. 14841509, Oct. 1997.

[9] K. Temme, T. J. Osborne, K. G. Vollbrecht, D. Poulin, and F. Verstraete, Quantum Metropolis sampling, Nature, vol. 471, no. 7336, pp. 8790, Mar. 2011.

[10] M. Ozols, M. Roetteler, and J. Roland, Quantum rejection sampling, in Proceedings of the 3rd Innovations in Theoretical Computer Science Conference on - ITCS 12, 2012.

[11] L. M. K. Vandersypen, M. Steffen, G. Breyta, C. S. Yannoni, M. H. Sherwood, and I. L. Chuang, Experimental realization of Shors quantum factoring algorithm using nuclear magnetic resonance, Nature, vol. 414, no. 6866, pp. 883887, Dec. 2001.

[12] A. Politi, J. C. F. Matthews, and J. L. OBrien, Shors Quantum Factoring Algorithm on a Photonic Chip, Science, vol. 325, no. 5945, pp. 12211221, Sep. 2009.

[13] M. Dobek, G. Johansson, V. Shumeiko, and G. Wendin, Arbitrary accuracy iterative quantum phase estimation algorithm using a single ancillary qubit: A two-qubit benchmark, Physical Review A, vol. 76, no. 3, Sep. 2007.

[14] J. O'Loan, Topics in Estimation of Quantum Channels, arXiv:1001.3971, 2010.

[15] V. Kliuchnikov, A. Bocharov, and K. M. Svore, Asymptotically Optimal Topological Quantum Compiling, Physical Review Letters, vol. 112, no. 14, Apr. 2014.

[16] N. Xu, J. Zhu, D. Lu, X. Zhou, X. Peng, and J. Du, Quantum Factorization of 143 on a Dipolar-Coupling Nuclear Magnetic Resonance System, Physical Review Letters, vol. 108, no. 13, Mar. 2012.

[17] L. M. K. Vandersypen, M. Steffen, G. Breyta, C. S. Yannoni, M. H. Sherwood, and I. L. Chuang, Experimental realization of Shors quantum factoring algorithm using nuclear magnetic resonance, Nature, vol. 414, no. 6866, pp. 883887, Dec. 2001.
[18] X. Peng et al., Quantum Adiabatic Algorithm for Factorization and Its Experimental Implementation, Physical Review Letters, vol. 101, no. 22, Nov. 2008.
[19] A. Y. Kitaev, Quantum computations: algorithms and error correction, Russian Mathematical Surveys, vol. 52, no. 6, pp. 11911249, Dec. 1997.
[20] A. Y. Kitaev, A. Shen, and M. Vyalyi, Classical and Quantum Computation (American Mathematical Society, Providence, Rhode Island, 2002).
[21] D. S. Abrams and S. Lloyd, Quantum Algorithm Providing Exponential Speed Increase for Finding Eigenvalues and Eigenvectors, Physical Review Letters, vol. 83, no. 24, pp. 51625165, Dec. 1999.
[22] B. T. Torosov and N. V. Vitanov, Design of quantum Fourier transforms and quantum algorithms by using circulant Hamiltonians, Physical Review A, vol. 80, no. 2, Aug. 2009.
[23] R. Cleve, A. Ekert, C. Macchiavello, and M. Mosca, Quantum algorithms revisited, Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences, vol. 454, no. 1969, pp. 339354, Jan. 1998.
[24] J S. Lee, J. Kim, Y. Cheong, and S. Lee, Implementation of phase estimation and quantum counting algorithms on an NMR quantum-information processor, Physical Review A, vol. 66, no. 4, Oct. 2002.
[25] C. Travaglione and G. J. Milburn, Generation of eigenstates using the phase-estimation algorithm, Physical Review A, vol. 63, no. 3, Feb. 2001.
[26] S. Lloyd, M. Mohseni, and P. Rebentrost, Quantum principal component analysis, Nature Physics, vol. 10, no. 9, pp. 631633, Jul. 2014.
[27] Rebentrost,Patrick and Mohseni, Masoud and Lloyd, Seth Quantum Support Vector Machine for Big Data Classification Phys. Rev. Lett., vol. 113, pp. 130503, sep. 2014.
[28] QISKIT, https://qiskit.org.
[29] IBM Q Experience, https://quantumexperience.ng.bluemix.net/qx.
[30] H. Ahmadi, C. Chiang "Quantum phase estimation with arbitrary constant-precision phase shift operators" Quantum Information and Computation, Vol. 12, Issue 9-10, pp. 864-875, September 2012.
[31] Krysta M. Svore, Matthew B.Hastings, Michael Freedman "Faster Phase Estimation" Quantum Information and Computation, Vol. 14, Issue 3-4, pp. 306-328, March 2014.
[32] Nathan Wiebe, Chris Granade "Efficient Bayesian Phase Estimation" Physical Review Letters 117, 010503, June 2016.

Optimizing the spin reversal transform on the D-Wave 2000Q

Elijah Pelofske
CCS-3 Information Sciences
Los Alamos National Laboratory
Los Alamos, NM 87545, USA
epelofske@lanl.gov

Georg Hahn
CCS-3 Information Sciences
Los Alamos National Laboratory
Los Alamos, NM 87545, USA
ghahn@cantab.net

Hristo Djidjev
CCS-3 Information Sciences
Los Alamos National Laboratory
Los Alamos, NM 87545, USA
djidjev@lanl.gov

***Abstract*—Commercial quantum annealers from D-Wave Systems make it possible to obtain approximate solutions of high quality for certain NP-hard problems in nearly constant time. Before solving a problem on D-Wave, several pre-processing methods can be applied, one of them being the so-called spin reversal or gauge transform. The spin reversal transform flips the sign of selected variables and coefficients of the Ising or QUBO (quadratic unconstrained binary optimization) representation of the problem that D-Wave minimizes. The spin reversal transform leaves the ground state of the Ising model invariant, but can average out the biases induced through analog and systematic errors on the device, thus improving the quality of the solution that D-Wave returns. This work investigates the effectiveness of the spin reversal transform for D-Wave 2000Q. We consider two important NP-hard problems, the Maximum Clique and the Minimum Vertex Cover problems, and show on a variety of input problem graphs that using the spin reversal transform can yield substantial improvements in solution quality. In contrast to the native spin reversal built into D-Wave, we consider more general ways to reverse individual spins and we investigate the dependence on the problem type, on the spin reversal probability, and possible advantages of carrying out reversals on the qubit instead of the chain level. Most importantly, for a given individual problem, we use our findings to optimize the spin reversal transform using a genetic optimization algorithm.**

***Index Terms*—D-Wave, Gauge Transform, Genetic Algorithm, Ising, Quantum Annealing, Qubit, Spin Reversal Transform**

I. Introduction

Commercial quantum computers from D-Wave Systems Inc. [3] are designed to approximately solve NP-hard optimization problems that can be expressed as the minimization of a QUBO (quadratic unconstrained binary optimization) or an Ising problem, given by

$$H(x_1,\dots,x_n)=\sum_{i=1}^{n}a_ix_i+\sum_{i<j}a_{ij}x_ix_j, \qquad (1)$$

using a process called *quantum annealing*. In (1), the coefficients $a_i \in \mathbb{R}$ are the linear weights, and $a_{ij} \in \mathbb{R}$ are the quadratic couplers defining the problem for $i,j \in \{1,\dots,n\}$. If $x_i \in \{0,1\}$, (1) is called a QUBO problem. If $x_i \in \{-1,+1\}$, (1) is called an Ising problem. Both the QUBO and Ising formulations are equivalent [4]. Many important NP-hard problems can be expressed as the minimization of a quadratic function of the form (1), see [7]. We denote the number of variables in the Ising model as n throughout the remainder of the article.

When submitting a problem of the form (1) to the D-Wave annealer, it is preprocessed in two ways. First, the qubits are arranged on the quantum annealer in a particular graph structure, called the *Chimera* graph, consisting of a lattice of bipartite cells on the physical chip [2]. However, the connectivity structure of a QUBO or Ising model, that is its nonzero couplers in (1), does not necessarily match the connectivity of the Chimera graph. In order to alleviate this issue, a *minor embedding* of the QUBO or Ising connectivity to the Chimera graph can be computed. In such an embedding, several physical qubits are identified to act as one logical qubit in (1), often severely limiting the number of available qubits. The set of physical qubits on the chip representing a logical qubit (problem variable) is called a *chain*. Second, although $a_i, a_{ij} \in \mathbb{R}$ can be arbitrary, they are rescaled to $a_i \in [-2,2]$ and $a_{ij} \in [-1,1]$ and subsequently converted to analog currents on the chip using an 8-bit digital-to-analog converter.

During the annealing, *leakage* on the physical chip from the coupler a_{ij} ($i,j \in \{1,\dots,n\}$) can alter the linear weights a_i and a_j [11]. This effect is reported to be often more serious for chained qubits. Moreover, during the process of digital-to-analog conversion, the linear weights a_i will not be perfectly mapped to currents on the chip, and thus biased to the above or below.

The so-called *spin reversal* or *gauge* transform is a simple way to alleviate this issue for Ising problems. The spin reversal is based on the observation that, although theoretically quantum annealing is invariant under a gauge transformation, the calibration of the D-Wave device is not perfect and breaks the gauge symmetry. This implies that, indeed, spin reversed Ising systems realize slightly different systems on the annealer, yielding different results which can be averaged. To apply the spin reversal, we flip the sign of an arbitrary number of variables and coefficients of the Ising problem, thus resulting in a reinterpretation of an *up* as a *down* spin and vice versa. This leaves the ground state of (1) invariant, but has the potential to reduce analog and systematic errors on the device as described earlier (by averaging them out), thus improving the quality of the solution.

978-1-7281-5221-9/19/$31.00 ©2019 IEEE

In particular, to transform the qubit x_i from -1 to $+1$, we define a new function H' with $a'_i \to -a_i$ as well as $a'_{ij} \to -a_{ij}$ and $a'_{ji} \to -a_{ji}$ for all $j \in \{1, \ldots, n\}$. We observe that the ground state energies of H and H' are identical, and that the minimum of H' is the minimum of H with the i-th variable having a flipped sign. As reported in [11], reversing too few spins leaves the Ising model almost unchanged, whereas applying the spin reversal transform to too many qubits likely results in many pairs of connected qubits being transformed, thus effectively leaving the corresponding quadratic couplers unchanged. In both cases, the spin reversal transform might only have little effect.

It is important to note that the spin reversal transform can be applied on two different levels: After embedding the Ising model to be solved onto the D-Wave architecture, the actual embedded problem that D-Wave solves is read and the spin reversal is applied to any qubit independently – this is referred to in the remainder of the article as *spin reversal on the qubit level*. Second, we can apply the spin reversal in such a way that the physical qubits in a chain (representing one logical qubit) are all either spin reversed or all left unchanged – we refer to this technique in the remainder of the article as *spin reversal on the chain level*.

The SAPI interface of D-Wave allows us to apply a built-in (simple) form of the spin reversal transform. It is controlled through the parameter *num_spin_reversal_transforms* ($= N_s$) in the function *solve_ising*. In connection with the parameter *num_reads* ($= N_r$) which specifies the total number of anneal readouts, D-Wave will generate N_s spin reversed Ising problems and obtain N_r/N_s readouts for each. The spin reserved Ising models are obtained by flipping each qubit independently with probability roughly 0.5.

In this work, we aim to assess the effectiveness of the spin reversal transform in a more general way. The main contribution of this article is twofold. First, we evaluate the performance of the spin reversal transform: In particular, we apply it to both the raw Ising formulation and the embedded problem. We are furthermore interested in its performance as a function of the probability of flipping a single qubit. Lastly, we assess its performance on two different NP-hard graph problems, the Maximum Clique problem and the Minimum Vertex Cover problem, see [7], which will be introduced later.

Second, we aim to use our findings to optimize the spin reversal transform in practice. For this we employ a genetic optimization algorithm which, for a given problem instance given as an Ising model, finds the set of qubits on which the spin reversal transform is most effective. For this we consider a separate binary indicator for each qubit (reversed or not reversed), and optimize over all n indicators to find the best configuration.

In this article, we provide a rigorous assessment of the effectiveness of the spin reversal transform, which to the best of our knowledge has not been presented in the literature previously. Existing work published in the literature does employ spin reversals, yet only as a tool to possibly enhance solutions, and only using the in-built D-Wave implementation. In [6] the authors define a new metric, the *time-to-target* metric, as the time needed by classical solvers to match the results of a quantum annealer: for their experiments, the authors employ D-Wave's native gauge transform, but it is left unclear what the contribution of the transform to the solution quality is. In [10], the author primarily evaluates several techniques to allocate weights to chains of qubits, as well as two ways of determining the final value of a chain. The effect of four spin reversals is also considered briefly, however no further statement is made on the spin reversal transform apart from the fact that it shows a slight performance gain on certain systems. In [5], the authors conclude that gauge transformations are more effective on difficult problems. However, in [1], the authors demonstrate that significant correlations exist between different gauge transforms.

The article is structured as follows. Section II describes the spin reversal transform in detail, and presents a genetic algorithm to attempt to solve the optimization over all possible spin reversal transformations. Section III presents simulation results for the Erdős-Rényi graph family, both as a function of the input graph density as well as of the probability of the spin reversal transform, and for the two aforementioned NP-hard graph problems. We also present results highlighting the dependence of the genetic optimization algorithm on its parameters, and show how the effectiveness of the spin reversal transform can be considerably increased in comparison to the D-Wave transformation over the course of only a few 'genetic' generations. The article concludes with a discussion in Section IV.

II. The Gauge Transform

This section describes the setup we employ to apply the spin reversal transform.

A. Spin reversal on the qubit level

Given an input Ising model of type (1), we embed it onto the D-Wave Chimera graph first.

Before starting the annealing process, however, we read the embedded Ising model from the D-Wave connectivity graph. The embedded problem typically consists of more variables, precisely the physical chain qubits representing the logical qubits.

Given the embedded Ising model on the D-Wave chip with n qubits, we can reverse spins in the following way. Select a set $I \subseteq \{1, \ldots, n\}$ of qubit indices to be switched. In our experiments, we will generate the set I by adding each index in $\{1, \ldots, n\}$ to it independently with a given probability p_s.

We then sequentially select one $i \in I$ at a time, and set $a'_i := -a_i$. Furthermore, we set $a'_{ij} := -a_{ij}$ and $a_{ji} := -a_{ji}$ for all $j \in \{1, \ldots, n\}$. After having applied this procedure for all $i \in I$, we use the new sets of linear weights $\{a'_i\}$ and quadratic couplers $\{a'_{ij}\}$ to form a new Ising problem H'.

The new H' is then embedded onto the D-Wave chip (instead of the originally embedded problem) and solved.

Algorithm 1: Genetic algorithm for spin reversal tuning

input : H, N, p_{spin}, p_{mat}, p_{mut}, R, N_a;
output: final population of spin reversal vectors;

1 $n \leftarrow$ number of variables in H;
2 $S \leftarrow \{s_1, \ldots, s_N : s_i \in \mathbb{B}_n^{p_{\text{spin}}}\}$;
3 **for** $r \leftarrow 1$ **to** R **do**
4 **for** $s \in S$ **do**
5 $H' \leftarrow$ reverse spins s in H;
6 Request N_a anneals for H' on D-Wave and store minimal energy among those in e_s;
7 **end**
8 $E \leftarrow \{e_s : s \in S\}$;
9 $E_0 \leftarrow$ proportion p_{mat} of lowest energies in E;
10 $S_0 \leftarrow \{s \in S : e_s \in E_0\}$;
11 $w_0 \leftarrow \arg\max\{e_s : s \in S\}$;
12 **if** $r > 1$ **then**
13 Replace w_0 in S with w_1;
14 **end**
15 $S_1 \leftarrow \emptyset$;
16 **repeat** N **times**
17 Draw two random $s_1, s_2 \in S_0$ and combine bits randomly with probability 0.5; store result in S_1;
18 **end**
19 $S_2 \leftarrow \emptyset$;
20 **for** $s \in S_1$ **do**
21 Flip each bit in s independently with probability p_{mut} and store the mutated s in S_2;
22 **end**
23 $S \leftarrow S_2$;
24 $w_1 \leftarrow \arg\min\{e_s : s \in S\}$;
25 **end**
26 **return** S;

B. Spin reversal on the chain level

Another way of applying the spin reversal is at the chain level. In contrast to Section II-A, we read the embedded problem and switch the signs of either all physical qubits in a chain that encode one logical qubit, or of none of them.

C. Optimizing the spin reversal transform

Algorithm 1 presents the genetic algorithm we use to optimize the spin reversal transform for the D-Wave annealer. Let an input Ising model H with n variables be given which we aim to solve on D-Wave. In the following, we characterize a spin reversal through a boolean vector of length n, where each *True* entry encodes that the corresponding variable is reversed.

Using Algorithm 1, we aim to find the spin reversal transform (the boolean vector) among all 2^n possible binary vectors that yields the minimal average energy on D-Wave (within a pre-specified number of N_a anneals).

We use a genetic algorithm to optimize over generations of boolean vectors (or bitstrings) of length n which are interpreted as spin reversals in the aforementioned sense. Algorithm 1 works as follows: First, in line 2, we draw N random bitstrings from $\mathbb{B}_n^{p_{\text{spin}}}$ having length n and probability p_{spin} for an entry *True*. Those are stored in a set S.

Next, the initial population in S is evaluated with regards to their quality of solution. For this, we apply the spin reversal to the variables indicated in each $s \in S$, resulting in a new Ising model H', perform N_a anneals for H' on D-Wave, and record the minimal energy e_s obtained in this way for s. All those energies are stored in a set E. Afterwards, we compute a subset E_0 of E corresponding to the proportion p_{mat} of lowest (i.e., best) energies. We store the boolean strings that correspond to the energies in E_0 in a set S_0. Additionally, we record the bitstring w_0 leading to the highest (i.e., worst) energy.

Next, for all generations after the first one, the *worst* individual w_0 is replaced by the individual w_1 that resulted in the minimal (i.e., best) energy in the previous generation (see line 24), thereby ensuring that the current global minimum solution found is never lost.

Next, we carry out a *crossover* operation on the population in line 16. For this, we take two arbitrary $s_1, s_2 \in S_0$ and combine them into a new boolean vector by choosing its entries independently from s_1 and s_2 with probability 0.5. The boolean vector obtained in this way is added to a set S_1 which was initialized as $S_1 = \emptyset$. This is repeated N times in order to leave the population size invariant in each generation.

Finally, a *mutation* step is applied in line 20. We flip the entries of each $s \in S_1$ independently with a low probability p_{mut} and store the resulting s in a set S_2 (likewise initialized with $S_2 = \emptyset$). After mutations have been applied to all $s \in S_1$, we restart the evaluation of the population with the updated set S_2. In preparation for the next iteration, in line 24, the bitstring w_1 is recorded with lowest (i.e., best) energy in population S.

We create R generations in this way.

After termination of Algorithm 1, we are left with the last population of boolean vectors which encode different spin reversals. In order to choose our best candidate for applying the spin reversal to the Ising model H, we determine their minimal energies one last time: We apply each to H (resulting in a new H' as before), record the lowest minimal energy for H' in N_a anneals on D-Wave, and return the boolean vector among the entire population yielding the minimum energy solution.

III. Experiments

In all the experiments that follow, we consistently employed D-Wave's default parameters with the exception of annealing time, which we set to 1 microsecond.

A. Evaluation of the spin reversal transform

In the following two subsections, we evaluate the spin reversal transform on both the qubit and the chain level.

The test graphs we employ are $G(|V|, p_G)$ Erdős–Rényi graphs with $|V| = 64$ vertices and edge probability $p_G \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$. The reason for using $|V| = 64$ vertices stems from the fact that this is the largest size of an arbitrary graph that can be embedded onto D-Wave 2000Q, see [8], [9].

In order to generate an Ising model to solve for each test graph, we consider two NP-hard graph problems: the Maximum Clique problem, which asks for a maximum fully-connected set of vertices of a graph, and the Minimum Vertex Cover problem, which asks for a minimum set of vertices in a graph such that each edge is incident to at least one vertex from that set. The QUBO/Ising formulations of those two problems in the form of (1) can be found in [7].

The spin reversal transform requires the index set I which specifies the qubits to be spin-reversed, see Section II-A. In the simulations of this section we generate the index sets with varying probabilities $p_s \in \{0.01, 0.05, 0.1, 0.2, \ldots, 0.8, 0.9, 0.95, 0.99\}$, that is for each element in $\{1, \ldots, n\}$ we decide with an independent Bernoulli trial of probability p_s whether it should be included in I or not (where n is the number of variables in the Ising model, see (1)).

For each value of p_s, we generate 50 graphs, apply the two Ising model formulations for the Maximum Clique and Minimum Vertex Cover problems, and solve them using the following two techniques using $N_a = 1000$ anneals on D-Wave: (a) we solve with D-Wave's spin reversal parameter using $N_s = 10$ (see Section I) – we will refer to this technique as *D-Wave's native spin reversal*; (b) with spin reversal on the qubit level; (c) with spin reversal on the chain level. For every batch of anneals, we report the best solution found as the mean among the 1% lowest energy solutions on D-Wave. This is to report a more robust measurement than merely recording the best solution found.

In this subsection, all lines in the figures are averages over 50 repetitions. For each repetition, we regenerate both the input graph, the Ising model, and its embedding on D-Wave.

1) Results for spin reversal on the qubit level: Figure 1 shows experimental results for a comparison of D-Wave's standard anneal to a general spin reversal approach of Section II-A as a function of the spin reversal probability p_s (x-axis). We use test graphs of varying density (blue to magenta) applied to the Maximum Clique (left) and Minimum Vertex Cover (right) problems. The two plots display the mean energy difference to D-Wave's standard anneal on the y-axis. We also observe that the mean energy difference between D-Wave standard annealing and D-Wave's built in spin reversal is quite small. We display error bars of one standard deviation for each measurement.

The figure shows that for both the Maximum Clique and the Minimum Vertex Cover problem, the approach of Section II-A always yields a solution of higher energy (worse quality) than the one using D-Wave's normal annealing.

The curves in Figure 1 exhibit a maximum (which happens to occur around $p = 0.8$): Such a maximum is to be expected, since both very low and very high probabilities leave an Ising problem essentially unchanged (see Section I).

2) Results for spin reversal on the chain level: Figure 2 shows results of a comparison of D-Wave's standard anneal to the spin reversal on the chain level described in Section II-B. As in Section III-A1, our approach of Section II-B is performing worse than D-Wave's standard annealing in all cases.

B. Results for our genetic algorithm

This section starts by assessing the dependence of the performance of Algorithm 1 on its parameters. For this, we first define a base set of parameters: $N = 20$, $p_{\text{spin}} = 0.1$, $p_{\text{mat}} = 0.1$, and $p_{\text{mut}} = 0.01$. When varying each of those four parameters, we keep the others fixed at their base values.

Figure 3 shows results for graphs of density 0.5 applied to the Ising model of the Maximum Clique problem. The generated graph and the embedding of the Maximum Clique Ising model onto the D-Wave architecture were kept fixed. Algorithm 1 was run over $R = 100$ generations. The spin reversal was applied at the qubit level. We assess the dependence on $N \in \{20, 50, 80\}$ (top left), $p_{\text{spin}} \in \{0.001, 0.01, 0.1\}$ (top right), $p_{\text{mat}} \in \{0.1, 0.3, 0.5\}$ (bottom left) and $p_{\text{mut}} \in \{0.001, 0.01, 0.1\}$ (bottom right). In all subplots of Figure 3, the green lines consistently indicate the smallest value of the three investigated choices for each parameter, red indicates the medium value and blue the largest one.

Figure 3 shows that, as expected, larger values of N result in more diverse populations, which increases the probability of creating population members (in our case, the population members are the bitstrings indicating which qubits need to be spin reversed) resulting in spin transforms yielding a low energy. Likewise, the spin reversal probability p_{spin} (used to create the initial population) should be rather low, although high values of p_{spin} eventually result in the same low energy solution given enough generations are created. The mutation probability p_{mut} should likewise be chosen small, since large values have a tendency to add too much noise to good solutions. The rate p_{mat} specifying the proportion of population members with lowest energies after annealing that are used in the crossover stage should similarly be small in order to only pass on the best candidates to the next generation, however if the crossover proportion is too small, diversity among the population becomes too low.

Note that in Figure 3 (top right) assessing the dependence on the spin reversal probability p_{spin}, in contrast to the other three plots the blue, red and green lines do not start in the same point in generation zero. This can be explained as follows: Only the parameters p_{spin} and N appear in the initialization in line 2 of Algorithm 1. Whereas for large enough N, the lowest energy solution of the initial population stays roughly the same, the spin reversal probability p_{spin} affects the population and its overall minimal energies as a whole in generation zero.

We observe that in all subfigures of Figure 3, our genetic algorithm converges to nearly the same minimum energy for the best parameter choice, which is to be expected for a good optimization scheme.

Based on Figure 3, we will use Algorithm 1 in the remainder of the article with parameters

$$N = 80,\ p_{\text{spin}} = 0.1,\ p_{\text{mat}} = 0.1,\ p_{\text{mut}} = 0.01. \qquad (2)$$

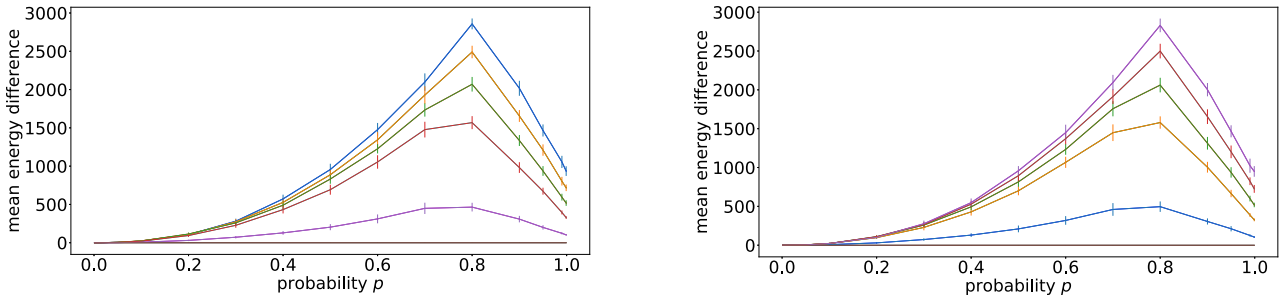


Fig. 1. Mean energy difference of the spin reversal on qubit level of Section II-A to the minimum energy found from a normal annealing schedule as a function of p_s. Test graphs generated with edge probabilities 0.1 (blue), 0.3 (yellow), 0.5 (green), 0.7 (red) and 0.9 (magenta), brown line is the D-Wave native spin reversal. Maximum Clique (left) and Minimum Vertex Cover (right) problems.

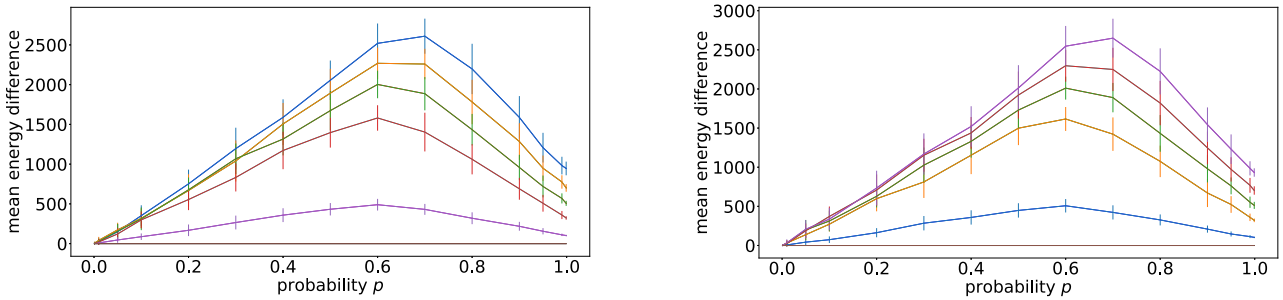


Fig. 2. Mean energy difference of the minimum energy found from spin reversal on chain level of Section II-B to the minimum energy obtained with D-Wave's standard anneal as a function of p_s. Test graphs generated with edge probabilities 0.1 (blue), 0.3 (yellow), 0.5 (green), 0.7 (red) and 0.9 (magenta), brown line is the D-Wave native spin reversal. Maximum Clique (left) and Minimum Vertex Cover (right) problems.

When using Algorithm 1 in connection with spin reversal at the chain level, see Figure 7 in Appendix A.

Using the parameters given in (2), we are able to employ Algorithm 1 to determine a (reasonably) good set of variables to be spin-reversed (with the aim of creating an equivalent Ising model which yields lower minimal energies during annealing).

Figure 4 shows simulation results for D-Wave's native spin reversal and six different realizations of Algorithm 1 over 100 generations. The blue datapoints symbolize results for the Minimum Vertex Cover problem, red stands for the Maximum Clique problem. For each datapoint of the genetic algorithm, we plot the difference of the lowest 1% mean energy (obtained when applying the current best set of variables to be spin reversed to the Ising model of the Maximum Clique problem) to the solution obtained with D-Wave's spin reversal.

We observe that for the set of parameters given in (2), after only a handful of generations (around $R > 5$), Algorithm 1 has found a combination of variables in the Ising model, which, after spin reversal has been applied to them, yield a modified (but equivalent) Ising model with (substantially) lower energies after annealing than D-Wave's spin reversal.

Figure 5 visualizes how the number of spin reversals in the best solution found by Algorithm 1 changes over the course of $R = 100$ generations. As before, blue datapoints stand for the Minimum Vertex Cover problem, and red datapoints for the Maximum Clique problem. We observe that the numbers seem to increase steadily, though the increase levels off and, as expected, the number of spin reversals in the best solution starts to plateau towards higher generation numbers.

C. Spatial correlation of spin reversed qubits on the D-Wave chip

It is of interest to investigate if there is a spatial correlation on the physical D-Wave chip between those qubits for which Algorithm 1 determines that a spin reversal improves the solution quality.

For the Maximum Clique problem on a graph of 64 vertices and graph density 0.5, we apply Algorithm 1 using the optimized parameters determined in Section III-B and $R = 100$ generations. Figure 6 shows a section of the qubit connectivity graph (the Chimera graph) of D-Wave 2000Q, where we color the single current best solution of spin reversed qubits with blue (qubit is reversed) and red (qubit is not reversed).

Figure 6 shows that at initialization (left), the assignment is random (with the pre-set probability p_{spin} of reversing a bit upon initialization). After 100 generations, the distribution of spin reversed qubits has changed: As already observed in Figure 5, the proportion of spin reversed qubits has increased.

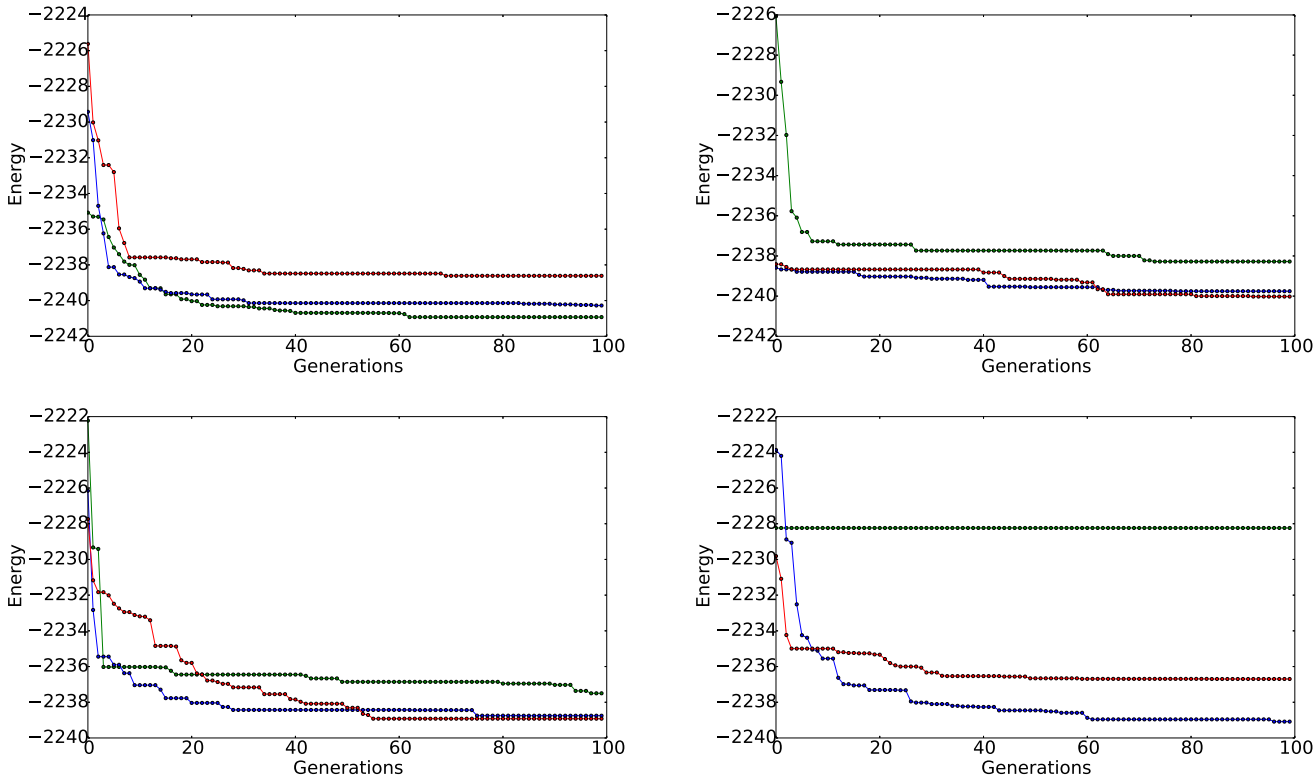


Fig. 3. Dependence of Algorithm 1 on its parameters $N \in \{20, 50, 80\}$ (top left), $p_{\text{spin}} \in \{0.001, 0.01, 0.1\}$ (top right), $p_{\text{mat}} \in \{0.1, 0.3, 0.5\}$ (bottom left) and $p_{\text{mut}} \in \{0.001, 0.01, 0.1\}$ (bottom right) when applied to spin reversal at qubit level. We consistently use red for the smallest value, blue for the medium value, and green for the largest parameter value in all subplots.

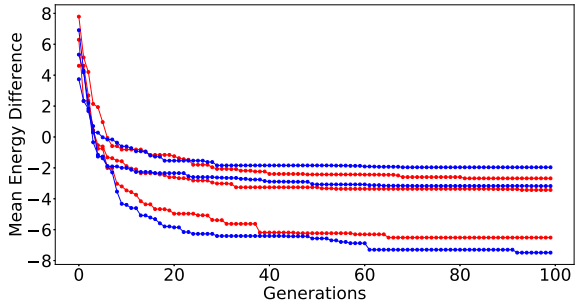


Fig. 4. Comparison of the energy difference between D-Wave's native spin reversal (N_a=10000, N_s=100) and Algorithm 1 as a function of the generation number. Maximum Clique (red) and Minimum Vertex Cover (blue) problems.

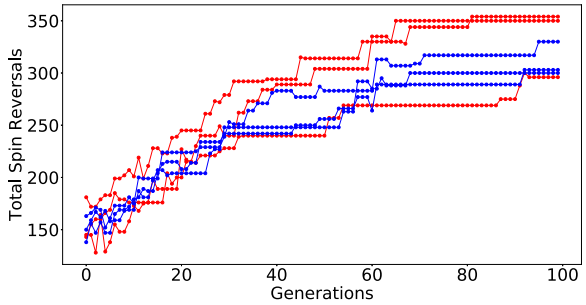


Fig. 5. Total number of observed spin reversals in Algorithm 1 as a function of the number of generations. Maximum Clique (red) and Minimum Vertex Cover (blue) problems.

IV. Conclusions

This paper investigated the spin reversal on the D-Wave 2000Q quantum annealer. In particular, we applied our own approach of a spin reversal which, in contrast to the native one built into D-Wave 2000Q, allows us to specify the probability with which spins are reversed, and we apply the spin reversal to the qubit and chain level. Importantly, we present a genetic algorithm to select the individual qubits which, if spin reversed, yield a modified Ising model that is easier to minimize (resulting in a minimum of better quality).

We summarize our findings as follows:

1) The native spin reversal on D-Wave 2000Q seems to be highly optimized for the annealer, even though D-Wave claims that its spin reversal is applied at a constant flipping probability of 0.5. In our experiments we failed to achieve lower energy solutions in the average case.
2) We show using a genetic algorithm that selecting the specific spins to be reversed is much more beneficial, and considerably improves upon D-Wave's native spin reversal. Importantly, we show that typically only a

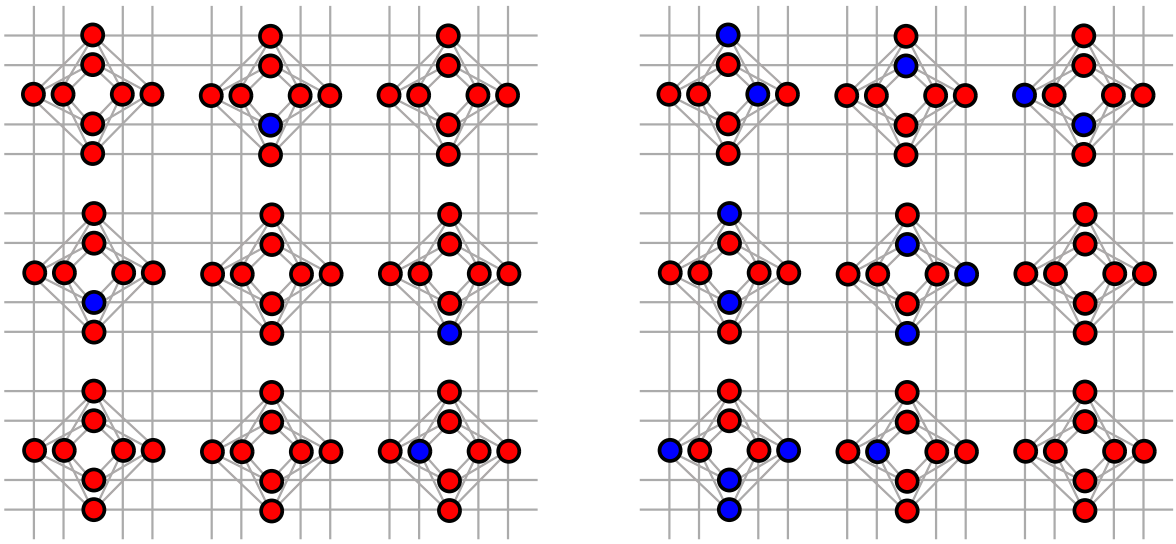


Fig. 6. Section of the qubit connectivity graph of D-Wave 2000Q. Spin reversed qubits displayed in blue and non-reversed qubits in red for the single current best solution of Algorithm 1 at initialization (left) and after 100 generations (right).

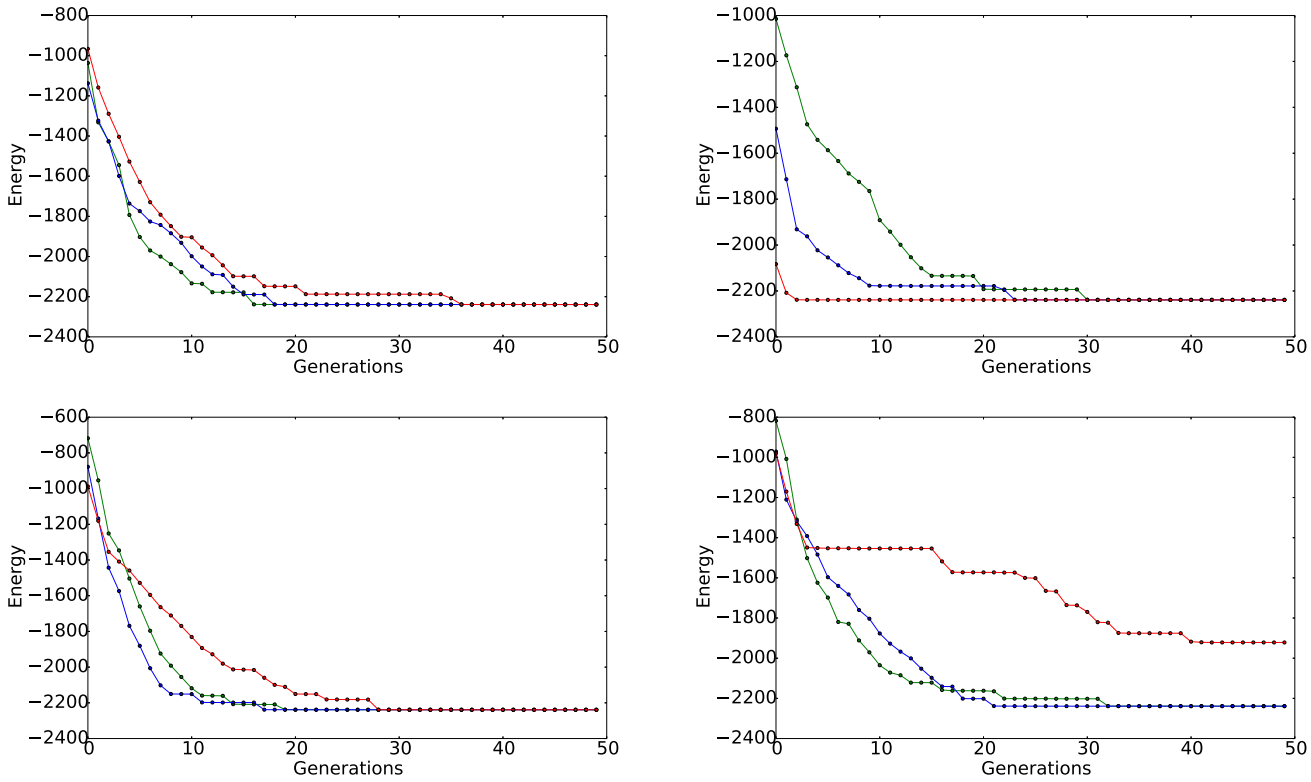


Fig. 7. Dependence of Algorithm 1 on its parameters $N \in \{20, 50, 80\}$ (top left), $p_{\text{spin}} \in \{0.1, 0.3, 0.5\}$ (top right), $p_{\text{mat}} \in \{0.1, 0.3, 0.5\}$ (bottom left) and $p_{\text{mut}} \in \{0.001, 0.01, 0.1\}$ (bottom right) when applied to spin reversal at chain level for 50 generations. We consistently use green for the smallest value, red for the medium value, and blue for the largest value in all subplots.

handful of generations (around $R = 10$) is sufficient to arrive at a combination of individual qubits which, if reversed, yield Ising models with solutions of substantially better quality. Therefore, increasing the computational effort (the number of anneals) by a constant factor (around $R = 10$ generations with a population size of $N \leq 80$ each) can yield substantially better solutions.

The genetic algorithm we employ to optimize the spin reversal transform is a rather basic version, and much more sophisticated approaches exist in the literature. Future work

includes the implementation and tuning of such advanced methods, with which we hope to (1) find even better sets of qubits to be spin reversed; and (2) decrease the required population size and number of generations, thus resulting in a reduced overhead when optimizing the spin transform.

Moreover, future research is needed to find out about the inner workings of D-Wave's native spin reversal – being able to reproduce D-Wave's native spin reversal could further improve performance of our genetic algorithm approach.

Additionally, it is always worthwhile to further investigate how the results in this article are dependent on the problem being solved.

ACKNOWLEDGMENT

This work was supported by the US Department of Energy through the Los Alamos National Laboratory. Los Alamos National Laboratory is operated by Triad National Security, LLC, for the National Nuclear Security Administration of U.S. Department of Energy (Contract No. 89233218CNA000001) and by the Laboratory Directed Research and Development program of Los Alamos National Laboratory under project numbers 20190065DR and 20180267ER.

REFERENCES

[1] Sergio Boixo, Troels F. Rønnow, Sergei V. Isakov, Zhihui Wang, David Wecker, Daniel A. Lidar, John M. Martinis, and Matthias Troyer. Evidence for quantum annealing with more than one hundred qubits. *Nature Phys*, 10:218–224, 2014.

[2] G. Chapuis, H. Djidjev, G. Hahn, and G. Rizk. Finding Maximum Cliques on the D-Wave Quantum Annealer. *Proceedings of the 2017 ACM International Conference on Computing Frontiers (CF'17)*, pages 1–8, 2017.

[3] D-Wave Systems. Quantum Computing for the Real World Today, 2000.

[4] Hristo Djidjev, Guillaume Chapuis, Georg Hahn, and Guillaume Rizk. Efficient Combinatorial Optimization Using Quantum Annealing. *LA-UR-16-27928. arXiv:1801.08653*, 2016.

[5] Andrew D. King and Catherine C. McGeoch. Algorithm engineering for a quantum annealing platform. *CoRR*, abs/1410.2628, 2014.

[6] James King, Sheir Yarkoni, Mayssam M. Nevisi, Jeremy P. Hilton, and Catherine C. McGeoch. Benchmarking a quantum annealing processor with the time-to-target metric. *arXiv:1508.05087*, pages 1–29, 2015.

[7] A. Lucas. Ising formulations of many NP problems. *Front Phys*, 2(5):1–27, 2014.

[8] Elijah Pelofske, Georg Hahn, and Hristo Djidjev. Solving large maximum clique problems on a quantum annealer. *Proceedings of the International Workshop on Quantum Technology and Optimization Problems QTOP'19*, 2019.

[9] Elijah Pelofske, Georg Hahn, and Hristo Djidjev. Solving large minimum vertex cover problems on a quantum annealer. *Proceedings of the Computing Frontiers Conference CF'19*, 2019.

[10] K. L. Pudenz. Parameter setting for quantum annealers. In *2016 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–6, 2016.

[11] D-Wave Systems. D-Wave System Documentation, 2019.

APPENDIX

A. Further assessments of Algorithm 1 on chain level

We repeat the assessments of Algorithm 1 in Section III-B for the spin reversal applied at chain level using $R = 50$ generations. As in Section III-B, we evaluate the dependence of the genetic algorithm on its parameters N, p_{spin}, p_{mat}, and p_{mut} while keeping those parameters which are not being varied at their values given in Section III-B, except for $p_{\text{spin}} = 0.5$, see section III.

We find that Algorithm 1 converges quite quickly to a low energy solution, where the total number of spin reversals is quite low (not shown).

Entangled State Preparation for Non-binary Quantum Computing

Kaitlin N. Smith and Mitchell A. Thornton
Quantum Informatics Research Group
Southern Methodist University, Dallas, Texas, USA
{knsmith, mitch}@smu.edu

***Abstract*—A common model of quantum computing is the gate model with binary basis states. Here, we consider the gate model of quantum computing with a non-binary radix resulting in more than two basis states to represent a quantum digit, or qudit. Quantum entanglement is an important phenomenon that is a critical component of quantum computation and communications algorithms. The generation and use of entanglement among radix-2 qubits is well-known and used often in quantum computing algorithms. Quantum entanglement exists in higher-radix systems as well although little is written regarding the generation of higher-radix entangled states. We provide background describing the feasibility of multiple-valued logic quantum systems and describe a new systematic method for generating maximally entangled states in quantum systems of dimension greater than two. This method is implemented in a synthesis algorithm that is described. Experimental results are included that demonstrate the transformations needed to create specific forms of maximally entangled quantum states.**

I. Introduction

We are at an exciting age for quantum computation. Noisy intermediate-scale quantum (NISQ) technology is becoming more robust and larger in scale. We are beginning to see experiments, such as those involving molecular structure and linear algebra, that demonstrate the power of quantum machines. While the development of a fault-tolerant quantum computer (QC) is still in progress, promising research causes many people to eagerly anticipate the future of quantum information science (QIS).

Current QIS technology is in an elementary and emerging form. The QC paradigm differs greatly in many aspects from classical computation. When comparing the two models, it is popularly argued that most significant difference is quantum information's ability to demonstrate quantum superposition and entanglement. In particular, quantum entanglement is an important phenomenon that is a critical component of most quantum computational and communications algorithms. The ability to experimentally demonstrate entanglement is significant because this phenomenon enables quantum computing algorithms that exhibit a computational advantage as compared to their classical counterparts. Another very important application of entanglement is that it allows for the implementation of ultra-secure quantum communications protocols. For example, entanglement is necessary for some variations of quantum key distribution (QKD) [1], [2], quantum factoring of composite prime numbers [3], quantum radar [4], quantum teleportation [5], and many other applications.

Most well-known quantum algorithms such as Shor's factoring, Grover's search, and many others depend upon and exploit the properties of entanglement in their implementation. Additionally the entire concept of many QIS systems such as teleportation, quantum communication channels, and others are based on the property of entanglement. The well-known recent Chinese experiments based upon their Micius satellite demonstrated that a quantum channel could be created between the earth and space. The Micius experiments utilized quantum entanglement generators as a key function [6]. The ability to create entangled quantum states for non-binary systems would directly enable these very well-known and accepted results in QIS to be generalized and applied to higher-radix qudit systems.

Most of the emphasis in the literature has been placed on radix $r = 2$ entanglement among qubits. Many of the fundamental characteristics used to represent information are binary in nature, such as particle spin and photon polarization. However, there are also other quantum phenomena that can be used to support information representation with a higher-radix digit set. The advantages of using higher-radix discrete systems for information representation are well-known for conventional information computation and communication systems. In conventional electronics, the use of radices greater than two allow for multiple bits of information to be transmitted and processed per conductor on an integrated circuit, increasing processing bandwidth while simultaneously decreasing on-chip routing congestion due to a decreased number of required conductors. In conventional data communications, it is very common to transmit symbols in modulation schemes such as quadrature-amplitude modulation (QAM) that allow for multiple bits to be communicated per transmitted symbol thus significantly increasing data rates. These and other improvements enjoyed in higher-radix conventional electronic systems can also be advantageous in some QIS systems; however, there is a need for common operations to be specified such as those that generate entanglement.

It is possible to create entangled states in higher-radix systems in a systematic way where qudits initialized to basis states are evolved to a state of entanglement. We present the feasibility of using multiple-valued logic (MVL) quantum systems and provide a novel contribution of a methodology for generating entanglement that we implemented in a synthesis algorithm. The prototype implementation produces entangle-

978-1-7281-5221-9/19/$31.00 ©2019 IEEE

ment generator circuits that yield maximally-entangled qudits of dimension greater than two. Thus our result can be viewed as a particular form of a quantum state generator. We use operators for creating entanglement in quantum systems of $r > 2$ that were presented in [7], [8]. This work builds on concepts from [7], [8], and the new contribution is a methodology for synthesizing the cascade of gates required to prepare entangled higher-radix states from a fixed starting basis. The techniques described in our work have been prototyped, and the algorithm outputs a description of operators required for entanglement. Experimental results are included that demonstrate the transformations needed to create algorithm-specific quantum states. In previous work, the required structure for entanglement generation was introduced [7]. Although the generalized circuit for a maximally entangled qudit pair was previously described, the choice and placement of $\mathbf{A}_{h,k}$ gates from those available to create a particular entangled state was left undefined. Such a methodology for determining the gates required for entangled state preparation and the accompanying prototype synthesis tool used to generate the quantum algorithm or circuit is a new finding in this paper.

This paper is organized as follows. Important background information that is needed to understand the contribution of this work can be found in Section II. In Section III, the viability of higher-radix quantum computation will be discussed and example realizations of this technology are given. Section IV provides a discussion of the types of operators or gates that we use for higher-radix maximally entangled state preparation. Section V discusses the methodology for developing the generator functions for entangled state preparation and examples and results from our prototype algorithm are included. Finally, Section VI provides a summary and conclusions.

II. Quantum Information Preliminaries

A. Information Representation and Manipulation

The most common unit of quantum information is the radix-2 quantum bit or "qubit." Qubits represent a linear combination of the basis states $|0_2\rangle = \begin{bmatrix}1 & 0\end{bmatrix}^{\mathrm{T}}$ and $|1_2\rangle = \begin{bmatrix}0 & 1\end{bmatrix}^{\mathrm{T}}$ where the subscripts are used to refer to the radix value r to avoid confusion when multiple values of r are of discussion. In general, a qubit is represented as

$$|\phi_2\rangle = \alpha\,|0_2\rangle + \beta\,|1_2\rangle\,. \tag{1}$$

Here, α and β are probability amplitudes with complex values, $c \in \mathbb{C}$, that take the form of $c = x + iy$ where i is the imaginary number satisfying $i^2 = -1$. The probability that $|\phi_2\rangle$ is measured as $|0_2\rangle$ is $\alpha^*\alpha = |\alpha|^2$ and the probability that $|\phi_2\rangle$ is measured as$|1_2\rangle$ is $\beta^*\beta = |\beta|^2$ as per the principle known as Born's rule in quantum mechanics.

Generalizing the qubit construct, higher-radix quantum digits of $r > 2$ are known as "qudits." A radix-r qudit is a linear combination of r basis states expressed as

$$|\phi_r\rangle = \sum_{i=0}^{r-1} a_i\,|i_r\rangle\,. \tag{2}$$

Because the probabilities of occupying any state must sum to unity, the complex-valued coefficients, $a_i \in \mathbb{C}$, satisfy

$$\sum_{i=0}^{r-1} |a_i|^2 = \sum_{i=0}^{r-1} a_i^* a_i = 1. \tag{3}$$

The mathematical model of an overall pure quantum state of a quantum algorithm or circuit is represented as a single vector formed using the individual qubit or qudit states. The vector is of dimension r^n where r is the radix and n is the number of individual qubits or qudits in the system. The quantum state vector, $|\phi_r\rangle$, is thus an element of a finite discrete Hilbert vector space also of dimension r^n denoted as $|\phi_r\rangle \in \mathbb{H}_{r^n}$. The formulation of the overall quantum state vector is accomplished by combining the quantum parallel state of individual qubits or qudits with a tensor or outer product operation. For example, the states $|\phi_r\rangle$ and $|\theta_r\rangle$ would be denoted and combined as $|\phi_r\rangle\,|\theta_r\rangle = |\phi\theta_r\rangle = |\phi_r\rangle \otimes |\theta_r\rangle$.

A particular algorithm or circuit represents a set of transformation operators that cause the quantum state to evolve in time until the state is eventually collapsed via measurement operations. To preserve Born's rule before measurement, quantum operations that preserve state must allow for a unity sum of the probability values that derive from the wavefunction's probability amplitudes. Thus, from a mathematical point of view, the quantum gates that compose an algorithm or circuit can be described as a square unitary transformation matrix, $\mathbf{U}$, of size $r^n \times r^n$. Quantum state evolution is typically specified in terms of a series or cascade of common operators that are usually in the form of one- or two-qudit operations. The one- or two-qudit operators are expanded to be in the form of unitary $r^n \times r^n$ matrices by forming the tensor product with appropriate identity matrices so that they become transformation operators over the entire quantum state vector. These individual $r^n \times r^n$ transformation matrices are applied in a serial order to the initial quantum state vector resulting in the final evolved state vector. The direct matrix product of the individual serial operations yields the overall transformation matrix for the algorithm or circuit. It should be mentioned that in the gate-model of QIS as used here, the actual implementation can be in the form of application specific hardware, or as the atomic operations in a programmable quantum computer. Thus, our methodology is equally applicable in a QIS circuit synthesis tool or as a technique to be used in a QC compiler. The evolution of a quantum state by a quantum operator is described mathematically as

$$|\phi_r(t_n)\rangle = \mathbf{U}\,|\phi_r(t_0)\rangle\,. \tag{4}$$

B. Quantum Superposition

The state of a qubit or qudit is generally specified as a linear combination of a set of r basis functions that span the discrete Hilbert space, $\mathbb{H}_{r^n}$. Each basis function is scaled by a complex-valued probability amplitude, a_i. A quantum state is said to be in a basis state when all but one of its probability amplitudes are zero-valued. Alternatively, when two or more

probability amplitudes are non-zero, the quantum state is exhibiting the characteristic known as quantum "superposition." Superposition is convenient since it allows a set of qudits to represent more than one value simultaneously. Superposition is responsible for many of the performance enhancements that quantum algorithms exhibit as compared to conventional algorithms for electronic computers since it essentially provides parallelism of information representation. A single qubit or qudit is is said to be "maximally superimposed" when its state vector can be expressed as a linear combination of all basis vectors such that the magnitude of each probability amplitude is equal to $\frac{1}{\sqrt{r}}$. Likewise, the overall quantum state of an algorithm or circuit can be maximally superimposed when each constituent qubit or qudit is maximally superimposed. In this case, the maximally superimposed overall state vector is expressed as a sum of scaled basis states where each scalar is $\frac{1}{\sqrt{r^n}}$ and the magnitude squared of the probability amplitude is $\frac{1}{r^n}$. States of partial superposition exist when some qubits or qudits in a quantum algorithm are in a basis state and others are superimposed. Furthermore, if all magnitudes of the probability amplitudes, $|a_i|$, are non-zero but also unequal, then we consider the quantum state to be in superposition, but not maximal superposition. An alternative definition of maximal superposition is in terms of measurement. A quantum state vector is maximally superimposed when a measurement would yield any of the r^n basis states with equal probability.

C. Quantum Entanglement

Entanglement is one of the most unique and significant aspects of QIS because entangled individual quantum elements interact and behave as a single system, even when they are separated by a large distance. Operations and measurements performed on one portion of an entangled group directly influence the state of the other members. With the Hilbert vector space model, it is impossible to describe any single element of an entangled set independently. As an example, the state $|\alpha\beta_2\rangle = a_0\,|00_2\rangle + a_1\,|11_2\rangle$ where $|a_i| \neq 0$, represents an entangled state comprised of the two qubits $|\alpha_2\rangle$ and $|\beta_2\rangle$. Mathematically, this state is cannot be factored and is therefore inseparable. Measurement of the first qubit, $|\alpha_2\rangle$, gives insight to the value of the second qubit, $|\beta_2\rangle$, without needing a second observation to occur. That is, if the measurement of qubit $|\alpha_2\rangle$ resulted in $|0_2\rangle$, then one would automatically know that the value of $|\beta_2\rangle$ simultaneously collapsed to $|0_2\rangle$ although it was not directly measured. This must be the case because it is impossible for $|\beta_2\rangle$ to be any value other than $|0_2\rangle$ if it is known that $|\alpha_2\rangle = |0_2\rangle$.

When the values of $|a_i|^2$ are equivalent in an entangled state, the state demonstrates maximal entanglement. The example state $|\alpha\beta_2\rangle$ would be maximally entangled if $a_0 = a_1 = \frac{1}{\sqrt{2}}$. Note this is quite different from the definition of maximal superposition since some of the probability amplitude values are zero.

III. Higher-radix Quantum Realizations

The nature of MVL systems allows for higher density transmission and computation of data because with $r > 2$, more information is stored in each fundamental unit of information [9]. Despite this advantage, classical computing is primarily implemented in binary due to the bistable nature of transistors. For example, MOSFETs can be in saturation or cutoff when they are treated as switching elements. If MVL were to be implemented using MOSFET transistors, it would be necessary to define voltage values in the active region that correspond to specific information values. Thus, an $r = 3$ ternary system would require voltages corresponding to the digits $\{0, 1, 2\}$. From a practical point of view, there would need to be voltage ranges specified, commonly characterized as "noise margins," that define how much a particular voltage can vary from a specified nominal voltage that represents a logic level. Thus, implementing higher-radix systems in conventional electronics is theoretically possible, but it is rarely done in practice because the advantages of maximizing the number of transistors per unit area that act as binary switches outweighs the advantage gained by using larger transistors that implement switching among r different voltages representing a higher-valued, non-binary radix system. Smaller transistors require smaller rail voltages to operate properly and subdividing these small rail-to-rail voltage intervals into more than two discrete ranges would result in noise margins that are impractical to implement. This is the primary reason that conventional digital electronics has continued to use binary switching models although the advantages of higher-radix information representation are well-known. However, in terms of data communication, it is common to use higher values of radices. As mentioned, QAM is a common example where radices of value 4, 8, 16, 32, and 64 are realized. Generally, these radices are powers of two to allow for simple conversion between transmitted data and the binary processors present in conventional electronic computers.

Today's quantum devices are noisy and error prone, but this does not mean that higher-radix quantum systems that implement qudits are infeasible. The issues preventing the common use of higher-radix systems for conventional computation, namely the noise margin issue, are not as parasitic in quantum computation. It remains to be seen if other issues will arise that give preference to some computational radices versus others. However, the "noise" present in today's NISQ QCs has to do with the undesired decoherence of a quantum state rather than issues akin to voltage noise margins, and it is anticipated that these decoherence rates will improve over time. Other QIS research groups have written about the advantages of implementing qudits rather than qubits in QCs that could lead to more powerful quantum computation [10]. The use of higher-radix systems for the representation and processing of information in QIS is potentially viable because of the discrete nature of certain quantum phenomena that is used to carry or represent information. Qudit implementations have been demonstrated experimentally as well as theoretically, and

they have the potential to increase in popularity as quantum technology becomes more mature. Examples of non-binary qudit-based QIP realizations based upon the photon include orbital angular momentum (OAM) [11], time-energy [12], frequency [13], time-phase [14], and location. Qudits implemented with superconducting solid-state technology such as transmon circuits have also been reported [15]. To accommodate to higher-dimensioned quantum systems, especially those that are compatible with radix-2 technology, methodologies for qudit control and readout have been developed [16], [17].

The implementation of higher-radix quantum computation would allow for the compression of data. For example, number of radix-r qudits, M, required to encode the information of N qubits is equal to

$$M = \frac{N}{log_2(r)}. \tag{5}$$

Increasing the radix of a system greatly increases computation and communication bandwidth. Although including more logic levels provides a certain level of computational advantage, an increase in dimension does increase system complexity because of the added opportunity for introducing error [18]. This presents the question of determining the best radix for quantum computation. This value will be heavily influenced by the number of clearly defined and manipulatable logic levels for a particular technology platform.

To perform meaningful QIP, it is critical to have the ability to prepare higher radix states, such as those where information is entangled, for the execution of quantum algorithms. While there is a considerable amount of past work regarding binary entanglement, there is a limited number of past references regarding the entanglement of qudits. Properties of maximum qudit entanglement have been studied in [19], [20]. Qudit entangled states have also been experimentally demonstrated [21]. However, a general methodology for the synthesis of a state preparation algorithm to yield a maximally entangled state from an arbitrary input state has not been previously reported to the best of our knowledge.

IV. Entanglement Generators

A generator function is required for quantum entangled state preparation. For binary or radix-2 quantum information, the Bell state generator is widely known and used to create entangled qubit pairs that are sometimes referred to as "EPR pairs." A Bell state generator consists of two key components: a single-qubit Hadamard gate that evolves one of the qubit pair into a state of maximal superposition, and a controlled-NOT, $\mathbf{C}_{NOT}$, gate that acts as the entangling gate for the two qubits. The typical use of a Bell state generator is to initialize the qubit pair into one of the four basis states followed by evolving them through the Hadamard and Controlled-NOT gate pair. The Bell state generator can be used as inspiration for a circuit structure that implements higher-radix entangled state preparation. Two key components of such an entanglement generator circuit are thus required to prepare entangled states for any radix. First, one of the involved qudits must be transformed into maximal superposition. Second, multi-qubit or qudit interaction is required to combine the states into an mathematically inseparable and therefore entangled form. These transformations can be accomplished with Chrestenson and controlled modulo-add operations, respectively. Entanglement generation with Chrestenson and controlled modulo-add gates is demonstrated for radix-4 quantum systems in [7] and radix-3 quantum systems in [8].

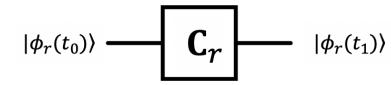


Fig. 1. Symbol of the radix-r Chrestenson gate, $\mathbf{C}_r$.

A. The Chrestenson Gate

The Hadamard operator,

$$\mathbf{H} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{6}$$

evolves a qubit that is initially in a basis state into a state of maximal superposition. With $\mathbf{H}$, a basis state qubit is transformed such that it has an equal probability of being measured as either $|0_2\rangle$ or $|1_2\rangle$. For higher-radix systems, the Chrestenson operator is the generalized version of the Hadamard operator and is applied to generate a qudit in maximal superposition. When a qudit in a basis state is applied to a radix-r Chrestenson gate, it is transformed such that it has an equal probability of observation with respect to any of its basis states.

The radix-r Chrestenson transform, $\mathbf{C}_r$, is

$$\mathbf{C}_r = \frac{1}{\sqrt{r}}\begin{bmatrix} w_0^0 & w_0^1 & \dots & w_0^{(r-1)} \\ w_1^0 & w_1^1 & \dots & w_1^{(r-1)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{(r-1)}^0 & w_{(r-1)}^1 & \dots & w_{(r-1)}^{(r-1)} \end{bmatrix} \tag{7}$$

where each element in the transformation matrix is a r^{th} root of unity raised to an integral power in the form of $w_k^j = e^{(i\frac{2\pi}{r}\times k)\times j}$ where $j, k = 0, 1, \dots, (r-1)$ [22]. The column index sets the value of j while the row index sets the value for k. The symbol for $\mathbf{C}_r$ is pictured in Fig. 1. If the radix-2 Chrestenson transform, $\mathbf{C}_2$, is derived using Eqn. 7, the Hadamard matrix results, confirming that $\mathbf{C}_r$ acts as a generalized superposition operator. More details concerning the theory of Chrestenson transforms can be found in references [22], [23] and an example implementation of a radix-4 Chrestenson gate for location-encoded photonic qudits can be found in references [24], [25].

B. The Controlled Modulo-Add Gate

The $\mathbf{NOT}$ operation, also known as the Pauli-$\mathbf{X}$ operator,

$$\mathbf{X} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \tag{8}$$

can be generalized into a modulo-r addition-by-k operator where $r = 2$ and $k = 1$. This is demonstrated by the evolution of qubit $|0_2\rangle$ to be $|((0+1)\text{mod } 2)_2\rangle = |1_2\rangle$ and $|1_2\rangle$ to be $|((1+1)\text{mod } 2)_2\rangle = |0_2\rangle$. This alternate viewpoint of the Pauli-$\mathbf{X}$ is useful in the generalization of the Bell state generator into a qudit entanglement generator for radix-r qudits. The controlled version of the Pauli-$\mathbf{X}$ or $\mathbf{NOT}$ gate is denoted as the $\mathbf{C}_{NOT}$ gate,

$$\mathbf{C}_{NOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{9}$$

With respect to modulo-r addition-by-k operators, the $\mathbf{C}_{NOT}$ or controlled-$\mathbf{X}$ gate can be referred to as a controlled-modulo-2 addition-by-1 transformation where the control value is $|1_2\rangle$. This viewpoint is directly applicable to controlled modulo-r addition-by-k operators with activation values from the set $\{0, 1, \cdots, (r-1)\}$ for the development of higher-radix maximal entanglement generators.

In the case of radix-2 systems, only two different modulo-2 additions are possible since there are two computational basis vectors, $|0_2\rangle$ and $|1_2\rangle$. Furthermore, one of these is the trivial case of modulo-2 addition-by-zero that results in the identity transformation matrix. Additionally, although most past work in binary QIS consider only the single $\mathbf{C}_{NOT}$ operator wherein the target is activated when the control is $|1_2\rangle$, another variation of a controlled-modulo-add operation could be constructed where the control logic level is $|0_2\rangle$. This variation of the $\mathbf{C}_{NOT}$ operation can also be used in a Bell state generator to prepare entangled qudit states. In general, any value from the set $\{0, 1, \cdots, (r-1)\}$ can be used as the activation or control value for a modulo-r addition-by-k gate.

Single qudit modulo-addition operations are represented by $r \times r$ transfer matrices denoted as $\mathbf{M}_k$ for transformations that cause a modulo-k addition with respect to modulus r, as used in [26]. These modulo-addition operators that cause a change of basis are also referred to in the literature as Heisenberg-Weyl operators [27]. The $\mathbf{M}_k$ matrices are all in the form of a permutation matrix and the modulo-0 addition operation is the identity function, or $\mathbf{M}_0 = \mathbf{I}_r$ where $\mathbf{I}_r$ is the $r \times r$ identity matrix. For qudit systems with radix-r, $r > 2$, there are $r-1$ different single non-trivial modulo-r additions. Because the modulo-addition operation can have a controlled form with r available control levels, there exist a total of $r(r-1) = r^2 - r$ non-trivial controlled-modulo-addition operators. The controlled-modulo-addition transformation is denoted as $\mathbf{A}_{h,k}$, where h is the control value that enables the modulo-addition by k operation to occur. $\mathbf{A}_{h,k}$ operates over two qudits of radix r, and its transfer matrix takes the form of the $r^2 \times r^2$ matrix

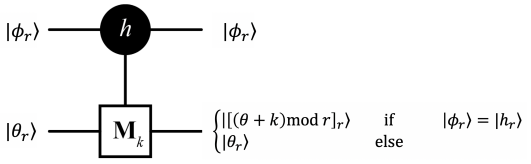


Fig. 2. Symbol of the controlled modulo-add gate, $\mathbf{A}_{h,k}$.

$$\mathbf{A}_{h,k} = \begin{bmatrix} \mathbf{D}_0 & \mathbf{0}_r & \cdots & \cdots & \cdots & \cdots & \mathbf{0}_r \\ \mathbf{0}_r & \mathbf{D}_1 & \mathbf{0}_r & \cdots & \cdots & \cdots & \mathbf{0}_r \\ \vdots & \mathbf{0}_r & \ddots & \mathbf{0}_r & \cdots & \cdots & \mathbf{0}_r \\ \vdots & \vdots & \mathbf{0}_r & \mathbf{D}_j & \mathbf{0}_r & \cdots & \mathbf{0}_r \\ \vdots & \vdots & \vdots & \mathbf{0}_r & \ddots & \mathbf{0}_r & \vdots \\ \vdots & \vdots & \vdots & \vdots & \mathbf{0}_r & \ddots & \mathbf{0}_r \\ \mathbf{0}_r & \mathbf{0}_r & \mathbf{0}_r & \mathbf{0}_r & \cdots & \mathbf{0}_r & \mathbf{D}_{(r-1)} \end{bmatrix}, \tag{10}$$

$$\text{where, } \mathbf{D}_i = \begin{cases} \mathbf{M}_0 = \mathbf{I}_r, & i \neq h \\ \mathbf{M}_k, & i = h. \end{cases}$$

In Eqn. 10, each submatrix along the diagonal is denoted as $\mathbf{D}_i$ and is of dimension $r \times r$. The $\mathbf{A}_{h,k}$ operation only allows the modulo-addition by k transformation to occur on the target whenever the control qudit is in state, $|h_r\rangle$. For a radix-2 system, the $\mathbf{A}_{1,1}$ gate derives the $\mathbf{C}_{NOT}$ transformation of Eqn. 9 when Eqn. 10 is applied. The generalized symbol of $\mathbf{A}_{h,k}$ is pictured in Fig. 2.

C. Demonstration of Higher-Radix Entanglement

A higher-radix maximal entanglement generator for two radix-r qudits takes the form of a Chrestenson gate, $\mathbf{C}_r$, on the control qudit of $r-1$ different $\mathbf{A}_{h,k}$ gates that operate after the $\mathbf{C}_r$ gate [7]. Each of the control values, h, of the $r-1$ $\mathbf{A}_{h,k}$ gates has a separate and distinct value from the set $\{0, 1, \cdots, (r-1)\}$ and each of the modulo-add-by-k target operations, k, of the $r-1$ $\mathbf{A}_{h,k}$ gates, takes on a separate and distinct value from the set $\{1, \cdots, (r-1)\}$.

An example radix-3 entanglement generator is pictured in Fig. 3. This circuit includes $\mathbf{C}_3$ calculated with Eqn. 7 to be

$$\begin{aligned} \mathbf{C}_3 &= \frac{1}{\sqrt{3}} \begin{bmatrix} w_0^0 & w_0^1 & w_0^2 \\ w_1^0 & w_1^1 & w_1^2 \\ w_2^0 & w_2^1 & w_2^2 \end{bmatrix} \\ &= \frac{1}{\sqrt{3}} \begin{bmatrix} 1 & 1 & 1 \\ 1 & e^{\frac{i2\pi}{3} \times 1} & e^{\frac{i2\pi}{3} \times 2} \\ 1 & e^{\frac{i2\pi}{3} \times 2} & e^{\frac{i2\pi}{3} \times 4} \end{bmatrix}. \end{aligned} \tag{11}$$

The $r - 1 = 3 - 1 = 2$ gates in the $\mathbf{A}_{h,k}$ cascade are derived from Eqn. 10 as

$$\mathbf{A}_{1,1} = \left[\begin{array}{ccc:ccc:ccc} 1&0&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0&0\\ \hdashline 0&0&0&0&0&1&0&0&0\\ 0&0&0&1&0&0&0&0&0\\ 0&0&0&0&1&0&0&0&0\\ \hdashline 0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&0&1 \end{array}\right] \quad (12)$$

and

$$\mathbf{A}_{2,2} = \left[\begin{array}{ccc:ccc:ccc} 1&0&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0&0\\ \hdashline 0&0&0&1&0&0&0&0&0\\ 0&0&0&0&1&0&0&0&0\\ 0&0&0&0&0&1&0&0&0\\ \hdashline 0&0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&0&1\\ 0&0&0&0&0&0&1&0&0 \end{array}\right]. \quad (13)$$

In Eqns. 12 and 13, the dashed lines are present to show the placement of the $\mathbf{M}_1$ in the center sub-matrix and $\mathbf{M}_2$ in the lower right sub-matrix, respectively. Using the transformation matrices above, the transfer function for the example maximum entanglement generator is

$$\mathbf{T}_{max} = \mathbf{A}_{(2,2)}\mathbf{A}_{(1,1)}(\mathbf{C_3} \otimes \mathbf{I_3}). \quad (14)$$

To demonstrate the preparation of a fully entangled state with the generator of Fig. 3, the value of the input state $|\phi\theta_3\rangle$ is initialized to the ground state of $|00_3\rangle$. The resulting fully entangled output is

$$\begin{aligned}\mathbf{T}_{max}\,|00_3\rangle &= \mathbf{A}_{(2,2)}\mathbf{A}_{(1,1)}(\mathbf{C_3} \otimes \mathbf{I_3})\,|00_3\rangle \\ &= \frac{1}{\sqrt{3}}\left(|00_3\rangle + |11_3\rangle + |22_3\rangle\right).\end{aligned} \quad (15)$$

In Eqn. 15, the output state is in the form of a maximally entangled state since the basis states present with non-zero probability amplitudes have magnitudes that are equal and are mathematically inseparable by factorization. Therefore, the state is entangled. As another demonstration, an entangled output can be produced with a generator circuit where $h \neq k$ in the controlled modulo-add operations. Consider the transformation matrices of

$$\mathbf{A}_{1,2} = \left[\begin{array}{ccc:ccc:ccc} 1&0&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0&0\\ \hdashline 0&0&0&0&1&0&0&0&0\\ 0&0&0&0&0&1&0&0&0\\ 0&0&0&1&0&0&0&0&0\\ \hdashline 0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&0&1 \end{array}\right] \quad (16)$$

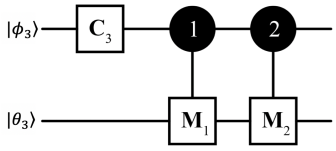


Fig. 3. Example radix-3 maximal entanglement generator.

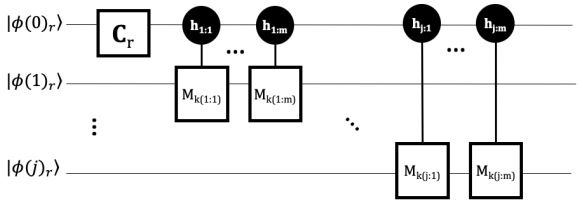


Fig. 4. Generalized structure of circuit needed for radix-r maximal entanglement among n qudits where $j = n - 1$ and $m = r - 1$.

and

$$\mathbf{A}_{2,1} = \left[\begin{array}{ccc:ccc:ccc} 1&0&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0&0\\ \hdashline 0&0&0&1&0&0&0&0&0\\ 0&0&0&0&1&0&0&0&0\\ 0&0&0&0&0&1&0&0&0\\ \hdashline 0&0&0&0&0&0&0&0&1\\ 0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&1&0 \end{array}\right]. \quad (17)$$

These $\mathbf{A}_{h,k}$ operators can be combined in order to form the maximal entanglement generator

$$\mathbf{T}_{max} = \mathbf{A}_{(2,1)}\mathbf{A}_{(1,2)}(\mathbf{C_3} \otimes \mathbf{I_3}). \quad (18)$$

If the value of the input state is fixed at $|00_3\rangle$, the transformed state becomes the fully entangled output of

$$\begin{aligned}\mathbf{T}_{max}\,|00_3\rangle &= \mathbf{A}_{(2,1)}\mathbf{A}_{(1,2)}(\mathbf{C_3} \otimes \mathbf{I_3})\,|00_3\rangle \\ &= \frac{1}{\sqrt{3}}\left(|00_3\rangle + |12_3\rangle + |21_3\rangle\right).\end{aligned} \quad (19)$$

Multiple qudits may be transformed into entangled groups if additional cascades of $\mathbf{A}_{h,k}$ operators acting on different targets are added. In these cascades, the rules for h and k values, as defined earlier, are followed. Each group is treated as an independent set where h and k values are appropriate and must not repeat. An illustration of an n qudit maximal entanglement generator structure is included in Fig. 4. In this schematic, each $\mathbf{A}_{h,k}$ operator is characterized by two index values in the form of $j_i : m_i$ where j_i indicates the qudit that acts as the target and m_i is the operator's index within the cascade. The generator of Fig. 4 could be considered a higher-radix generalization of the circuitry needed for the preparation of GHZ states whenever the number of involved qudits, n, is greater than 2.

Data: Radix (r), qudits, input state ($input$), and desired entangled state ($basis$)
Result: Gate sequence, G_i, and associated controls, c_i, and targets, t_i
Set $G \leftarrow [\,]$, $c \leftarrow [\,]$, $t \leftarrow [\,]$;
G.append(Chrestenson($radix$));
c.append(**NULL**);
t.append(0);
for $i : 1 \leq i < qudits$ **do**
  for *item* **in** *basis* **do**
    $h \leftarrow item[0]$;
    $k \leftarrow item[i] + (r - input[i])\%r$;
    if $k \neq 0$ **then**
      G.append(A(h, k));
      c.append(0);
      t.append(i);
  end
end
Algorithm 1: Find entangled state generator circuit

V. Non-binary Entangled State Preparation

It is common knowledge that the Bell state generator can be implemented for radix-2 entangled state preparation. As the dimension of a quantum system grows in size, however, it becomes less intuitive what set of operators are required to prepare a entangled set of basis states where $r > 2$. In addition, it may be necessary to generate a specific entangled state using qudits that are initialized to fixed basis values. For example, many quantum implementations initialize quantum information into a ground state, $|000\ldots00_r\rangle$, that must be used as input to the state preparation generator circuit. For this reason, it is desirable to develop a synthesis technique that derives the gate cascade required to evolve an arbitrary quantum state into a targeted entangled state.

In this work, we outline a methodology for determining the circuit structure needed to create a maximally-entangled, radix-r state with a desired set of basis states from a particular input of qudits. For radix-r, up to r qudits may become maximally entangled so that each qubit index in each linearly-combined basis has a unique value ranging from $[0 : r-1]$. For this reason, the state preparation circuits generated by this tool can include between two and r qudits. The entangled state generator algorithm has been developed in Python as a prototype synthesis tool that outputs the necessary cascade of Chrestenson and controlled-modulo-add gates to act as the generator for the desired entangled quantum state distribution. The pseudocode that finds the required generator circuit to transform a fixed input basis state into an entangled state is included in Algorithm 1.

Finding the entangled state generator circuit requires the input parameters of radix, number of qudits, input state, and desired entangled basis states. The input basis state is in the form of a list while the desired entangled state is a list of sublists that each represent a basis state in the linear combination. For instance, an input of $|\phi\theta_r\rangle = |00_3\rangle$ is passed as $input = [0, 0]$ and an entangled state of $|\phi\theta_r\rangle = \frac{1}{\sqrt{3}}(|00_3\rangle + |11_3\rangle + |22_3\rangle)$ would be passed as $basis = [[0, 0], [1, 1], [2, 2]]$. It should be noted that the probability amplitude for each of the entangled basis values is set by the column of the $\mathbf{C}_r$ matrix with an index equal to the control level in the input qudit state. Therefore, each basis will be multiplied by a radix-r root of unity along with the scale factor of $\frac{1}{\sqrt{r}}$. For example, if the output entangled state in Eqn. 15 is examined where the superimposed control qudit is $|0_3\rangle$, it is clear that each of the basis states has a probability amplitude equal to an element from the 0^{th} index column of the $\mathbf{C}_3$ transform.

Following the structure pictured in Fig. 4, if n total qudits are to be entangled, the first qubit, indexed as $|\phi(0)_r\rangle$ will always be the superimposed qubit that acts as the control for the $\mathbf{A}_{h,k}$ gates. Therefore, the $\mathbf{C}_r$ operator will act on this qubit before any other operators. Next, $n-1$ cascades of $\mathbf{A}_{h,k}$ gates with appropriate h and k values needed to target each of the remaining qudits must be determined. As a note, to maximally entangle n qudits, $(n-1)(r-1)$ total $\mathbf{A}_{h,k}$ are required. These gates will always have qudit $|\phi(0)_r\rangle$ as the control qudit and each of the remaining qudits will act as a target for an $\mathbf{A}_{h,k}$ cascade in order to become entangled as a group. It is known that each cascade will contain $r-1$ gates.

Preparing the desired entangled state requires the generation of $n-1$ $\mathbf{A}_{h,k}$ cascades. Each qudit that will act as a target, the qudits ranging from $|\phi(1)_r\rangle$ to $|\phi(j)_r\rangle$, will be iterated through. As seen in the algorithm pseudocode, value of h is derived from the basis in the output state that becomes entangled. The value of k determines the extent of entanglement because it controls what modulo-add-k operation is implemented.

```
r = 3
qudits = 2
basis = [[0,0],[1,1],[2,2]]
input_state = [0,0]


gate_seq = find_ent_gen_ckt(r,qudits,input_state,basis)
for i in gate_seq:
    print("%s : %s"%(i,gate_seq[i]))
```

```
Input state: [0, 0]
Desired state: [[0, 0], [1, 1], [2, 2]]

Chrestenson gate: C_3

Needed controlled modulo-additions (Control index 0):
Target qubit index : 1
      A_(1,1)
      A_(2,2)

G : ['C_3', 'A_(1,1)', 'A_(2,2)']
c : [None, 0, 0]
t : [0, 1, 1]
```

Fig. 5. Sample output of generator circuit synthesis to prepare $\frac{1}{\sqrt{3}}(|00_3\rangle + |11_3\rangle + |22_3\rangle)$ from ground state $|00_3\rangle$.

A demonstration of generator circuit synthesis is included in Fig. 5. In this example, the structure of the radix-3 maximal entanglement generator from Fig. 3 is determined using the parameters of the circuit input, $|00_3\rangle$, and the desired entangled quantum state basis values of $|00_3\rangle, |11_3\rangle, |22_3\rangle$.

TABLE I
REQUIRED GENERATOR CIRCUIT COMPONENTS FOR TWO-QUDIT MAXIMALLY ENTANGLED STATE PREPARATION

Input	**Basis States of Maximally Entangled Output**	$\mathbf{C}_r$ **Operator**	$\mathbf{A}_{h,k}$ **Operators** ($\|\phi(0)_r\rangle$ control and $\|\phi(1)_r\rangle$ target)
$\|00_2\rangle$	$\|00_2\rangle, \|11_2\rangle$	$\mathbf{C}_2$	$\mathbf{A}_{1,1}$
$\|11_2\rangle$			$\mathbf{A}_{0,1}$
$\|00_3\rangle$	$\|00_3\rangle, \|11_3\rangle, \|22_3\rangle$	$\mathbf{C}_3$	$\mathbf{A}_{1,1}$,$\mathbf{A}_{2,2}$
$\|11_3\rangle$			$\mathbf{A}_{0,2}$,$\mathbf{A}_{2,1}$
$\|22_3\rangle$			$\mathbf{A}_{0,1}$,$\mathbf{A}_{1,2}$
$\|00_4\rangle$	$\|00_4\rangle, \|11_4\rangle, \|22_4\rangle, \|33_4\rangle$	$\mathbf{C}_4$	$\mathbf{A}_{1,1}$,$\mathbf{A}_{2,2}$, $\mathbf{A}_{3,3}$
$\|11_4\rangle$			$\mathbf{A}_{0,3}$,$\mathbf{A}_{2,1}$,$\mathbf{A}_{3,2}$
$\|33_4\rangle$			$\mathbf{A}_{0,1}$,$\mathbf{A}_{1,2}$,$\mathbf{A}_{2,3}$
$\|00_5\rangle$	$\|00_5\rangle, \|11_5\rangle, \|22_5\rangle, \|33_5\rangle, \|44_5\rangle$	$\mathbf{C}_5$	$\mathbf{A}_{1,1}$,$\mathbf{A}_{2,2}$, $\mathbf{A}_{3,3}$, $\mathbf{A}_{4,4}$
$\|22_5\rangle$			$\mathbf{A}_{0,3}$,$\mathbf{A}_{1,4}$,$\mathbf{A}_{3,1}$,$\mathbf{A}_{4,2}$
$\|44_5\rangle$			$\mathbf{A}_{0,1}$,$\mathbf{A}_{1,2}$,$\mathbf{A}_{2,3}$, $\mathbf{A}_{3,4}$
$\|00_6\rangle$	$\|00_6\rangle, \|11_6\rangle, \|22_6\rangle, \|33_6\rangle, \|44_6\rangle, \|55_6\rangle$	$\mathbf{C}_6$	$\mathbf{A}_{1,1}$,$\mathbf{A}_{2,2}$, $\mathbf{A}_{3,3}$, $\mathbf{A}_{4,4}$, $\mathbf{A}_{5,5}$,
$\|22_6\rangle$			$\mathbf{A}_{0,4}$,$\mathbf{A}_{1,5}$,$\mathbf{A}_{3,1}$,$\mathbf{A}_{4,2}$,$\mathbf{A}_{5,3}$
$\|55_6\rangle$			$\mathbf{A}_{0,1}$,$\mathbf{A}_{1,2}$,$\mathbf{A}_{2,3}$, $\mathbf{A}_{3,4}$, $\mathbf{A}_{4,5}$
$\|00_7\rangle$	$\|00_7\rangle, \|11_7\rangle, \|22_7\rangle, \|33_7\rangle, \|44_7\rangle, \|55_7\rangle, \|66_7\rangle$	$\mathbf{C}_7$	$\mathbf{A}_{1,1}$,$\mathbf{A}_{2,2}$, $\mathbf{A}_{3,3}$, $\mathbf{A}_{4,4}$, $\mathbf{A}_{5,5}$,$\mathbf{A}_{6,6}$
$\|33_7\rangle$			$\mathbf{A}_{0,4}$,$\mathbf{A}_{1,5}$,$\mathbf{A}_{2,6}$,$\mathbf{A}_{4,1}$,$\mathbf{A}_{5,2}$,$\mathbf{A}_{6,3}$
$\|66_7\rangle$			$\mathbf{A}_{0,1}$,$\mathbf{A}_{1,2}$,$\mathbf{A}_{2,3}$, $\mathbf{A}_{3,4}$, $\mathbf{A}_{4,5}$,$\mathbf{A}_{5,6}$
$\|00_8\rangle$	$\|00_8\rangle, \|11_8\rangle, \|22_8\rangle, \|33_8\rangle, \|44_8\rangle, \|55_8\rangle, \|66_8\rangle, \|77_8\rangle$	$\mathbf{C}_8$	$\mathbf{A}_{1,1}$,$\mathbf{A}_{2,2}$, $\mathbf{A}_{3,3}$, $\mathbf{A}_{4,4}$, $\mathbf{A}_{5,5}$,$\mathbf{A}_{6,6}$,$\mathbf{A}_{7,7}$
$\|33_8\rangle$			$\mathbf{A}_{0,5}$,$\mathbf{A}_{1,6}$,$\mathbf{A}_{2,7}$,$\mathbf{A}_{4,1}$,$\mathbf{A}_{5,2}$,$\mathbf{A}_{6,3}$,$\mathbf{A}_{7,4}$
$\|77_8\rangle$			$\mathbf{A}_{0,1}$,$\mathbf{A}_{1,2}$,$\mathbf{A}_{2,3}$, $\mathbf{A}_{3,4}$, $\mathbf{A}_{4,5}$,$\mathbf{A}_{5,6}$,$\mathbf{A}_{6,7}$
$\|00_9\rangle$	$\|00_9\rangle, \|11_9\rangle, \|22_9\rangle, \|33_9\rangle, \|44_9\rangle, \|55_9\rangle, \|66_9\rangle, \|77_9\rangle, \|88_9\rangle$	$\mathbf{C}_9$	$\mathbf{A}_{1,1}$,$\mathbf{A}_{2,2}$, $\mathbf{A}_{3,3}$, $\mathbf{A}_{4,4}$, $\mathbf{A}_{5,5}$,$\mathbf{A}_{6,6}$,$\mathbf{A}_{7,7}$,$\mathbf{A}_{8,8}$
$\|44_9\rangle$			$\mathbf{A}_{0,5}$,$\mathbf{A}_{1,6}$,$\mathbf{A}_{2,7}$,$\mathbf{A}_{3,8}$,$\mathbf{A}_{5,1}$,$\mathbf{A}_{6,2}$,$\mathbf{A}_{7,3}$,$\mathbf{A}_{8,4}$
$\|88_9\rangle$			$\mathbf{A}_{0,1}$,$\mathbf{A}_{1,2}$,$\mathbf{A}_{2,3}$, $\mathbf{A}_{3,4}$, $\mathbf{A}_{4,5}$,$\mathbf{A}_{5,6}$,$\mathbf{A}_{6,7}$,$\mathbf{A}_{7,8}$

TABLE II
REQUIRED GENERATOR CIRCUIT COMPONENTS FOR MULTI-QUDIT MAXIMALLY ENTANGLED STATE PREPARATION

Input	**Basis States of Maximally Entangled Output**	$\mathbf{C}_r$ **Operator**	**Target Qudit** ($\|\phi(0)_r\rangle$ control)	$\mathbf{A}_{h,k}$ **Operators**
$\|012_4\rangle$	$\|000_4\rangle, \|111_4\rangle, \|222_4\rangle, \|333_4\rangle$	$\mathbf{C}_4$	$\|\phi(1)_4\rangle$	$\mathbf{A}_{0,3}$,$\mathbf{A}_{2,1}$,$\mathbf{A}_{3,2}$
			$\|\phi(2)_4\rangle$	$\mathbf{A}_{0,2}$,$\mathbf{A}_{1,3}$,,$\mathbf{A}_{3,1}$
$\|0123_4\rangle$	$\|0000_4\rangle, \|1111_4\rangle, \|2222_4\rangle, \|3333_4\rangle$	$\mathbf{C}_4$	$\|\phi(1)_4\rangle$	$\mathbf{A}_{0,3}$,$\mathbf{A}_{2,1}$,$\mathbf{A}_{3,2}$
			$\|\phi(2)_4\rangle$	$\mathbf{A}_{0,2}$,$\mathbf{A}_{1,3}$,,$\mathbf{A}_{3,1}$
			$\|\phi(3)_4\rangle$	$\mathbf{A}_{0,1}$,$\mathbf{A}_{1,2}$,,$\mathbf{A}_{2,3}$
$\|012_5\rangle$	$\|000_5\rangle, \|111_5\rangle, \|222_5\rangle, \|333_5\rangle, \|444_5\rangle$	$\mathbf{C}_5$	$\|\phi(1)_5\rangle$	$\mathbf{A}_{0,4}$,$\mathbf{A}_{2,1}$,$\mathbf{A}_{3,2}$,$\mathbf{A}_{4,3}$
			$\|\phi(2)_5\rangle$	$\mathbf{A}_{0,3}$,$\mathbf{A}_{1,4}$,$\mathbf{A}_{3,1}$,$\mathbf{A}_{4,2}$
$\|0123_5\rangle$	$\|0000_5\rangle, \|1111_5\rangle, \|2222_5\rangle, \|3333_5\rangle, \|4444_5\rangle$	$\mathbf{C}_5$	$\|\phi(1)_5\rangle$	$\mathbf{A}_{0,4}$,$\mathbf{A}_{2,1}$,$\mathbf{A}_{3,2}$,$\mathbf{A}_{4,3}$
			$\|\phi(2)_5\rangle$	$\mathbf{A}_{0,3}$,$\mathbf{A}_{1,4}$,$\mathbf{A}_{3,1}$,$\mathbf{A}_{4,2}$
			$\|\phi(3)_5\rangle$	$\mathbf{A}_{0,2}$,$\mathbf{A}_{1,3}$,$\mathbf{A}_{2,4}$,$\mathbf{A}_{4,1}$
$\|01234_5\rangle$	$\|00000_5\rangle, \|11111_5\rangle, \|22222_5\rangle, \|33333_5\rangle, \|44444_5\rangle$	$\mathbf{C}_5$	$\|\phi(1)_5\rangle$	$\mathbf{A}_{0,4}$,$\mathbf{A}_{2,1}$,$\mathbf{A}_{3,2}$,$\mathbf{A}_{4,3}$
			$\|\phi(2)_5\rangle$	$\mathbf{A}_{0,3}$,$\mathbf{A}_{1,4}$,$\mathbf{A}_{3,1}$,$\mathbf{A}_{4,2}$
			$\|\phi(3)_5\rangle$	$\mathbf{A}_{0,2}$,$\mathbf{A}_{1,3}$,$\mathbf{A}_{2,4}$,$\mathbf{A}_{4,1}$
			$\|\phi(4)_5\rangle$	$\mathbf{A}_{0,1}$,$\mathbf{A}_{1,2}$,$\mathbf{A}_{2,3}$,$\mathbf{A}_{3,4}$

As mentioned previously, all of these basis states will have a probability magnitude of $\frac{1}{\sqrt{3}}$ because that is the value of each of the elements of the 0^{th} column of the $\mathbf{C}_3$ transform.

When generating a maximally-entangled qudit state, it may be necessary to begin with a set of qudits that are initialized to a particular basis before transformation procedures. Depending on the original quantum state, different cascades of $\mathbf{A}_{h,k}$ operations must be implemented to achieve a targeted entangled output. To address this scenario, additional examples of synthesized circuit cascades for systems ranging from $r = 2$ to $r = 9$ can be found in Table I for the two-qudit case. This table includes sample input circuit values, found in column one, that target a set of maximally entangled basis states, found in column two. The required $\mathbf{C}_r$ operator is in column three and the synthesized $\mathbf{A}_{h,k}$ cascade needed to generate the desired linear combination of basis states is found in column four. Other input qudit and $\mathbf{A}_{h,k}$ operator combinations also generate the Table I entangled outputs, but Table I is not all inclusive in the interest of space. As a note, since only two qudits become entangled only $n - 1 = 2 - 1 = 1$ cascade is required. More examples of synthesis are included in Table II. In these results, entangled states including more than two qudits are analyzed for radix-4 and radix-5 systems. These circuit descriptions would be constructed using the form of Fig. 4.

VI. Conclusion

We have described how non-binary quantum computation can potentially lead to advantages since more information processing per atomic operation can occur. The technical difficulties that prevented wide-spread adoption of MVL or higher-radix computing with conventional electronic computers is not present in the case of gate-model QCs. Thus, the use of higher-radix computational models should be considered. We also described the fundamental importance of the phenomenon of quantum entanglement and the fact that it is used in virtually all of the well-known QC algorithms and communications protocols that are defined for binary QCs. Next, we described how the bipartite entangled pair generator known as a Bell state generator for binary systems can be used to motivate analogous means of entanglement generation for radix-r algorithms and circuits. The concept of entanglement, and in particular, maximal entanglement, was described for high-radix QIS both mathematically, with several physical examples given. Finally, we derived a systematic methodology for the synthesis of maximal entanglement generation circuits and algorithms and provided an example with our prototype synthesis tool.

In the future, we plan to extend the methodology to account for various forms of partial entanglement such that circuits can be generated that yield subsets of entangled qudits. We also plan to incorporate this methodology into our existing general purpose QC compiler and synthesis tool [28].

References

[1] A. K. Ekert, "Quantum cryptography based on bell's theorem," *Physical review letters*, vol. 67, no. 6, p. 661, 1991.

[2] D. G. Enzer, P. G. Hadley, R. J. Hughes, C. G. Peterson, and P. G. Kwiat, "Entangled-photon six-state quantum cryptography," *New Journal of Physics*, vol. 4, no. 1, p. 45, 2002.

[3] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*. IEEE, 1994, pp. 124–134.

[4] M. Lanzagorta, "Quantum radar," *Synthesis Lectures on Quantum Computing*, vol. 3, no. 1, pp. 1–139, 2011.

[5] C. H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, and W. K. Wootters, "Teleporting an unknown quantum state via dual classical and einstein-podolsky-rosen channels," *Physical review letters*, vol. 70, no. 13, p. 1895, 1993.

[6] J. Yin, Y. Cao, Y.-H. Li, S.-K. Liao, L. Zhang, J.-G. Ren, W.-Q. Cai, W.-Y. Liu, B. Li, H. Dai *et al.*, "Satellite-based entanglement distribution over 1200 kilometers," *Science*, vol. 356, no. 6343, pp. 1140–1144, 2017.

[7] K. N. Smith and M. A. Thornton, "Entanglement in higher-radix quantum systems," in *IEEE International Symposium on Multiple-Valued Logic (ISMVL)*. IEEE, 2019.

[8] ——, "Higher dimension quantum entanglement generators," *Journal on Emerging Technologies in Computing (to appear)*, 2019.

[9] D. M. Miller and M. A. Thornton, "Multiple valued logic: Concepts and representations," *Synthesis lectures on digital circuits and systems*, vol. 2, no. 1, pp. 1–127, 2007.

[10] C. Q. Choi, "Qudits: The real future of quantum computing?" *IEEE Spectrum Magazine*, 2017.

[11] G. Gibson, J. Courtial, M. J. Padgett, M. Vasnetsov, V. Pas'ko, S. M. Barnett, and S. Franke-Arnold, "Free-space information transfer using light beams carrying orbital angular momentum," *Optics express*, vol. 12, no. 22, pp. 5448–5456, 2004.

[12] T. Zhong, H. Zhou, R. D. Horansky, C. Lee, V. B. Verma, A. E. Lita, A. Restelli, J. C. Bienfang, R. P. Mirin, T. Gerrits *et al.*, "Photon-efficient quantum key distribution using time–energy entanglement with high-dimensional encoding," *New Journal of Physics*, vol. 17, no. 2, p. 022002, 2015.

[13] J. M. Lukens and P. Lougovski, "Frequency-encoded photonic qubits for scalable quantum information processing," *Optica*, vol. 4, no. 1, pp. 8–16, 2017.

[14] N. T. Islam, *High-Rate, High-Dimensional Quantum Key Distribution Systems*. Springer, 2018.

[15] T. Liu, Q.-P. Su, J.-H. Yang, Y. Zhang, S.-J. Xiong, J.-M. Liu, and C.-P. Yang, "Transferring arbitrary d-dimensional quantum states of a superconducting transmon qudit in circuit qed," *Scientific reports*, vol. 7, no. 1, p. 7039, 2017.

[16] J. Randall, S. Weidt, E. Standing, K. Lake, S. Webster, D. Murgia, T. Navickas, K. Roth, and W. Hensinger, "Efficient preparation and detection of microwave dressed-state qubits and qutrits with trapped ions," *Physical Review A*, vol. 91, no. 1, p. 012322, 2015.

[17] J. Randall, A. Lawrence, S. Webster, S. Weidt, N. Vitanov, and W. Hensinger, "Generation of high-fidelity quantum control methods for multilevel systems," *Physical Review A*, vol. 98, no. 4, p. 043414, 2018.

[18] P. Gokhale, J. M. Baker, C. Duckering, N. C. Brown, K. R. Brown, and F. T. Chong, "Asymptotic improvements to quantum circuits via qutrits," in *ACM/IEEE International Symposium on Computer Architecture (ISCA)*. IEEE, 2019.

[19] M. Enríquez, I. Wintrowicz, and K. Życzkowski, "Maximally entangled multipartite states: a brief survey," in *Journal of Physics: Conference Series*, vol. 698, no. 1. IOP Publishing, 2016, p. 012003.

[20] R. Thew, K. Nemoto, A. G. White, and W. J. Munro, "Qudit quantum-state tomography," *Physical Review A*, vol. 66, no. 1, p. 012303, 2002.

[21] M. Kues, C. Reimer, P. Roztocki, L. R. Cortés, S. Sciara, B. Wetzel, Y. Zhang, A. Cino, S. T. Chu, B. E. Little *et al.*, "On-chip generation of high-dimensional entangled quantum states and their coherent control," *Nature*, vol. 546, no. 7660, p. 622, 2017.

[22] H. Chrestenson *et al.*, "A class of generalized walsh functions." *Pacific Journal of Mathematics*, vol. 5, no. 1, pp. 17–31, 1955.

[23] N. Y. Vilenkin, "Concerning a class of complete orthogonal systems," in *Dokl. Akad. Nauk SSSR, Ser. Math*, no. 11, 1947.

[24] K. N. Smith, T. P. LaFave, D. L. MacFarlane, and M. A. Thornton, "A radix-4 chrestenson gate for optical quantum computation," in *2018 IEEE 48th International Symposium on Multiple-Valued Logic (ISMVL)*. IEEE, 2018.

[25] K. N. Smith, T. P. LaFave Jr, D. L. MacFarlane, and M. A. Thornton, "Higher-radix chrestenson gates for photonic quantum computation," *Journal of Applied Logics - IfCoLoG Journal of Logics and their Application*, vol. 5, no. 9, pp. 1781–1798, 2018.

[26] M. A. Thornton, D. W. Matula, L. Spenner, and D. M. Miller, "Quantum logic implementation of unary arithmetic operations," in *Multiple Valued Logic, 2008. ISMVL 2008. 38th International Symposium on*. IEEE, 2008, pp. 202–207.

[27] R. A. Bertlmann and P. Krammer, "Bloch vectors for qudits," *Journal of Physics A: Mathematical and Theoretical*, vol. 41, no. 23, p. 235303, 2008.

[28] K. N. Smith and M. A. Thornton, "A quantum computational compiler and design tool for technology-specific targets," in *ACM/IEEE International Symposium on Computer Architecture (ISCA)*. IEEE, 2019.

Experimental Insights from the Rogues Gallery

Jeffrey S. Young*, Jason Riedy†, Thomas M. Conte*, Vivek Sarkar*, Prasanth Chatarasi*, Sriseshan Srikanth*
*School of Computer Science
Georgia Institute of Technology
Atlanta, GA
{jyoung9,conte,vsarkar,cprasanth,seshan}@gatech.edu
†School of Computational Science and Engineering
jason.riedy@cc.gatech.edu

I. Introduction to the Rogues Gallery

The Rogues Gallery is a new deployment for understanding next-generation hardware with a focus on unorthodox and uncommon technologies. This testbed project was initiated in 2017 in response to Rebooting Computing efforts and initiatives. The Gallery's focus is to acquire new and unique hardware (the rogues) from vendors, research labs, and start-ups and to make this hardware widely available to students, faculty, and industry collaborators within a managed data center environment. By exposing students and researchers to this set of unique hardware, we hope to foster cross-cutting discussions about hardware designs that will drive future performance improvements in computing long after the Moore's Law era of cheap transistors ends. We have defined an initial vision of the infrastructure and driving engineering challenges for such a testbed in a separate document, so here we present highlights of the first one to two years of *post-Moore* era research with the Rogues Gallery and give an indication of where we see future growth for this testbed and related efforts.

The important research insights from this testbed (so far) are the following:

- **Post-Moore computing research is built on a hierarchy of novel architectures with varying levels of software support.** Of the current rogues, the Emu has the most "developed" software stack, but this means that it still lacks critical libraries, compiler optimizations, and APIs for interfacing with target applications. Neuromorphic, quantum, and other more revolutionary architectures still lack much of the needed compiler and library infrastructure to map meaningful applications.
- **We can demonstrate small "wins" from technologies like the Emu Chick or Hybrid Memory Cube but these results indicate the need for deeper study and more focused engineering.** Our initial Emu results indicate a suitability for graph analysis, but we cannot yet run large graphs or at scale due to load imbalance and thread migration issues. Likewise, the packet-oriented nature of the Hybrid Memory Cube (HMC) interface allows for more focused latency and BW tradeoff studies, but applications research using a combined HMC and FPGA platform is limited by the lack of well-supported high-level synthesis (HLS) tools for FPGAs and similar programmable devices. Both of these early prototypes indicate a focused need for more research studies and engineering efforts.
- **Tools, runtimes, and benchmarking are an important challenge for post-Moore research that needs more research investment as an enabling infrastructure.** Tools like the Habanero programming model and runtime [1], Kokkos portability API [2], and benchmarking are "high-effort" investments. However, their respective benefits can justify the cost, if we can build sufficient communities around platforms and tools that are convincing to the research community and funding agencies.
- **Education and engagement with undergrads will be important elements to push post-Moore research into the forefront for students.** There is currently a strong push from industry and funding agencies for students and researchers to tackle machine learning topics on current-day architectures. We need to engage students in post-Moore computing efforts by bridging from current machine learning to topics that require future architectures. We detail one example of how we are working to map from machine learning applications to post-Moore architectures in Section VIII.

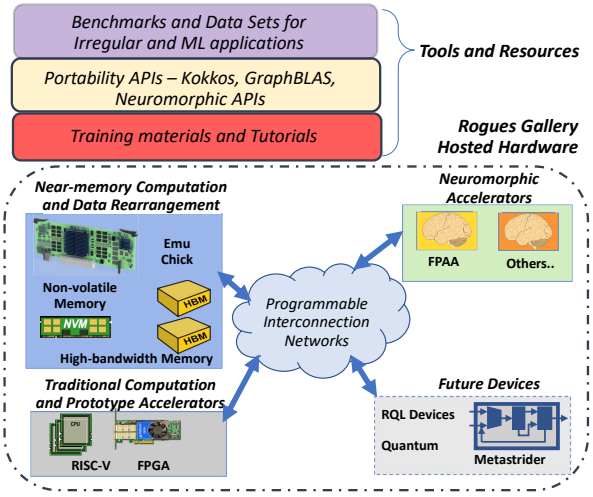


Fig. 1. High-level overview of the CRNCH Rogues Gallery

978-1-7281-5221-9/19/$31.00 ©2019 IEEE

II. The Rogues Gallery

As Figure 1 demonstrates, the CRNCH Rogues Gallery initially encompasses five different areas of research: 1) **Traditional computation and accelerators** that include enabling technologies like FPGAs, RISC-V, and related HPC technologies that provide a good bridge to post-Moore hardware. 2) **Near-memory computation and data rearrangement technologies** like High-bandwidth memories (HBM), NVM (as typified by newer 3D XPoint devices), near-storage accelerators, and prototypes like the Emu Chick [3], [4]. 3) **Neuromorphic accelerators**, which extend our conception of machine learning accelerators to brain-inspired computing with lower power requirements and real-time processing power. 4) **Revolutionary Architectures** that include sparse accelerators like our local project, Strider, emerging quantum efforts, reversible, and thermodynamic computing. 5) **Tools and resources** that are critical to map today's algorithms and applications of interest to novel architectures, especially those that are truly revolutionary. We present examples of each of these categories from the Rogues Gallery in the following sections.

III. Traditional Architectures - Reconfigurable Computing

Field Programmable Gate Arrays are not post-Moore architectures in themselves, but they can enable the evaluation of novel memory systems and accelerators like the neuromorphic DANNA implementation [5]. Recent work with the Rogues Gallery has focused on compilation tools for FPGAs and evaluating memory systems like the Hybrid Memory Cube [6], which has been made available for researchers as part of a Micron-supported Xilinx FPGA and HMC module called the AC-510. This platform has enabled several recent publications [7], [8] that are focused on low-level characterizations of this type of 3D stacked memory. As Figure 2 shows, the packet-oriented nature of HMC requests allows for packaging multiple DRAM read requests into a larger payload that can then be sent to the memory with related trade-offs in latency or improved overall bandwidth utilization of the serialized link. While Micron has recently moved away from future HMC designs, the similarities of HMC accesses with traditional networking payloads and the abstracted nature of the HMC's DRAM interface may play a role in future post-Moore successors to High-bandwidth memory (HBM) including those enabled by packet-oriented technologies like Gen-Z [9].

More traditional FPGA research results have included the usage of Intel and Xilinx platforms for research into runtimes and compiler frameworks. Most notably, standard Arria 10 boards provide an opportunity to evaluate joint industry and academic research on the Temporal to Spatial (T2S) programming system [10] that allows for decoupling of different spatial optimizations and orientations from the functionality of an accelerated kernel like a matrix-multiply operation. Results from this study show that dense tensor algebra kernels (GEMM, Tensor-Times Matrix, etc.) can easily be mapped to an Intel FPGA platform with a 1.76X speedup

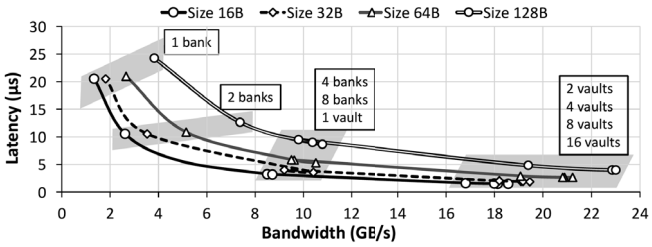


Fig. 2. Characterization of the AC-510 FPGA+HMC platform shows clear trade-offs for latency versus bandwidth as packet sizes change. [8]

for GEMM over an NDRange-based OpenCL implementation and up to 88% of a hand-tuned HDL implementation.

Ongoing work in this space is looking at mapping machine learning and high-performance computing kernels to reconfigurable platforms using compiler optimizations and systems like T2S and related research projects like OpenARC [11] and Chisel to reduce the overhead of testing new accelerators from that required with hand-tuned HDL or OpenCL codes.

IV. Near-memory Computation

We envision that near-memory processing architectures will increasingly migrate computation near high-bandwidth and non-volatile memories as in previous work [12], [13], [14], [15] and will be supplemented by an evolving set of near-storage processors like the recently announced Western Digital SweRV RISC-V core. In our local testbed, we have been focused on characterizing and evaluating optimizations for current near-memory systems like HMC and novel architectures like the Emu system. The Emu architecture focuses on improving random-access bandwidth scalability by migrating lightweight *Gossamer* threads to data and by emphasizing fine-grained memory access. We leave the more detailed description of this system to other related work [3], [16], [4], [17]. Our currently evolving prototype contains eight nodes that are each organized into eight gossamer cores (also referred to as a nodelet) and that are clocked at 175 MHz with DDR4 DRAM memory that is clocked at 1600MHz. The current generation of Emu system, the Emu Chick, includes one stationary processor for each of the eight nodelets contained within a node. The Emu Chick provides an interesting use case as an intermediate-level rogue in that it has a basic Linux-based OS on the stationary cores, a detailed single-threaded simulator, and support for Cilk++ and some basic Python functionality. While there are many missing libraries for the Emu architecture, students have used the testbed to perform characterization experiments, run graph-oriented benchmarks, and test basic machine learning algorithms using a SciKit-Learn interface.

Detailed microbenchmark characterization for this architecture has been explored in [18], so we present two sample results that demonstrate initial "wins" for irregular algorithm design. Figure 3 shows a custom pointer chasing benchmark that was designed to test the irregular access capabilities of the Emu system in a more fine-grained manner than the

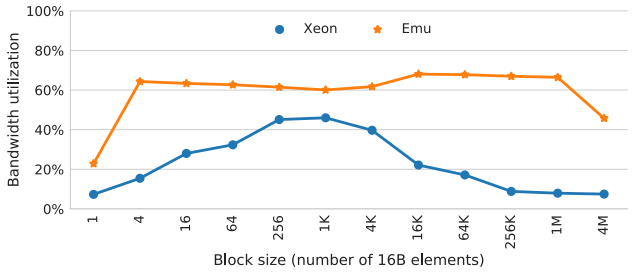


Fig. 3. Bandwidth utilization of pointer chasing, compared between Sandy Bridge Xeon and Emu (64 nodelets) [18]

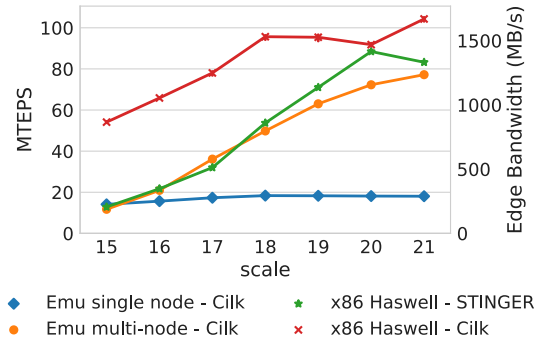


Fig. 4. Streaming BFS on the Emu Chick versus x86 Cilk and STINGER for balanced Erdös-Rényi) graphs. Scale: $\log_2$ of the number of vertices. [19]

traditional GUPS benchmark. Blocks of data in a linked list are increasingly shuffled to simulate different levels of random access, and the CPU system (a Sandy Bridge box) exhibits a somewhat expected bandwidth utilization trend line that follows the caching and paging trend lines up to 16K sized blocks. At the same time, the Emu is able to achieve a relatively consistent bandwidth utilization result for all but the smallest and largest block sizes.

The results in Figure 4 compare Emu single node (8 cores) and multi-node (64 cores) with a Haswell server system running a streaming BFS application using either Cilk or a recent state-of-the-art STINGER [20] implementation. Despite the differences in clock rate and memory speed for the Emu and CPU systems, the Chick is able to achieve similar rates of edge traversal for the input balanced graphs. This result indicates that at least for small amounts of data, the cacheless Emu architecture can perform as well as a heavily optimized CPU-based system. Note that the Emu currently does not perform nearly as well for unbalanced RMAT-style graphs due to load imbalance issues with the current code implementation.

Interestingly, both of these positive results were somewhat limited in their impact for the Emu architecture due to a later analysis [18] that showed that the current Chick prototype is compute-bound due to the underlying FPGA boards that are used to construct its fabric and gossamer core architecture. This further analysis demonstrates that **we need to be careful in extrapolating larger trends for post-Moore architectures from small "wins" that show positive progress for key applications**.

A. *Compiler-oriented optimizations*

Even though the Emu system is designed to improve the performance of data-sensitive workloads exhibiting weak-locality, the thread migrations across nodelets can also hamper the performance if overhead from the thread migration dominates the benefits achieved through the migration. Also, frequent thread creations can hamper the performance because thread creation on a remote nodelet results in migrating to that remote nodelet and spawning locally [21], [16].

Recent work using the Rogues Gallery Emu Chick system [22] measures the total number of migrations arising from a set of popular graph applications such as Bellman-Ford's algorithm for the single-source shortest path problem (SSSP), triangle counting, and conductance, using an in-house simulation environment of the Emu prototype. These measurements showed that many unnecessary and redundant thread migrations occurred because of naive algorithmic implementations. To address these unnecessary thread migrations, two high-level compiler optimizations, loop fusion and edge flipping, and one low-level compiler transformation that incorporates hardware support for remote atomic updates were explored to address overheads arising from thread migration, creation, and atomic operations.

A preliminary evaluation of these compiler transformations was performed by manually applying them for SSSP, triangle counting, and conductance over a set of RMAT graphs from Graph500. RMAT graphs (edges of these graphs are generated randomly with a power-law distribution) were used with the number of vertices running from 2^6 to 2^{14} (scale 6 through 14) and average degree 16 as specified by Graph500 [23]. These graphs were generated using the utilities present in the STINGER framework [24]. This evaluation targeted a single node of the Emu hardware prototype, and summary results are presented for the combined set of optimizations on the three types of input graphs in Figure 5. Applying all of thes optimizations results in an overall geometric mean reduction of 22.12% in thread migrations [22]. This preliminary study clearly motivates further exploration of the implementation of automatic compiler transformations to alleviate these thread migration overheads arising from running graph applications on the Emu system.

V. Neuromorphic

The Field Programmable Analog Array (FPAA) [25] implements a combined analog and digital board that can implement many analog and neuromorphic architectures [26], [27]. The FPAA combines a 2D array of ultra-low-power floating-gate analog plus digital blocks with a driving 16-bit MSP430 microprocessor. The exploration and development platform consists of a USB-attached board with multiple analog input ports (Figure 6).

For a device manufactured on a 350nm CMOS process, the FPAA provides a very low-power platform for neuromorphic, machine learning, and classification tasks. For example, the

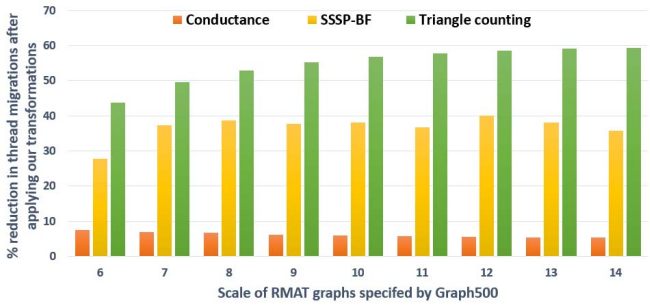


Fig. 5. Experimental evaluation (i.e., %reduction in thread migrations) of three graph algorithms (Conductance, SSSP-BF and Triangle counting) on the RMAT graphs from scales 6 to 14 specified by Graph500. Transformations applied on the algorithms: Conductance/SSSP-BF/Triangle counting: (Node fusion)/(Edge flipping and Remote updates)/ (Remote updates). The evaluation is done a single node (8 nodelets) of the Emu system [22].

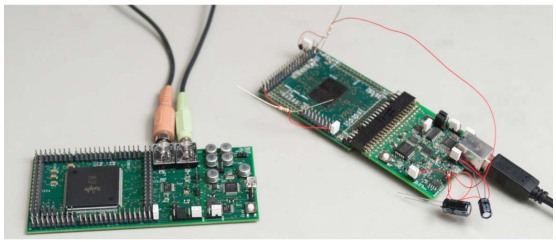


Fig. 6. The Field Programmable Analog Array (FPAA)

FPAA uses 23 μW to recognize the word "dark" in the TIMIT database [26]. Similarly, classifying acoustic signals from a knee joint requires 15.29 μW [27]. Both of these types of computations are performed in real time, so these power savings translate directly to energy savings and help to provide justification for further research in mixed analog and digital hardware for multiple applications.

Our current focus for research with the FPAA platform is focused on extending existing classification and spiking neural network examples to support higher level neuromorphic APIs like the TennLab exploratory framework [28] and Sandia National Lab's N2A [29]. Currently the FPAA is primarily programmed with a graphical interface using Scilab and XCos tools, so we initially need to provide a mapping from high-level neuromorphic APIs to the modular and macro-block programming environment that is used to create mixed analog and digital designs on the FPAA. While this initial engineering effort may not be groundbreaking in terms of post-Moore research, it is a critical effort to bring this novel hardware to a wider community of potential users.

VI. Revolutionary Architectures

The final category of post-Moore devices are those that are revolutionary in terms of upending the traditional von Neumann model for computing. While we group reversible, thermodynamic, quantum, and some specialized accelerators in this category, we currently are investigating two types of revolutionary architectures, sparse non-von Neumann accelerators and quantum computing enabling software and algorithms.

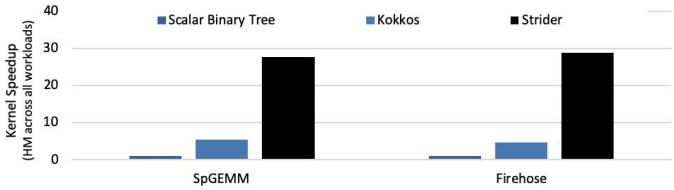


Fig. 7. Intelligent and dynamic memory-centric marshalling of sparse data with the Strider architecture significantly improves DRAM performance and results in over a magnitude of speedup for sparse reductions [31], [32].

A. Accelerators for Sparse Data

Conventional architectures are inefficient for a class of emerging workloads that exhibit low locality of reference. These sparse data applications suffer from a high cache miss ratio and expose DRAM (main memory) latency to the critical path. Recent research [30], [31], [32] has revealed that DRAM-centric sparse representations, algorithms and architectures have the potential of improving the state-of-the-art performance and energy by an order of magnitude. In particular, these works have focused on the sparse reduction kernel, wherein an associative operation is applied to the values of two or more key-value pairs that share the same key. For example, in generalized sparse matrix-matrix multiplication (SpGEMM, used in a variety of domains including graph analytics and HPC [33]), the accumulation phase is nothing but a sparse reduction problem where the "keys" are the matrix indices of the non-zero partial products ("values") that need to be summed (reduction operator).

Figure 7 demonstrates the tremendous speedup achieved due to using such an approach with our custom architecture, Strider, when compared with software binary tree and Kokkos-based approaches to the same problem. Speedups of up to 30x are measured as an average across a variety of SpGEMM-based workloads as well as for Firehose, a cybersecurity benchmark that models soft real time events. While these results are obtained via implementations in cycle-accurate simulators, FPGA synthesis of the underlying hardware operations [34] suggests that we could pursue a future integration of such accelerators for irregular data streams in the reconfigurable arena of the Rogues Gallery (Section III).

B. Quantum

The Rogues Gallery focuses on the programming and systems level of quantum computing while leveraging both local [35] and remote "Noisy Intermediate-Scale Quantum" devices (NISQ) [36] systems for intermediate results. The strenuous physical requirements for hosting quantum systems limits their physical deployment, so we focus more on easing intermediate access steps for algorithm development and helping users access available remote testbeds.

Programs for NISQ devices must adapt to the noise and errors [37]. This currently requires multiple runs along with adapting both algorithms and hardware mappings. The Rogues Gallery framework already recognizes systems where stability is an issue and extends directly to quantum sampling.

Beyond all of the systems issues lie the educational aspects. Quantum computing requires many mental pivots from classical computing. Section VIII briefly outlines our undergraduate efforts for quantum platforms leveraging existing frameworks like Qiskit[38] and building on existing training programs like NSF's EPicQC quantum computing tutorials and summer schools.

VII. Software Resources

Making new hardware available is a key component of a successful post-Moore testbed, but we argue that the tools and benchmarks made available are critical to build an interested community around specific architectures.

A. Benchmarking from Micro to Macro

Initial research into post-Moore architectures has resulted in a rich set of microbenchmarks that can be used and modified to test new architectures. For instance, characterizations of the Emu Chick system at our institute has led to a novel pointer-chasing benchmark that can be run on CPU-based systems as well as on the Emu platform, streaming graph benchmarks, sparse-matrix vector microkernels, and related graph and sparse microbenchmarks from related characterizations [17]. Likewise, local tensor libraries like ParTI [39] have variants for traditional systems and the Emu system and are supported by growing, collaborative datasets like the FROSTT tensor repository [40].

Figure 8 shows results for traditional HPC-oriented systems with the newly released Spatter benchmark suite [41], which enables users to test variations of gather and scatter operations and to characterize and evaluate patterns of indexed accesses that occur commonly in sparse algorithms and HPC applications. These results show the correlation between the sparsity of accesses, where the sparsity of 16 is equivalent to accessing one out of every 16 elements, and the effective bandwidth of these accesses when compared to a more traditional STREAM benchmark. While some current systems like the KNL struggle with reasonably sparse gather operations, newer GPUs like the V100 and AVX-enabled CPUs like the Skylake can effectively perform gather operations with high amounts of performance.

Our current focus is on extending Spatter by adding new backends for post-Moore architectures, including the cacheless Emu architecture, FPGAs with OpenCL, Metastrider, and neuromorphic accelerators. We anticipate that these cacheless architectures may be much better than traditional CPU- and GPU-based platforms at performing these types of critical memory accesses.

B. Portability Tools and Libraries

In addition to benchmarking efforts, there are early efforts to explore both the Kokkos performance portability API [42] and the GraphBLAS [43] on the Emu Chick in collaboration with other labs and universities. These engineering efforts contribute to the overall community around a tool and enable a larger number of applications to be tested on a novel post-Moore architecture. For instance, the ongoing development of an Emu Cilk backend for Kokkos would enable the execution of many of the Mantevo mini-apps and other Kokkos-enabled applications that have been ported by researchers at Sandia National Labs and other DoE-associated laboratories. It is highly unlikely that graduate students could port this same set of codes to the Emu Chick by generating custom Cilk kernels for a set of mini-apps.

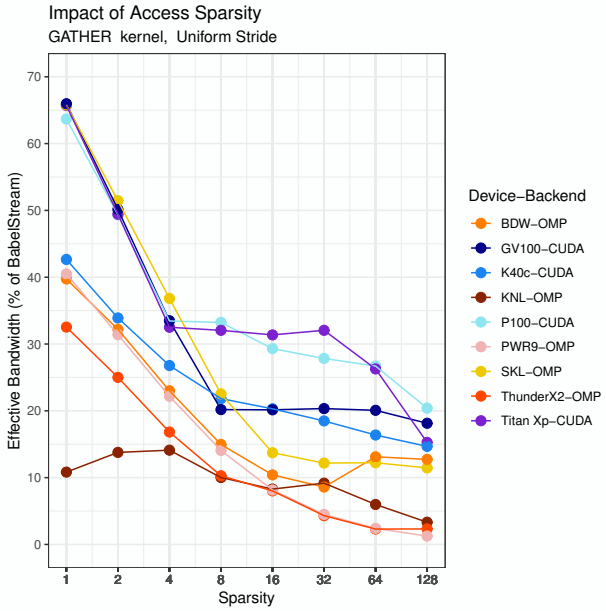


Fig. 8. Evaluating CUDA and OpenMP Gather with the Spatter benchmark suite. [41]

Likewise, we look to build on related external libraries and APIs for neuromorphic computing to enable a wider audience for neuromorphic hardware like the FPAA. With a limited number of researchers in the post-Moore computing space and many high-effort infrastructure pieces to build, we should look to grow our community's research impact by building on and improving existing tools and frameworks where at all possible.

VIII. Education and Outreach

One important aspect of the Rogues Gallery is ensuring that knowledge of novel platforms grows beyond the labs from which they come. Many new ideas are tested only locally, educating a only few graduate students about a platform's benefits and drawbacks. Making the platforms more widely available combined with providing tutorial and educational material should accelerate novel computing research. We also organize sessions at scientific computing conferences that bring together potential users, the Rogues Gallery, and other test beds.

A. Training and Demonstration

In April 2018 we held a neuromorphic workshop combining invited speakers with *hands-on* FPAA demonstrations. The hands-on portion required physical attendance. Participants organized into small groups, each of which had a FPAA to set up and use.

Our more recent approach focuses on remote access to our platforms. Rogues Gallery recently set up a JupyterLab[1] environment for tutorials. These allow active participation for those who want it, and a pre-made demonstration for those who would rather listen and read. The pre-made portion also is useful when the venue's network is unreliable. Even without making new Jupyter kernels, the environment permits editing code, running compilers, visualizing results, and even shell access. This proved useful for tutorials run at ASPLOS 2019 and PEARC 2019.

Tutorial and presentation material is made available through our institutional website and a separate Gitlab website[2]. Soon we hope these tutorials can be run entirely remotely, which would ease access from classes.

B. Education and Undergraduate Research

The authors had experience with novel platforms as undergraduates back when computing was not as homogeneous as it is now. We understand the benefits of exposing undergraduates to more than the few dominant platforms. One benefit to hosting the Rogues Gallery at a university is integrating the Gallery into undergraduate education.

Our initial undergraduate research class, part of the Vertically Integrated Projects program [44], provides novel architecture access for early computing and engineering students. The students are engaged and self-organized into groups focused on the FPAA, the Emu Chick, and integrating quantum simulators like Qiskit [38]. The students interested in quantum computing have learned initial skills from tutorials provided by efforts like the NSF EPiQC program and have identified the need for both diagrammatic and programmatic expression of quantum algorithms. These undergraduate students are *excited* to use novel architectures and provide feedback on how their preparation does or does not match the expectations in many platforms' documentation. Additionally, many of our microbenchmarking and initial engineering efforts provide a good entry point for undergraduate students looking to get involved with post-Moore computing research.

We also provide access to external graduate students at multiple universities. So far the access has been mainly limited to final projects for parallel computing classes, but we anticipate providing more materials that can extend the reach of post-Moore computing into traditional classwork.

IX. Looking to the Future of post-Moore Testbeds

We provide an overview of recent research results that have been enabled by the Rogues Gallery testbed to provide a sampling of what near-term and future research for post-Moore computing is ongoing at our institute. As with other larger architectural testbeds like CENATE [45] at Pacific Northwest National Lab, ExCL[3] at Oak Ridge National Lab, and Sandia HAAPS[4], we are focused on different aspects of the post-Moore computing landscape with a slightly different but overlapping audience of industry collaborators, researchers, and students.

To make sense of this broad research landscape we have attempted to organize the emerging candidate architectures into a basic classification scheme focused on how "near-term" a potential architecture might be. We propose a few initial lessons learned that reinforce our common need to investigate new post-Moore candidate architectures at a deep level while supplementing them with a research organizaton built around tools, benchmarking, and common APIs. Finally, we propose that efforts like our own undergraduate post-Moore course and large-scale education and tutorial efforts like those supported by NSF (e.g., EPiQC) are critical to growing a thriving community around post-Moore architecture testbeds.

Acknowledgments

This work is supported in part by Micron's and Intel's hardware donations, the NSF XScala project (ACI-1339745), the NSF SuperSTARLU project (OAC-1710371), and IARPA. We also thank Eric Hein, Janice McMahon and the rest of the team at Emu for their assistance in setting up and supporting the Emu Chick system for our users. Special thanks goes to Dr. Jennifer Hasler for her support of the FPAA and related training and tutorial materials for students.

Sandia National Laboratory supports the Rogues Gallery VIP undergraduate research class, and parts of this research have been funded through the Laboratory Directed Research and Development (LDRD) program at Sandia. Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.

References

[1] D. Majeti and V. Sarkar, "Heterogeneous Habanero-C (H2C): A portable programming model for heterogeneous processors," in *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*, May 2015, pp. 708–717.

[2] H. Carter Edwards, C. R. Trott, and D. Sunderland, "Kokkos," *J. Parallel Distrib. Comput.*, vol. 74, no. 12, pp. 3202–3216, Dec. 2014.

[3] T. Dysart, P. Kogge, M. Deneroff, E. Bovell, P. Briggs, J. Brockman, K. Jacobsen, Y. Juan, S. Kuntz, and R. Lethin, "Highly scalable near memory processing with migrating threads on the Emu system architecture," in *Workshop on Irregular Applications: Architecture and Algorithms (IA3)*. IEEE, 2016, pp. 2–9.

[4] J. S. Young, E. Hein, S. Eswar, P. Lavin, J. Li, J. Riedy, R. Vuduc, and T. M. Conte, "A microbenchmark characterization of the emu chick," *CoRR*, 2018.

[5] J. P. Mitchell, M. E. Dean, G. R. Bruer, J. S. Plank, and G. S. Rose, "DANNA 2: Dynamic adaptive neural network arrays," in *Proceedings of the International Conference on Neuromorphic Systems*, ser. ICONS '18. New York, NY, USA: ACM, 2018, pp. 10:1–10:6.

[1] https://jupyterlab.readthedocs.io/en/stable/

[2] https://gitlab.com/crnch-rg

[3] https://excl.ornl.gov/

[4] https://www.sandia.gov/asc/computational_systems/HAAPS.html

[6] *Hybrid Memory Cube Specification 1.0*, Hybrid Memory Cube Consortium, 2013.

[7] R. Hadidi, B. Asgari, B. A. Mudassar, S. Mukhopadhyay, S. Yalamanchili, and H. Kim, "Demystifying the characteristics of 3D-stacked memories: A case study for Hybrid Memory Cube," in *2017 IEEE International Symposium on Workload Characterization (IISWC)*, Oct 2017, pp. 66–75.

[8] R. Hadidi, B. Asgari, J. Young, B. A. Mudassar, K. Garg, T. Krishna, and H. Kim, "Performance implications of NoCs on 3D-stacked memories: Insights from the Hybrid Memory Cube," in *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, April 2018, pp. 99–108.

[9] G.-Z. Consortium, "Gen-Z overview (whitepaper)," *Gen-Z (online)*, 2016. [Online]. Available: http://genzconsortium.org/wp-content/uploads/2016/11/Gen-Z-Overview-V1.pdf

[10] N. Srivastava, H. Rong, P. Barua, G. Feng, H. Cao, Z. Zhang, D. Albonesi, V. Sarkar, W. Chen, P. Petersen *et al.*, "T2S-Tensor: Productively generating high-performance spatial hardware for dense tensor computations," in *The 27th IEEE International Symposium On Field-Programmable Custom Computing Machines (FCCM)*, 2019.

[11] S. Lee, J. Kim, and J. S. Vetter, "OpenACC to FPGA: A framework for directive-based high-performance reconfigurable computing," in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2016, pp. 544–554.

[12] S. Lloyd and M. Gokhale, "In-memory data rearrangement for irregular, data-intensive computing," *Computer*, vol. 48, no. 8, pp. 18–25, Aug 2015.

[13] J. C. Beard, "The sparse data reduction engine: Chopping sparse data one byte at a time," in *Proceedings of the International Symposium on Memory Systems*, ser. MEMSYS '17. New York, NY, USA: ACM, 2017, pp. 34–48.

[14] M. Shantharam, K. Iwabuchi, P. Cicotti, L. Carrington, M. Gokhale, and R. Pearce, "Performance evaluation of scale-free graph algorithms in low latency non-volatile memory," in *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, May 2017, pp. 1021–1028.

[15] S. Li, C. Xu, Q. Zou, J. Zhao, Y. Lu, and Y. Xie, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *Proceedings of the 53rd Annual Design Automation Conference*, ser. DAC '16. New York, NY, USA: ACM, 2016, pp. 173:1–173:6.

[16] E. Hein, T. Conte, J. S. Young, S. Eswar, J. Li, P. Lavin, R. Vuduc, and J. Riedy, "An initial characterization of the Emu Chick," in *The Eighth International Workshop on Accelerators and Hybrid Exascale Systems (AsHES)*, May 2018.

[17] M. Belviranli, S. Lee, and J. S. Vetter, "Designing algorithms for the EMU migrating-threads-based architecture," *High Performance Extreme Computing Conference 2018*, 2018.

[18] J. Young, E. R. Hein, S. Eswar, P. Lavin, J. Li, E. J. Riedy, R. W. Vuduc, and T. Conte, "A microbenchmark characterization of the Emu Chick," *CoRR*, vol. abs/1809.07696, 2018.

[19] E. R. Hein, S. Eswar, A. Yasar, J. Li, J. S. Young, T. M. Conte, Ü. V. Çatalyürek, R. Vuduc, E. J. Riedy, and B. Uçar, "Programming strategies for irregular algorithms on the Emu Chick," *CoRR*, vol. abs/1901.02775, 2019.

[20] J. Riedy, H. Meyerhenke, D. A. Bader, D. Ediger, and T. G. Mattson, "Analysis of streaming social networks and graphs on multicore architectures," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Kyoto, Japan, Mar. 2012.

[21] P. M. Kogge and S. K. Kuntz, "A Case for Migrating Execution for Irregular Applications," in *Proceedings of the Seventh Workshop on Irregular Applications: Architectures and Algorithms*, ser. IA3'17. New York, NY, USA: ACM, 2017, pp. 6:1–6:8.

[22] P. Chatarasi and V. Sarkar, "A Preliminary Study of Compiler Transformations for Graph Applications on the Emu System," in *Proceedings of the Workshop on Memory Centric High Performance Computing*, ser. MCHPC'18. New York, NY, USA: ACM, 2018, pp. 37–44.

[23] D. A. Bader, J. Berry, S. Kahan, R. Murphy, E. J. Riedy, and J. Willcock, "Graph500 Benchmark 1 (search) Version 1.2," Graph500 Steering Committee, Tech. Rep., Sep. 2011.

[24] D. Ediger, R. McColl, J. Riedy, and D. A. Bader, "STINGER: High performance data structure for streaming graphs," in *2012 IEEE Conference on High Performance Extreme Computing*, Sept 2012, pp. 1–5.

[25] S. George, S. Kim, S. Shah, J. Hasler, M. Collins, F. Adil, R. Wunderlich, S. Nease, and S. Ramakrishnan, "A programmable and configurable mixed-mode FPAA SoC," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 6, pp. 2253–2261, June 2016.

[26] J. Hasler and H. Marr, "Finding a roadmap to achieve large neuromorphic hardware systems," *Frontiers in Neuroscience*, vol. 7, p. 118, 2013.

[27] S. Shah, C. N. Teague, O. T. Inan, and J. Hasler, "A proof-of-concept classifier for acoustic signals from the knee joint on a FPAA," in *2016 IEEE SENSORS*, Oct. 2016, pp. 1–3.

[28] J. S. Plank, C. D. Schuman, G. Bruer, M. E. Dean, and G. S. Rose, "The TENNLab exploratory neuromorphic computing framework," *IEEE Letters of the Computer Society*, vol. 1, pp. 17–20, 2018.

[29] F. Rothganger, C. Warrender, D. Trumbo, and J. Aimone, "N2A: a computational tool for modeling from neurons to algorithms," *Frontiers in Neural Circuits*, vol. 8, p. 1, 2014.

[30] S. Srikanth, T. M. Conte, E. P. DeBenedictis, and J. Cook, "The superstrider architecture: Integrating logic and memory towards non-von neumann computing," in *2017 IEEE International Conference on Rebooting Computing (ICRC)*, Nov. 2017, pp. 1–8.

[31] E. P. DeBenedictis, J. Cook, S. Srikanth, and T. M. Conte, "Superstrider associative array architecture: Approved for unlimited unclassified release: Sand2017-7089 c," in *2017 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2017, pp. 1–7.

[32] S. Srikanth, A. Jain, J. Lennon, T. Conte, E. DeBenedictis, and J. Cook, "Metastrider: Architectures for scalable memory centric reduction of sparse data streams," *ACM Transactions on Architecture and Code Optimization (TACO)*, 2019.

[33] Y. Nagasaka, S. Matsuoka, A. Azad, and A. Buluç, "High-performance sparse matrix-matrix products on intel knl and multicore architectures," *arXiv preprint arXiv:1804.01698*, 2018.

[34] A. Jain, S. Srikanth, E. P. DeBenedictis, and T. Krishna, "Merge network for a non-Von Neumann accumulate accelerator in a 3D chip," in *2018 IEEE International Conference on Rebooting Computing (ICRC)*. IEEE, 2018, pp. 1–11.

[35] M. Ghadimi, V. Blūms, B. G. Norton, P. M. Fisher, S. C. Connell, J. M. Amini, C. Volin, H. Hayden, C.-S. Pai, D. Kielpinski, M. Lobino, and E. W. Streed, "Scalable ion–photon quantum interface based on integrated diffractive mirrors," *npj Quantum Information*, vol. 3, no. 1, p. 4, Jan 2017.

[36] J. Preskill, "Quantum Computing in the NISQ era and beyond," *Quantum*, vol. 2, p. 79, Aug. 2018.

[37] S. S. Tannu and M. K. Qureshi, "Not all qubits are created equal: A case for variability-aware policies for nisq-era quantum computers," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19. New York, NY, USA: ACM, 2019, pp. 987–999.

[38] G. Aleksandrowicz, T. Alexander, P. Barkoutsos, L. Bello, Y. Ben-Haim, D. Bucher, F. J. Cabrera-Hernádez, J. Carballo-Franquis, A. Chen, C.-F. Chen, J. M. Chow, A. D. Córcoles-Gonzales, A. J. Cross, A. Cross, J. Cruz-Benito, C. Culver, S. D. L. P. González, E. D. L. Torre, D. Ding, E. Dumitrescu, I. Duran, P. Eendebak, M. Everitt, I. F. Sertage, A. Frisch, A. Fuhrer, J. Gambetta, B. G. Gago, J. Gomez-Mosquera, D. Greenberg, I. Hamamura, V. Havlicek, J. Hellmers, Ł. Herok, H. Horii, S. Hu, T. Imamichi, T. Itoko, A. Javadi-Abhari, N. Kanazawa, A. Karazeev, K. Krsulich, P. Liu, Y. Luh, Y. Maeng, M. Marques, F. J. Martín-Fernández, D. T. McClure, D. McKay, S. Meesala, A. Mezzacapo, N. Moll, D. M. Rodríguez, G. Nannicini, P. Nation, P. Ollitrault, L. J. O'Riordan, H. Paik, J. Pérez, A. Phan, M. Pistoia, V. Prutyanov, M. Reuter, J. Rice, A. R. Davila, R. H. P. Rudy, M. Ryu, N. Sathaye, C. Schnabel, E. Schoute, K. Setia, Y. Shi, A. Silva, Y. Siraichi, S. Sivarajah, J. A. Smolin, M. Soeken, H. Takahashi, I. Tavernelli, C. Taylor, P. Taylour, K. Trabing, M. Treinish, W. Turner, D. Vogt-Lee, C. Vuillot, J. A. Wildstrom, J. Wilson, E. Winston, C. Wood, S. Wood, S. Wörner, I. Y. Akhalwaya, and C. Zoufal, "Qiskit: An open-source framework for quantum computing," 2019.

[39] "ParTI github," https://github.com/hpcgarage/ParTI, 2018.

[40] S. Smith, J. W. Choi, J. Li, R. Vuduc, J. Park, X. Liu, and G. Karypis, "FROSTT: The formidable repository of open sparse tensors and tools," http://frostt.io/, 2017.

[41] P. Lavin, J. Riedy, R. Vuduc, and J. Young, "Spatter: A customizable scatter/gather benchmark," http://spatter.io/, 2019. [Online]. Available: http://spatter.io/

[42] H. C. Edwards, C. R. Trott, and D. Sunderland, "Kokkos: Enabling manycore performance portability through polymorphic memory access patterns," *Journal of Parallel and Distributed Computing*, vol. 74, no. 12,

pp. 3202 – 3216, 2014, domain-Specific Languages and High-Level Frameworks for High-Performance Computing.

[43] J. Kepner, P. Aaltonen, D. Bader, A. Buluc, F. Franchetti, J. Gilbert, D. Hutchison, M. Kumar, A. Lumsdaine, H. Meyerhenke, S. McMillan, J. Moreira, J. D. Owens, C. Yang, M. Zalewski, and T. Mattson, "Mathematical foundations of the GraphBLAS," *CoRR*, 2016.

[44] J. Sonnenberg-Klein, R. T. Abler, E. J. Coyle, and H. H. Ai, "Multi-disciplinary vertically integrated teams: Social network analysis of peer evaluations for Vertically Integrated Projects (VIP) program teams," in *2017 ASEE Annual Conference & Exposition*. Columbus, Ohio: ASEE Conferences, June 2017.

[45] N. R. Tallent, K. J. Barker, R. Gioiosa, A. Marquez, G. Kestor, L. Song, A. Tumeo, D. J. Kerbyson, and A. Hoisie, "Assessing advanced technology in CENATE," *2016 IEEE International Conference on Networking, Architecture and Storage (NAS)*, Aug 2016.

Future Computing Systems (FCS) to Support "Understanding" Capability

Ray Beausoleil
System Architecture Lab
Hewlett Packard Labs
Palo Alto, CA, USA
ray.beausoleil@hpe.com

Kirk Bresniker
System Architecture Lab
Hewlett Packard Labs
Palo Alto, CA, USA
kirk.bresniker@hpe.com

Cat Graves
System Architecture Lab
Hewlett Packard Labs
Palo Alto, CA, USA
catherine.graves@hpe.com

Kimberly Keeton
System Architecture Lab
Hewlett Packard Labs
Palo Alto, CA, USA
kimberly.keeton@hpe.com

Suhas Kumar
System Architecture Lab
Hewlett Packard Labs
Palo Alto, CA, USA
suhas.kumar@hpe.com

Can Li
System Architecture Lab
Hewlett Packard Labs
Palo Alto, CA, USA
can.li@hpe.com

Dejan Milojicic
System Architecture Lab
Hewlett Packard Labs
Palo Alto, CA, USA
dejan.milojicic@hpe.com

Sergey Serebryakov
System Architecture Lab
Hewlett Packard Labs
Palo Alto, CA, USA
sergey.serebryakov@hpe.com

John Paul Strachan
System Architecture Lab
Hewlett Packard Labs
Palo Alto, CA, USA
john-paul.strachan@hpe.com

Thomas Van Vaerenbergh
System Architecture Lab
Hewlett Packard Labs
Palo Alto, CA, USA
thomas.van-vaerenbergh@hpe.com

***Abstract*—The massive explosion in data acquisition, processing, and archiving, accelerated by the end of Moore's Law, creates a challenge and an opportunity for a complete redesign of technology, devices, hardware architecture, software stack and AI stack to enable future computing systems with "understanding" capability. We propose a Future Computing System (FCS) based on a memory driven computing AI architecture, that leverages different types of next generation accelerators (e.g., Ising and Hopfield Machines), connected over an intelligent successor of the Gen-Z interconnect. On top of this architecture we propose a software stack and subsequently, an AI stack built on top of the software stack. While intelligence characteristics (learning, training, self-awareness, etc.) permeate all layers, we also separate AI-specific components into a separate layer for clear design. There are two aspects of AI in FCSs: a) AI embedded in the system to make the system better: better performing, more robust, self-healing, maintainable, repairable, and energy efficient. b) AI as the level of reasoning over the information contained within the system: the supervised and unsupervised techniques finding relationships over the data placed into the system.**

Developing the software and AI stack will require adapting to each redundant component. At least initially, specialization will be required. For this reason, starting with an interoperable, memory driven computing architecture and associated interconnect is essential for subsequent generalization. Our architecture is composable, i.e., it could be pursued in: a) its entirety, b) per-layer c) per component inside of the layer (e.g., only one of the accelerators, use cases, etc.); or d) exploring specific characteristics across the layers.

Keywords*—*Memory Driven Computing, accelerators, AI.

I. Introduction

The end of Moore's Law and the massive explosion of data acquisition creates an opportunity to rethink all computing layers, including the materials, devices, hardware architecture, software stack, algorithms, applications and use cases. As it is common in the cases of large disruption, a clear path from specialized to general is required for initial inventions to transfer to a broad adoption. In other words, initially, future computing systems (FCSs) will be highly specialized. Similarly, it is prudent not to bet on one technology, but to diversify across alternatives, especially at the bottom of the stack, such as for the materials and devices. For this reason, the architectures that are inclusive and adaptable to different hardware are the only path to eventual generalization [1].

In addition, to be able to reach the next level of AI systems, it is required to support "understanding" capability beyond the traditional characteristics of the AI, such as learning. One of the ways to describe understanding is through the Table 1, below.

Table 1. Defining "Understanding" Capability

	Deployment domain (Vertical)	Real-time	Evolving	Accountability/ Compliance
AI (learning)	Single	Offline training	Limited	No
+understanding	Across multiple	Continuously	Degrees of Self-*	Yes

To achieve "understanding" capability, the future computing system needs to evolve through awareness, cognition, training/learning and finally understanding. Awareness is constituted in the basics of autonomic computing and self-* characteristics [2][3][4]. It requires self-monitoring, self-control/management, self-healing, etc. Autonomous systems evolve through a deliberate, aided, autonomous, and adaptive phase [5]. To support FCS, autonomic behavior requires an extension for intelligence. Cognition requires memory, containing a knowledge database of previous experiences (relevant to the current problem) and a monitoring unit that maps the current situation to these previous

978-1-7281-5221-9/19/$31.00 ©2019 IEEE

experiences, and links them with the best sub-systems that can optimally solve the required subtasks to tackle the overall problem the system is facing. The memory needs to allow for both remembering and learning at different timescales (short term memory versus long term memory, etc.). Training/learning also consists of a sequence of evolutionary steps including: being trained on classes of data, learning (ability to gradually learn from the past), transfer learning (adapt weights trained on one class of problems to other classes) and self-learning (the system is able to autonomously identify the problems, divide it into subtasks, train itself to perform well on these subtasks and consequently also perform well on the initially detected main problem). Finally understanding is the highest degree which can react to circumstances that were not even trained/learned, in other words reason on-demand. To achieve all these levels, FCS requires substantial amounts of computing and memory.

In this paper, we will first explain how Memory-Driven Computing is a perfect starting point for the development of future computing architectures (Sec. II). Subsequently, we will highlight which challenges might hamper the development of FCSs (Sec. III). Next, we explain our architecture (Sec. IV, schematic shown in Fig. 1) In Sec. V we propose some potential applications and use cases. Finally, the expected operational characteristics of the architecture are discussed (Sec. VI).

II. Recent progress in memory-driven Computing

In the research and development community, we have been laying the groundwork for the development of future architectures of intelligent future computing systems by moving from a processor-centric to a memory-centric approach. This paradigm shift has been induced by the current explosion of data. We are currently generating zettabytes of data on a yearly base, an amount which nearly doubles every two years. Consequently, as data becomes the new currency, architectural changes are necessary to overcome the limitations of the traditional compute-centric model [6]. Data-centric designs call for an architectural approach that minimizes data duplication, enables ubiquitous and heterogeneous computing resources, and deals with resilience and security from the ground up. We are already seeing the traditional monolithic server disaggregating into a rack-scale architecture [7][8], where pools of storage, networking and compute resources can be flexibly organized in a software-defined manner to match different workload characteristics. This is the first step towards an even more radical memory-centric approach, which we call Memory-Driven Computing (MDC). Another aspect is the rise of accelerators, such as GPUs, revival of FPGAs, introduction of TPUs and new ones such as the Dot Product Engine [7]. And all of this is prerequisite for future computing systems.

MDC is based on the idea that in traditional architectures the memory needed to deal with these massive amounts of data is a scarce resource, as the processors function as gate-keepers paging data between cost-effective block mode IO persistence storage devices and scarce byte addressable volatile memory resulting in capacity, bandwidth and latency bottlenecks. These dichotomies (IO versus memory; block versus byte; volatile versus persistent) have been greatly exacerbated due to the loss of Dennard scaling greatly lessened potential frequency increases and lead to more and more feature integration (IO and memory controllers) alongside increasing numbers of homogeneous cores on massive System-on-Chip (SoC) designs. While it allowed for power and latency reductions for the moderate scale system, it limits the low-latency domain to the small amount of SoC direct attached DRAM and relegates any large scale storage and accelerators to the IO domain, potentially orders of magnitude higher in latency and requiring OS mitigated accesses as block devices.

The next scaling methodology will be 3D integration and both traditional memory technologies (Flash) and novel memories (Memristors/ReRAM/PCM/STTM) are well on this path due to their intrinsic low power density, repairability, sparing, and correctability. Whether logic will make the same transition is an open question. The MDC architecture anticipates both this capacity and diversity of persistent, byte-addressable memories as well as diverse accelerators, by opening and then extending the memory-semantic fabric to the enclosure, rack and aisle scale. This allows the placing of general purpose processors as peers to accelerators and memory controllers, some of the SoC/IO bottlenecks are eliminated and, consequently, truly massive datasets can be manipulated, while simultaneously achieving multiple orders of magnitude less energy/bit. Processors are still afforded the advantages of local, potentially high bandwidth memories, but they are logically unified in a single fabric address space with all memory resources on-fabric. In these systems, the traditional memory/storage paging hierarchy collapses, replacing traditional block-based storage (hard drives and SSDs) and traditional DRAM with byte-addressable non-volatile memory (NVM). As memory and storage converge into fast, persistent memory, we expect that the NVM pool will also shift from being directly attached to the main application CPU to becoming a shared resource that is accessible by all compute resources in the rack.

In 2017, Hewlett Packard Enterprise announced a prototype of the MDC architecture, which is the largest single-memory system currently known. This prototype had 160 TB of shared memory spread across 40 physical nodes, connected using a high-performance memory fabric protocol, Gen-Z, over a high-bandwidth, energy-efficient VCSEL-based photonic interconnect. It demonstrated performance enhancements on industry-relevant benchmark tasks, such as in-memory analytics (15x), similarity search (40x), large-scale graph interference (100x) and financial models (10,000x). The system ran an optimized Linux-based operating system and redesigned software programming tools that take advantage of the abundance of persistent memory. Based on the current system, HPE believes it could easily scale to a multi-exabyte single-memory system. In fact, the technology can successfully scale to a nearly limitless pool of memory and will serve as the foundation for computing systems of the future. Every computing category and device – from smartphones to supercomputers – has something to gain from this new computing paradigm. This work has implications for nearly every industry.

The MDC concept is heterogeneous by design and allows for task-specific processing. Consequently, inclusion of emerging accelerators in the MDC ecosystem will allow to leverage a mature software stack simplifying the design of a plug-and-play

hybrid interface that allows users to benchmark and compare the most advanced accelerator designs. In this unified platform, adhering to common interconnect protocols will reduce the overall development time for novel accelerators. In contrast, today, each computer component is connected using a different type of interconnect. Memory connects are using DDR, hard drives communicate via SATA, flash drives and GPUs via PCIe, etc. To address this issue, in 2016, a consortium called Gen-Z was unveiled, which comprises of leading technology companies dedicated to creating and commercializing the Gen-Z technology. The Gen-Z consortium includes over 50 members, including: AMD, ARM, Broadcom, Cray, Dell, HPE, Huawei, IDT, Micron, Samsung, SK Hynix, and Xilinx and it is constantly expanding and recruiting the list of members. Gen-Z is an open systems interconnect designed to provide memory semantic access to data and devices via direct-attached, switched or fabric topologies and is hence paving the way for composability with future accelerators.

The improvement in energy-efficiency, speed, and scale of future MDC implementations relies heavily on progress in a diverse set of technologies related to NVM (PCM, STT-RAM, Memristor, and 3D-Xpoint), interconnect (an evolution from VCSEL-based links to silicon photonics and beyond), network topology (replacement of traditional network topologies optimized for hybrid electro-optical network such as DragonFly, by topologies optimized for full-optical interconnects, and higher throughput and reduced congestion such as Hyper-X).

III. Feasibility of Developing a FCS

A. Design of the System to Support "Understanding"

MDC forms the optimal foundation for FCS, because it (1) is able to handle large amounts of data at high throughput/low latency for returning 'answers', (2) is scalable from edge systems to massive HPC systems, (3) eliminates communication barriers imposed by traditional von Neumann architectures and (4) is "data-intensive" vs. "compute-intensive only". However, FCS are only obtained by augmenting the MDC concept by incorporating functionality that allows for a degree of cognition and autonomy, and by doing it efficiently.

FCS can improve energy-efficiency over conventional platforms by several orders of magnitude by abandoning homogenous systems based on general purpose computing in favor of heterogeneous systems that can match workloads to optimized accelerators. For instance, in HPC, current machine learning applications benefit increasingly from access to FPGAs and GPUs for both inference and training. Training neural networks on specialized accelerators is a few orders of magnitude faster. In the next 3-5 years, we forecast an increase in the deployment of digital ASICs optimized for the ubiquitous matrix-vector multiplication (MVM) such as Google's TPU. However, we expect that the transition from digital to analogue or hybrid analogue/digital computing will be the game-changer in energy-efficiency for these workloads. One example is the MVM-unit based on memristor-crossbar arrays [9], which are 10-100x faster and 10-1000x more energy efficient than GPUs. Embedded in appropriate architectures, limitations of the analogue hardware such as a limited bit-resolution can be mitigated at the system level and these hybrid digital/analogue units enhance throughput, speed and efficiency in an extremely compact footprint.

Recent successes in deep learning, and upcoming applications such as autonomous driving have resulted in a plethora of different technologies that focus on either a digital, analogue, or hybrid way of the execution of MVMs or other core neural network functionality. By their neuromorphic design, these applications typically offload humans from tasks on which they already perform well. FCSs should aim beyond just replicating human intelligence, and aim to complement our brains by tackling challenges that go beyond our capabilities, such as solving NP-hard problems. These sets of problems are notoriously hard, scale exponentially as a function of system size, and appear in a variety of applications such as traffic flow optimization, supply chain management, airline scheduling, DNA sequencing, etc. Improving the efficiency of solving these problems would not only be a game changer for large-scale enterprises, but would also improve our day-to-day experience as individuals.

Today, the efficient solution to business-relevant Mixed Integer Programming (MIP) tasks is a problem-specific art-form, enabled by exact algorithms provided by software frameworks such as Gurobi or CPLEX. Initial attempts to come up with accelerators for meta-heuristics based on the concept of simulated annealing, by exploiting physics both in the quantum-regime (e.g., entanglement in super-conducting qubits [10]), and classical regime (e.g., memristor-Hopfield Neural Networks with chaotic memristors [11], Ising machines using symmetry-breaking in pulsed optical parametric oscillators [12] or integrated photonics [13]) indicate that hardware-accelerated meta-heuristics will change the current algorithm eco-system inside exact solvers (by allowing for better initial seeds) and consequently democratize the solution of these notoriously compute-intensive problems. Given the no-free lunch theorem [14], which solution technique will prevail is very problem-specific. Therefore, MDC's heterogeneity becomes critical as it allows to allocate different subroutines to the optimal accelerators, even on-the-fly by running problems in parallel on different flavors of accelerators. In addition, currently proposed annealing-based accelerators for combinatorial optimization are stochastic in nature. Good solutions are found by running the accelerators in parallel or sequential. In the latter-case there is potential for natural adaptability to time-varying problem definitions or compatibility with dynamic programming.

The memory will likely be non-volatile (due to power and scale requirements) which will require new programming models, such as managed data structures [15]. This model will have to address persistency, in a similar manner as transactions were addressed and be supported by processors which will understand that they deal with persistent memory and correctly manage caches. Similar problems are posed by sharing memory across many processing elements with different degrees and types of shared memory consistency. Because of the continued discrepancy in memory access latency, near-memory and in-memory computing will be a required programming model [16]. Enabled by computing in memory, reconfigurable computing will transpire the spectrum from reprogramming, NN weights training [17], and traditional course grained reconfigurable computing architectures.

B. FCS Computational Model

The most-efficient accelerators in the FCS framework will be based on analogue computing. When making these analogue accelerators, the goal is to leverage the physics of the hardware to efficiently emulate some computational functions from certain algorithms, e.g., use noise in hardware as a randomness source. To enable scalability, and an easy interface for users, it is important to combine the benefits of more mature technologies with the benefits of the emerging platforms by using a hybrid digital-analogue architecture. Indeed, analogue computing has suffered in the past from issues such as analogue noise, the need to cross the analog/digital domain, the lack of effective User Interfaces, etc., that prevented it to scale up and become useful in commercial applications. Most of all, digital computers scaled sufficiently to offset most of the analogue computing advantages. However, with the end of Moore's scaling, some of the analogue techniques are becoming viable alternatives again. In addition, recent inference and training algorithms seem to move to a regime where the performance on benchmark tasks becomes less sensitive to the bit-resolution of the weight matrices, which paves the way for lower-accuracy, but energy-efficient accelerators.

C. Hardware and Software Innovations Required

Hardware accelerators: The cognitive and learning capabilities of FCSs all rely on a knowledge base embedded in memory, and an energy-efficient ability to change its states during training. Current hardware accelerators are typically able to optimize the inference phase of machine learning algorithms, but a fast, high-yield mechanism with long endurance for hardware training is not yet commercially available. The research into this issue currently follows two main tracks: the unaltered transfer of traditional software learning algorithms to novel accelerator platforms (resulting in struggles with lack of linearly symmetric weight update mechanisms), the other approach is to take the physics from the accelerators into account from the beginning and alter the algorithms given device inaccuracies, imperfections and noise levels [18][19][20][21][22]. Due to their higher maturity, we expect the former set of technologies to be dominant for the next five years, but eventually software-hardware co-design will reach better performance in the subsequent decade. Upcoming technologies such as 3D memory stacking and heterogeneous integration, will allow for increased energy-efficiency, while scaling up the total memory bandwidth. Additionally, these technologies form a perfect starting point for the development of either neuromorphic or in-memory computing-based accelerators, reducing the overhead in movement of data between memory and processors, hence increasing energy-efficiency. ReRAM-related technologies clearly are reaching the levels of maturity to allow them to have impact on the overall computing industry, as companies and foundries such as Fujitsu, Panasonic and TSMC [23] are already offering or are looking into offering 1T1R-based products.

Photonics: In the next decade, we also expect that the maturation of silicon photonics, as an interconnect, will enable networks to scale to the pJ/bit level and below in a cost-effective manner [24]. Hybrid bonding of III-V on Si will allow for avoidance of some of the coupling losses, whereas the reduction of losses through adjoint-based design techniques combined with optical-proximity correction techniques, and the development of ultra-low loss SiN waveguides and a multi-layer platform, will increase the overall throughput of the optical links even more. In the current foundry-landscape, industry is increasingly committed to open Process Design Kits (PDKs) as a crucial part of the go-to-market strategy of upcoming technologies such as silicon photonics and flexible electronics. Open PDKs are a hardware equivalent of open source software. From a business perspective, the scaling required to make silicon photonics cost-effective can most effectively be obtained in a non-vertical way.

Memories: There are currently four main choices for non-volatile memory technology: PCM, STT-RAM, Memristor and 3D-Xpoint. Panasonic already has a product with NVM and it is believed that TSMC will have one as well. It is unclear who will be successful in productization, and when, but each will have impact on the processors and software stack, data structures programming for persistence, how to address the imminent huge scale beyond current address bits, and the fact that it will be shared across a plethora of traditional and new accelerators.

Hardware-Software Co-design: There are a number of challenges and opportunities in understanding the device characteristics. From the device-perspective: there are stochastic, spatial, and temporal variation when updating memristor connectivity. Traditional software algorithms, such as back-propagation, require challenging device specs, e.g., (preferentially linearly) symmetric weight updates. However, software-hardware co-optimization can help as algorithms can be adapted to the physics of the devices. From the circuit design point of view: we need ADCs and DACs, which unfortunately results in a huge power consumption. Performing more operations in the analogue domain mitigates this burden. At the architecture level there are implications on communication: does training need to happen in the same chip as inference? The main Von Neumann bottleneck is how to structure arrays and array sizes.

Software: Next generation systems blur the boundaries of firmware, OS, middleware, applications, and AI [25]. Synchronization, persistence, communication, and other software functions will be critical at all levels of the system. But there are more opportunities for new approaches altogether. Among many reasons why we use stochastic gradient descent in, e.g., AI applications, is because HW (GPUs) significantly accelerates this process. If we have even more compute power, can we come up with something beyond training multiple models in parallel and selecting the best one, such as, e.g., high dimensional derivative free optimizers?

Intelligent DevOps: Traditional software stacks benefited substantially from the DevOps model where development and operations are merging together. The new AI software can also benefit from DevOps, but it has to be adjusted. For example, what are the repositories equivalent to GitHub, how to express versioning of models and training data, etc.

Fault tolerance, security, and operational safety: FCSs will require extensive support to make them reliable, secure and safe. A number of prior technologies with adaptation could be adopted, e.g., redundancy for memory and computation (e.g., for weights) and sandboxing for security [26]. Using AI in the

lower FCS components can enable the efficient and reliable operation at scale. Some of the requirements for low operational temperature (e.g., for quantum computing) or substantial power consumption may not be productizable at all form factors, they are acceptable in Data centers or Cloud, but require alternatives at the edge/IoT.

D. Barriers and Enablers

When relying on analogue accelerators, there are issues related to **endurance**, **spatial and temporal variations**, **repeatability**, **robustness against noise-levels** and **scalability**. A way to deal with scalability is to work with hybrid analogue/digital systems, in which the analogue system is sized up into tiles close to the limit at which systematic errors and/or noise becomes prohibitive for overall system accuracy. Interestingly, analogue noise can also be considered a feature when emulating algorithms which are stochastic in nature, such as: stochastic gradient-descent, simulated annealing, Monte Carlo sampling, sampling from (restricted) Boltzmann machines, etc. By calibrating the noise distributions intrinsic to the hardware, and appropriately optimizing or adapting existing algorithms to these distributions, randomness can be emulated at unprecedented efficiency levels compared to currently available digital pseudo-random number generators. When working with hybrid analogue/digital systems, there is an important trade-off between potentially reduced energy-efficiency due to the presence of power-hunger ADCs and the benefits of signal restoration that data conversion back-and-forth between the digital and analogue domain offer.

In the world of ReRAM, read operations are typically faster than write operations, and write operations in the analogue world, because of higher accuracy requirements, typically are more cumbersome than in binary memory applications. Solving this issue either at the device, or at the algorithm level, will be a game changer in the overall power consumption of machine learning techniques in FCSs. Importantly, architecture decisions (e.g., choice of training on the same device as inference, or using dedicated devices, or traditional GPU-based solutions, for both approaches) can mitigate some of these initial child-diseases of neuromorphic hardware, and in the long run, we expect that a tight interplay between progress in device engineering and adaptation of machine learning algorithm to find sweet spots to minimize the data transfer induced bottlenecks during the training phase. For FCSs, the tightness in which training and inference can be embedded will determine the timescale in which FCSs can adapt to their continuously changing environment.

Programmability, ease of use. In MDC-based FCS, memory hierarchy is flat: PB-scale random access non-volatile memory with no disks. Thus, the conventional programming model where data is loaded from disks into memory, modified there and then stored back to disks will not be relevant anymore. Everything is in non-volatile memory and programmers will need a library that will allow them to create, modify, share, manage and store data there. FCS will need a library that (1) allows programmers to create known data structures (trees, arrays, graphs etc.) directly in NVM; (2) makes those data structures persistent, i.e., they reside in memory when application exists; (3) manages those data structures by providing a naming service (sharing data between processes), a support for multiple languages (e.g., C++, Java), an access and concurrency control. We call such library Managed Data Structures.

Sensitivity to quality of data and other aspects. Multi-tenant environment, diverse workloads and hardware components that may have impermanent working characteristics raise a question of data quality a FCS will use as input to self-awareness component. It should not only be robust to quality of such data but also immune to adversarial data that can be generated by compromised HW components or applications. Adversarial data is data that pretends to be normal data but statistically deviates from the distribution of a training dataset used to train a self-awareness module. A FCS needs to deal with noise in data. It can train a special type of neural network, called an autoencoder, to de-noise input data and reconstruct a clean signal. Same types of neural networks but with slightly different architecture can be used to identify data that does not belong to trained distribution and thus is abnormal data that may be a signal of some malfunction. For now, there is no good approach to deal with adversarial data and this needs to be solved. It is a challenging problem because the attack surface is enormous, potentially applied at any layer (device, hardware, software, AI, etc.) [27].

Composability. Given the diversity in compute requirements of the targeted application fields (e.g., Edge vs HPC), our architecture is composable by design. For instance, for some use cases, not all functionality of the architecture in its entirety might be required to deliver the desired performance, and, consequently, to reduce overhead in power consumption, or system size, some functionality might be omitted during deployment. In addition, by merging the different research efforts required to deliver FCS capabilities into one overarching research effort to deliver parts of a common system architecture, key breakthroughs related to either the overall architecture or its subparts can be amortized over the different application fields. Therefore, we propose to only pursue architectures that can be demonstrated either in: a) its entirety, b) per-layer c) per component inside of the layer (e.g., only one of the accelerators, use cases, etc.); or d) allow to explore specific characteristics across the layers. The key of our design is the usage of an interoperable, memory driven computing architecture and an associated interconnect as it will allow to bridge the specialization required for the novel types of accelerators with the subsequent generalization required for the demonstration of the entire architecture.

IV. Approach towards FCS

In Figure 1, we provide a schematic of an entire FCS architecture. As mentioned in the previous section, the architecture is composable, but in the current schematic we include the different stacks, their level of AI-awareness, and a color-coded indication of the level of maturity of the required subcomponents. In the following two subsections, we first discuss the technical constraints that need to be respected while developing the architecture, followed by a discussion of the features of the subcomponents in the stack of the proposed FCS implementation. At the bottom of the stack, we listed some example accelerators that could enhance the intelligence of

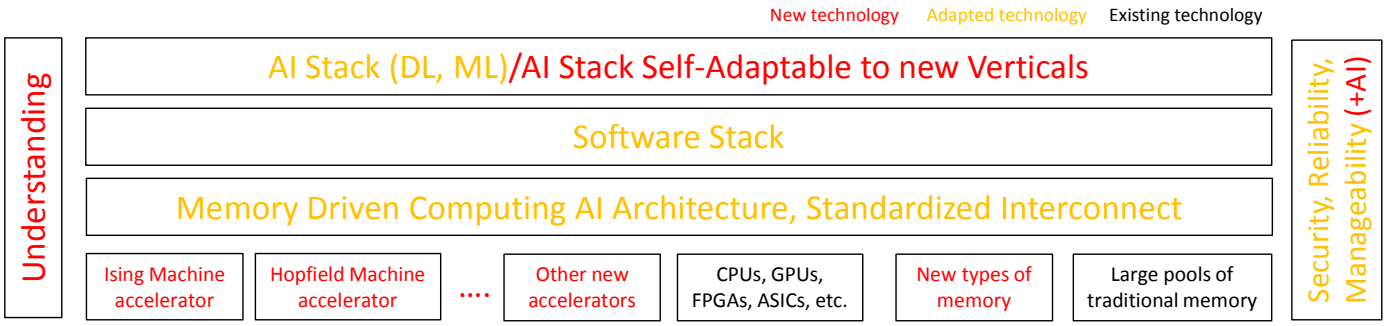


Figure 1. Architecture Future Computer System and AI to support Understanding Capability.

FCSs in an energy-efficient way, allowing it to operate at high speeds while maintaining a small footprint. This list is currently non-exhaustive, given that emerging applications and technologies might result in new needs for accelerators. Consequently, our architecture has been designed to be non-restrictive in the types of accelerators it can deal with.

A. Critical Technical Issues

Scaling: Ability to scale hardware; system software; algorithms; applications; AI/cognition, resulting in the increased understanding of overall FCSs. This is a vertical characteristic, in the sense that applies across all layers and any layer can become a scaling bottleneck.

Reliability and repeatability: because components can fail, layers need to be robust in presence of failures. In the past reliability was pushed to the higher layers for cost-effectiveness. AI-related results need to be repeatable and subject to attestation.

Security and trustworthiness: with the size of a FCS, the attack surface grows. The system security is only as strong as the weakest link. Therefore, new security technologies will have to be devised to address traditional and new problems. AI has to be trustworthy and explainable.

Reconfigurable computing will be required for broader adoption. Reconfiguration has to be fine grained, compiler-driven, tightly interacting with the architecture.

B. Approach

Our approach spans a spectrum among devices, hardware architecture, algorithms, self-awareness, user-assistance, and software stack. The rest of the section covers these areas.

Devices :

- *NVM*, which is cheap, energy-efficient, has low latency, is scalable, has high yield, reliability and endurance. Potential solutions are: PCM, memristors, spintronics, etc.
- *High-speed interconnects:* inter-chip and intra-chip. Silicon photonics is very promising, but when scaling optical links down to attojoule efficiency, we might want to extend current technology capabilities, by moving towards emerging technologies like plasmonics, photonic crystals, and/or look into multi-layer photonic circuits, and TIA-less receivers.
- *Randomness:* Shrinking electronic and photonic devices down to smaller footprints, and improving the energy-efficiency will result in devices which operate with a countable number of photons or electrons. Consequently, we expect increased stochastic behavior. This can be a feature, as noise in the hardware could be leveraged as a noise source instead of expensive digital circuits to emulate random number generators in stochastic algorithms.
- *A non-von-Neumann model of computing*, e.g., enabled by in-memory computing, to avoid costly movement of data. This will require new algorithms suited for in-memory computing, as well as compiler and operating system support and new applications design.
- *Trade-offs between analogue and digital* can result in savings in a balanced hybrid architecture by moving the computation between the analogue and digital domain.

Quantum Computing (QC) has achieved substantial recent interest and funding, and someday we could incorporate a QC as an accelerator for specific problems of broad commercial interest where quantum algorithms demonstrate an exponential advantage. Key challenges to QC remain: cryogenics; limited qubit connectivity; and (to date) demonstrations on toy problems only. Nevertheless, QC could strongly benefit from MDC as a classical interface to quantum resources, and MDC could in turn benefit from quantum thinking. For the last 40 years, we have learned to make reliable classical computers by first ignoring and then suppressing quantum effects. We should learn from the quantum community to harness quantum mechanics to build smaller, lower power, and cheaper classical computing technologies.

Hardware architecture:

Two primary hardware architecture principles govern the FCS.

A) FCSs will require a precision ensemble approach, the optimal computation, memory, and communications endpoints brought together for the duration of a workload. At the embedded scale, this could be static in the configuration of chiplets on a substrate or modules in a rugged enclosure. At the rack, aisle or datacenter scale, this could be dynamic, with a range of memory, general purpose and application specific accelerators marshalled for purpose, operating in ensemble as long as required to advance the state of data in persistent memory. Open, standard and scalable memory-semantic fabrics, like Gen-Z, enable the architectural, non-recurring and operational economics to allow precision endpoints to be delivered to market.

B) Relentless operational data capture and analysis connected back through the manufacturing and engineering processes. This enables the system to optimize itself against evolving workloads in a declarative management framework and enables continuous integration and deployment of quality and reliability improvements garnered from the entire operational population.

Algorithms:

- *Deep learning*: inference, RBMs for training of deep NNs.
- *Transfer learning:* reuse networks that have been trained for similar tasks, deploy weights that have been trained off-line and adapt them to the analogue hardware instantiation.
- *Meta-heuristics* for optimization problems such as simulated annealing.
- *Message passing algorithms* such as Gibbs sampling.
- FCSs are equipped with purpose-built pluggable HW/SW modules with algorithms that make the *system self-aware* and that assist in optimizing user workloads:
- System *continuously monitors runtime* characteristics.
- System uses the runtime data to *infer knowledge about itself and user workloads.*
- System uses the runtime data to *continuously learn and adapt.*
- *Learnable components detect:* a) adversarial input data (learn/understand concepts rather than transform input); and b) the best strategy to achieve goals – by cooperating (meta-classification) or by competing with each other (what generalized adversarial networks (GANs) do).

Self-awareness:

- *Time-to-failure prediction*. The FCS continuously analyzes its runtime characteristics and predicts chances of failures of its components.
- *Abnormal behavior detection* such as performance degradation due to, e.g., HW failure.
- *Intrusion detection system* that identifies behavior considered harmful and takes actions.

User assistance:

- *Monitor parameters* of user workloads and *reconfigure allocated HW resources* from various points of view such as energy consumption minimization.
- *Detect users workload bottlenecks* and suggest solutions (such as HW/SW update).

Software Stack:

Our experience with developing a software stack for a hybrid CMOS-memristor technology [9][28] shows that adopting a specialized ISA-programmable interface to extend an analogue based computer dramatically helps with programmability. Furthermore, it enhances memristor crossbars with general purpose execution units carefully designed to maintain area/energy efficiency and storage density. We have extended our accelerator design with a complete compiler to transform high-level code to the accelerator ISA and a detailed simulator for estimating performance and energy consumption. Our evaluations show that this hybrid design can achieve significant improvements compared to state-of-the-art CPUs, GPUs, and ASICs for ML. This way we balance specialization to address performance scale and power with generalization to maintain programmability and legacy in the ease of use [1].

V. Applications and Use cases

A. Applications

Financial risk management and derivative pricing. In the financial services industry, derivative products represent a multi-trillion dollar market, including options, swaps, swaptions, forwards and futures with underlying stocks, currencies and interest rates. Such products are priced using Monte Carlo simulations and they estimate the financial portfolio risk. The estimates are used by risk managers, senior managers, and regulators. The Monte Carlo simulations have a trade-off between speed and accuracy: a small number of simulations leads to fast, but inaccurate price, sensitivity and risk estimations; a large number of simulations leads to accurate, but slow estimates, raising the following problems:

- The risk desks cannot update their portfolio risk estimates more than once/twice a day, an exposure when the underlying market or the portfolios change significantly within the day.
- The trading desks need to price complex products in response to customer offers. The trading desk needs to be fast due to competition.
- MDC makes Monte Carlo pricing/estimations real-time (seconds), accelerating it up to 10000 compared to conventional approaches, by reformulating Monte Carlo in the following way:
  - Offline phase. Precomputing large amounts of representative statistics and storing them in memory (requires hundreds of petabytes of fast shared memory).
  - Online phase. Transforming a relevant subset of stored statistics with a small number of operations instead of performing simulations from scratch.

Processing time-varying graphs. Analyzing changes in a graph over a period of time is valuable in a wide range of domains from network security to social network analysis. However, the size and lack of data locality makes it hard, even impossible, in certain cases. That is why large-scale time-dependent graph analysis can leverage an MDC architecture. An example application that benefits from such technology is the analysis of news/trends in a social network. A social graph evolves in time and using its static snapshot to do analysis may result in wrong conclusions. MDC provides a natural way to work with such graphs:

- In general, graphs have no natural partitioning what makes it hard to distribute efficiently in memory on a conventional architectures.

- Because of a temporal aspect, real world graphs become extremely large, e.g., online e-commerce companies requiring hundreds of TB or telecommunication firms tens of PB.

Large scale graph inference. Graph processing is the key approach to solving large scale analytical problems, such as malware detection, genome analysis and online advertising. Due to graph sizes and the communication, conventional hardware is not suited well for such tasks. Traditionally, graphs have been used to answer questions about connectivity and path lengths, however, more powerful insights can be obtained by combining probability theory with the graphs. One typical example is real time detection of malicious domains accessed by an enterprise's hosts, exhibited through billions of HTTP proxy logs for a small list of known malicious and benign domains. The graphical model to this problem contains nodes that are either hosts or domains. Each vertex represents a binary-valued variable associated with a server or domain that indicates if it is malicious. An edge exists if the corresponding server has made a request to the corresponding domain. Through this graphical model one can infer the probability that a specific domain is malicious through a technique called Gibbs Sampling. MDC architecture provides features that makes such an inference real time:

- *A large pool of fast shared memory* is a natural place to store large graphs to be accessed by worker threads. The Facebook network size is over a billion of people and the friend links go to a trillion, the malware detection graph is 100's of millions computers and billions links, online advertising has 100s of millions people and millions transactions/day.
- *Scalability*. The number of computations involved in processing iterations is proportional to the number of edges. A large number of CPUs are required as the graph gets larger. Implementing this on a distributed system limits the scaling. MDC provides a fast mechanism to asynchronously communicate the state of the variables via shared memory.

B. Use Cases

The use cases have to be of commercial impact to be able to drive substantial investment in development. For example, gaming can have immediate deployment if it is of benefit to gaming community to improve: performance, how realistic games are, scale of deployment, quality of UI, etc. Autonomous vehicles already have push in the industry, but the highest degrees of self-driving vehicles autonomy will have similar requirements as FCS. Robots have finally started to gain some traction and adoption of FCS techniques could have immediate impact. However, to speed development over the course of 20 years, a grand challenge with increasing degrees of "Understanding" for a certain use case would be of most benefit.

VI. Operational Characteristics

A. Expected Performance and Robustness

While it is hard to predict the future, we made some predictions in the Table 2, at the end of this paper. The table is self-explanatory, but it suffices to say that we divide the future into three partially overlapping phases: exascale, heterogeneous, and post von Neumann. Each phase increases the readiness in support of an "Understanding" capability. While it is possible to predict both qualitatively and quantitatively in the first two phases, for the third phase it is only possible to predict the qualitative requirements. In addition and for the same reason, it is not possible to go into detailed prediction of some of the most interesting aspects, such as system uses of the runtime data to infer knowledge about itself and user workloads or time-to-failure prediction. These are very implementation specific and we can only go so deep in predicting them without the details of future technologies.

B. Approximate Power Specifications

For combinatorial optimization accelerators, e.g., memristor-based Hopfield neural networks operating at a 1GHz clock at room temperature, initial tests on academic benchmark problems indicate energy-efficiency performance enhancements over digital hardware of at least a 1000x, using a conservative architecture design [29] and a hybrid analogue-digital design. Unsurprisingly, this matches the numbers for the benefits memristor-crossbar array accelerators have for MVM compared to GPUs. For Ising machines, the photonic version of a combinatorial optimization accelerator, numbers can be enhanced with an additional 100x (in energy-efficiency and speed) by working with high-speed nonlinearities such as the Kerr-effect in integrated photonic chips [13], but these devices will take at least another decade to reach the maturity-level required with fabrication in volume fabs. Optical interconnects are close to reaching pJ/bit performance at 100Gb/s bandwidths and beyond for 0.10cents/bit. The next level aims to get to the aJ/bit level for some link applications [30], which will rely on emerging technologies such as TIA-less receivers, photonic crystals and/or plasmonics.

C. Proxy Applications, Benchmarks, and Metrics

In 5-10 years, we expect the emergence of new benchmarks, potentially seeded by the MIP community and similar to the graph500 list, ranking top performance with respect to speed, size, and energy-efficiency, for different heterogeneous systems capable of solving NP-hard problems. Similarly, we expect an equivalent of GUPS benchmark for the AI community, which could do dynamic inference and cognition across distributed knowledge bases similar to GUPS pointer chasing. To measure the degree of "understanding" of the FCS levels similar to autonomous vehicles could be adopted, with increasing degrees of self-*: acceleration, deceleration; +steering; +automated lane changes; +in case of failure pull to side of the road; fully automated under certain conditions; fully automated under all conditions. Instead, we could, e.g., introduce understanding of a) a limited set of specific data for which it was trained; b) multiple sets of data; c) a limited problem domain; d) multiple problem domains; e) any input data and any domain; f) fully dynamically adaptive understanding. See also [31].

VII. Summary

In this paper we have discussed the future computing systems that could support "understanding" capability. We presented feasibility, approach, applications and use cases, and finally operational characteristics.

While achieving these future computer systems is at least 20 years out in the future, there is an opportunity for a continuous

evolution of various aspects and demonstration through functional and non-functional improvements.

REFERENCES

[1] Pedro Bruel, Sai Rahul Chalamalasetti, Chris Dalton, Izzat El Hajj, Alfredo Goldman, Catherine Graves, Wen-mei Hwu, Phil Laplante, Dejan Milojicic, Geoffrey Ndu, John Paul Strachan, "Generalize or Die: Operating Systems Support for Memristor-based Accelerators," Proceedings of the 2017 IEEE International Conference on Rebooting Computing (ICRC'17), pages 1-8.

[2] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing," IEEE Computer Vol 36, No 1., 2003.

[3] D. A. Menasce and J. O. Kephart, Special Issue of IEEE Internet Computing on "Autonomic Computing," Vol 11 No 1, 2007.

[4] IEEE Future Directions Committee, "Future Directions Autonomous Systems", sites.ieee.org/futuredirections/2017/11/27/autonomous-systems-in-2018.

[5] IEEE Future Directions Autonomous Systems http://sites.ieee.org/futuredirections/2017/11/27/autonomous-systems-in-2018/

[6] P. Faraboschi, K. Keeton, T. Marsland & D. Milojicic (2015). "Beyond processor-centric operating systems." In proceedings of the USENIX HotOS 2015.

[7] Costa, P., "Towards rack-scale computing: Challenges and opportunities." In First International Workshop on Rack-scale Computing (2014), ACM.

[8] Kyathsandra, J., and Zhou, X., "Rack Scale Architecture: Designing the Data Center of the Future." http://bit.ly/idf14-rsa, Intel IDF14 Shenzhen, 2014.

[9] Ankit, A., El Hajj, I., Chalamalasetti, S.R., Ndu, G., Foltin, M., Williams, R.S., Faraboschi, P., Hwu, W.M., Strachan, J.P., Roy, K., Milojicic, D., "PUMA: A Programmable Ultra-efficient Memristor-based Accelerator for Machine Learning Inference," Accepted for publication at ACM ASPLOS, 2019.

[10] S. Boixo, et al. "Evidence for quantum annealing with more than one hundred qubits." Nature Physics 10, 218–224 (2014).

[11] S. Kumar, J. P. Strachan, & R.S. Williams (2017). "Memristors for analogue computing." Nature, 548(7667), 318–321. http://doi.org/10.1038/nature23307

[12] Ryan Hamerly, et al. (2018). Scaling advantages of all-to-all connectivity in physical annealers: the Coherent Ising Machine vs. D-Wave 2000Q, 1–17. arXiv:1805.05217v1

[13] N. Tezak *et al*., "Integrated Coherent Ising Machines Based on Self-Phase Modulation in Microring Resonators," in *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 26, no. 1, pp. 1-15, Jan.-Feb. 2020, Art no. 5900115. doi: 10.1109/JSTQE.2019.2929184

[14] Wolpert, D.H., Macready, W.G. (1997), "No Free Lunch Theorems for Optimization", IEEE Transactions on Evolutionary Computation 1, 67.

[15] Managed Data Structures https://github.com/HewlettPackard/mds

[16] Dejan Milojicic, Kirk Bresniker, Gary Campbell, Paolo Faraboschi, John Paul Strachan, Stan Williams, "Computing In-Memory, Revisited," Proceedings of the IEEE 38th International Conference on Distributed Computing Systems (ICDCS), Vienna, Austria, July 2018, Pages 1300-1309.

[17] Ankit, A., et al. "A Programmable Architecture for Neuromorphic Training using Highly Energy-efficient ReRAM," paper under submission.

[18] C. Li, et al. (2018). "Long short-term memory networks in memristor crossbars." ArXiv, 1(January). http://doi.org/10.1038/s42256-018-0001-4

[19] C. Li., et al. (2018). "Efficient and self-adaptive in-situ learning in multilayer memristor neural networks." Nature Communications, 9(1), 7–14. http://doi.org/10.1038/s41467-018-04484-2

[20] G. W. Burr (2019). "A role for analogue memory in AI hardware." Nature Machine Intelligence, 1(1), 10–11. http://doi.org/10.1038/s42256-018-0007-y

[21] S. Ambrogio et al. (2018). "Equivalent-accuracy accelerated neural-network training using analogue memory." Nature, 558(7708), 60–67. http://doi.org/10.1038/s41586-018-0180-5

[22] S. Agarwal et al. (2016). "Resistive memory device requirements for a neural algorithm accelerator." Proceedings of the International Joint Conference on Neural Networks, 2016–October, 929–938. http://doi.org/10.1109/IJCNN.2016.7727298

[23] C. Lee, H. Lin, C. Lien, Y. Chih, & J. Chang. (2017)., "A 1. 4Mb 40-nm embedded ReRAM macro with hybrid write verify for high endurance application." In IEEE Asian Solid-State Circuits Conference (Vol. 3, pp. 9–12).

[24] R.G. Beausoleil (2011). "Large-Scale Integrated Photonics for High-Performance computing." ACM Journal on Emerging Technologies in Computing Systems, 7(2). http://doi.org/10.1145/1970406.1970408

[25] P. Laplante, D. Milojicic, "Rethinking operating systems for rebooted computing," Proceedings of the 2016 IEEE International Conference on Rebooting Computing (ICRC'16), pages 1-8.

[26] Thomas M Conte, Erik P DeBenedictis, Avi Mendelson, Dejan Milojičić, "Rebooting Computers to Avoid Meltdown and Spectre," IEEE Computer, Volume 51, Issue 4, April 2018, Pages 74-77

[27] Brumley, Cunningham, Dalton, DeBenedectis, Dinca, Dubyak, Edwards, Hernandez, Horne, Johnson, Mastilovic, Matwyshyn, Mendelson, Milojicic, Moussouris, Shaw, Shoop, Tran, Walker, IEEE Report "Artificial Intelligence and Machine Learning Applied to Cybersecurity," March 2018.

[28] J. Ambrosi, R. Antunes, A. Ankit, S.R. Chalamalasetti, S. Chatterjee, I. El Hajj, G. Fachini, P. Faraboschi, M. Foltin, S. Huang, W-m. Hwu, G. Knuppe, S.V. Lakshminarasimha, D. Milojicic, M. Parthasarathy, F. Ribeiro, L. Rosa, K. Roy, P. Silveira, and J.P. Strachan, "Hardware-Software Co-Design for an Analog-Digital Accelerator for Machine Learning", Proceedings of the 2018 IEEE International Conference on Rebooting Computing (ICRC'18).

[29] Fuxi Cai, et al., "Memristor Hopfield Neural Networks." (2019, manuscript in preparation). https://arxiv.org/ftp/arxiv/papers/1903/1903.11194.pdf

[30] D. A.B. Miller (2017). "Attojoule Optoelectronics for Low-Energy Information Processing and Communications." Journal of Lightwave Technology, 35(3), 346–396. http://doi.org/10.1109/JLT.2017.2647779

[31] International Roadmap for Devices and Systems (IRDS™) 2017 Edition, Application Benchmarking, https://irds.ieee.org/images/files/pdf/2017/2017IRDS_AB.pdf.

Table 2. Evolution of Future Computing Systems to Achieve "Understanding" Capability.

Boundaries: <$500M, <20MW, single site

exascale → ← heterogeneity → ← post Moore →

degree of readiness of FCS with the "understanding" capability					
unknown	none	minimal	some	substantial	full

Layer		Year					Out of scope in 2040
		2020	2025	2030	2035	2040	
Focus		Compute-intensive	Data-intensive	Knowledge-intensive	Learning-intensive	Intelligence	Full understanding
quantitative	Capacity	1-5 PB (graphs?) (HBM)	10-20 PB (HBM)				Human memory
	Capability	200-300 PF(CMOS, 5nm)	1-2 EF (CMOS, 3nm)	Non-von, post-Moore	Built-in-heterogeneity	Self-evolving hardware	Regenerative, growing HW
	Injection	1/500 Bytes/FLOP	Reduces due to 3D compute	Increasing degrees of In-memory compute			
	Response time	MAC/ns	500MAC/ns				
	Efficiency	~10-50pJ/MAC	~10-50fJ/MAC				
	Benchmarks	graph500;imagenet;maxcut(100)	NFLscheduling	(piplib)++;'videonet';maxcut(1e6)	(IQ-tests)v2.0	Max-cut(1e10);	
qualitative	Self-autonomy	Self-monitor + oversight	Understands its state	Interacts with environment	Learns new concepts	Delivers directed goals	Fully independent, self-aware
	Agility	Recovery of components	Recovery overall	Self-support	Self-healing	Self-evolving	Self-reproducing
	Compliance	none	Component-wise security	Component-wise regulatory	Overall security	Regulatory/security	Global compliance
Software, applications	I/F	Chatbots	Digital Twins	Brain-computer sensing I/F	Brain-computer actuate I/F	Brain-computer decision I/F	Thinking interfaces
	Use Cases	Assisted transportation	Robotic Process Automation	Shared Cognition Security	Self-driving cars	Autonomous machines	Independent intelligence
	Applications	ML, data analytics	Knowledge mgmt.	Machine intelligence	Auto-ML	+dynamic numerical apps	Brain-computer AI I/F
	Data structures	Graphs	Dynamic graphs	Irregular graphs	Time varying graphs	Irregular time-varying graphs	Free forming data structures
	Algorithms	Inference	Learning, NP (QUBO)	Reasoning, NP (MIP)	Decision making	Real-time enhancements	Self-modifying
	AI	I/F for video, language	Goal-directed knowledge base	Informed learning system	Inference engine for goals	Optimized, real-time	Evolutionary
	System Software	Parallelism, Distribution, Scale	Heterogeneity	AI inside of OS	Distributed AI enhanced	Heterogeneous AI	Global OS for AI
Hardware	Architecture	Entering exascale	Heterogeneity	Embrace extreme heterogeneity	New types of accelerators	New AI architecture	Brain-mimicking
	Compute	Tradional CPU/GPU/ASIC/FPGA	Approximate	Analogue	Stochastic	Chaotic	Wetware
	Interconnect	Silicon photonics; Datacom	DWDM Silicon Photonics	Plasmonics	CPU-mem photonics	AI-driven interconnect	On-chip optics-computercom
	Memory	3D-Xpoint	PCM	STT-RAM	ReRAM	ReRAM Compute in Memory	DNA
	Technology	CMOS		Neuromorphic Quantum, Photonics Ising machines, memristor Hopfield Neural Nets			

training → transfer learning → awareness, cognition → self-learning → understanding →

AI → +Understanding capability →

On the Limits of Stochastic Computing

Back to Contents

Florian Neugebauer
Institute of Computer Architecture and Computer Engineering
University of Stuttgart
Stuttgart, Germany
florian.neugebauer@iti.uni-stuttgart.de

Ilia Polian
Institute of Computer Architecture and Computer Engineering
University of Stuttgart
Stuttgart, Germany
ilia.polian@iti.uni-stuttgart.de

John P. Hayes
Computer Engineering Laboratory
University of Michigan,
Ann Arbor, USA
jhayes@umich.edu

***Abstract*—Stochastic computing (SC) provides large benefits in area and power consumption at the cost of limited computational accuracy. For this reason, it has been proposed for computation intensive applications that can tolerate approximate results such as neural networks (NNs) and digital filters. Most system implementations employing SC are referred to as stochastic circuits, even though they can have vastly different properties and limitations. In this work, we propose a distinction between strongly and weakly stochastic circuits, which provide different options and trade-offs for implementing SC operations. On this basis, we investigate some fundamental theoretical and practical limits of SC that have not been considered before. In particular, we analyze the limits of stochastic addition and show via the example of a convolutional NN that these limits can restrict the viability of strongly stochastic systems. We further show that theoretically all non-affine functions do not have exact SC implementations and investigate the practical implications of this discovery.**

Keywords—Emerging technologies, error analysis, simulation, stochastic computing.

I. Introduction

Stochastic computing (SC) is considered a promising alternative to conventional binary computing for a number of applications such as image processing [2][11], digital signal processing [6][21] and neural networks [3][19]. Other applications include vector quantization [20], Bayesian sensor fusion [7] and low-density parity codes [14] in Wi-Fi standards. SC was originally proposed by Gaines and Poppelbaum in the 1960s [8], and has gained traction again in recent years due to an increasing demand for small, low-power circuits. This demand is driven by embedded devices and IOT applications that include sensor networks with tiny power sources and very tight requirements for circuit area.

The main advantage of SC lies in compact implementations of basic arithmetic functions like addition and multiplication, which are performed on stochastic numbers (SNs), the basic data representation form of SC. This number representation allows for very efficient implementations of these functions, among others. An often cited example in this regard is multiplication, which can be implemented using a single AND or XNOR operation, depending on the SN format employed. Addition only requires a single 2-to-1 multiplexer (MUX). Both of these operations can be implemented with significantly less hardware than their binary counterparts or even approximate binary versions. Neural networks (NNs) are an application that can potentially benefit strongly from SC because they employ huge numbers of these individually simple operations.

Another major advantage of SC is its inherent error tolerance. Every bit of an SN has identical significance, whereas the significance of bits in a binary number varies considerably. This property makes SNs and SC in general very resistant to soft errors, as a bit flip only causes a minor change in the value of an SN, whereas it can cause a major error in a binary (base 2) number, if a bit with high significance flips. This property also makes SC especially suitable for applications that regularly have to deal with noisy data, such as image and signal processing.

In contrast to the deterministic nature of conventional binary computing, SC employs a probabilistic approach. SNs are generated using random number generators (RNGs) and as a result, computations performed by stochastic circuits are in general not exact. Sources of inaccuracy in SC include correlation [5][18] and approximation errors of the target function [1], among others. Efforts have been made to analyze and manage these inaccuracies [5][16], which have improved the understanding of the underlying mathematical model of SC.

In this work, we address a central point that has so far received little attention: What are the basic computational limits of SC? We will approach this question from both the theoretical and the practical viewpoint. Building upon the aforementioned works on modelling SC, we will show that a large number of arithmetic functions cannot be implemented precisely in SC, but will always show a small bias. Surprisingly, this includes simple functions such as $f(x) = x^2$ that had previously been considered free of such bias. We will demonstrate that this is a fundamental property of SC and cannot be eliminated by modifying the circuit. We will further analyze the practical impact of this bias through simulation.

A second major limiting factor of SC that will be examined concerns addition. Computations in SC are normally limited to the range of values between -1 and 1 due to their probabilistic nature. Therefore, addition of k SNs implies downscaling of the result by a factor of k to stay within this interval. This can cause problems when many SNs with mostly small absolute values are added together. In this case, a high scaling factor can cause the result to be indistinguishable from regular noise. This turns out to be a major issue in NNs, as they often include additions of hundreds of values.

While recent work on SC mostly follows the same theoretical approach, their specific practical implementations can differ significantly. The terms "stochastic circuit" and "stochastic implementation" are used for a wide range of different systems, from small image processing systems that only include a few gates to comparatively large NN systems composed of stochastic and binary elements. Even though these systems have vastly different properties and limitations, they are commonly described in the same terms. We propose a definition to distinguish between certain types of SC systems that are subject to different computational limitations.

II. SC Background and SC Type Definition

A. Introduction to SC and SNGs

Stochastic computing is based on operations on bit streams of variable length referred to as stochastic numbers (SNs).

978-1-7281-5221-9/19/$31.00 ©2019 IEEE

Different representational forms for SNs exist; the two most common being unipolar and bipolar representations. A unipolar SN of length n with n_0 0s and n_1 1s represents the value n_1/n, while a bipolar SN represents the value $(n_1 - n_0)/n$. For example, the bit stream 01000110 has the unipolar value 0.375 and the bipolar value -0.25. The basic value range for unipolar SNs is therefore the unit interval $[0, 1]$, which is extended to $[-1, 1]$ with the bipolar representation. The trade-off for this extension is a reduced precision in bipolar representation of $p_b = 2/n$ compared to $p_u = 1/n$ in unipolar case. Here, precision is defined by the smallest representable non-zero value.

Its unusual number representation is the reason for the inherent error tolerance of SC. Any single bit flip changes the value of an SN only by p_b or p_u, depending on the form of the affected SN, as each bit has the same significance. Further SN representation methods exist, such as extended stochastic logic [4] and inverse bipolar representation [1], but they are not commonly used.

SNs are generated using stochastic number generators (SNGs). The most common type of SNG (shown in Fig. 1) consists of a pseudorandom number generator (PRNG) and a comparator. In each clock cycle, the PRNG provides a new pseudorandom binary number $R \in [0, 1]$, which is compared to the constant binary input value $B \in [0, 1]$. If $R < B$, the SNG outputs a 1, and it outputs a 0 otherwise. After n clock cycles, the SNG will have output $n \cdot B$ 1s on average and will have therefore created a unipolar SN which approximates B. In order to generate a bipolar SN, the constant input is set to $\hat{B} = (B + 1)/2$. Most often, a linear feedback shift register (LFSR) is used as the pseudorandom number source. LFSRs are small compared to the hardware implementations of other PRNGs and can reliably create an SN with the exact value B, provided that the sequence length of the LFSR is equal to the SN length n, and B is a multiple of $1/n$. If B is not a multiple of $1/n$, the LFSR-based SNG creates an SN with the closest possible approximation to B.

Fig. 2 shows some examples of basic arithmetic operations on stochastic numbers. Multiplication in the unipolar format is performed by an AND gate; for the bipolar format an XNOR gate is used. The probability for an output bit z_i to be 1 is then equal to the product of the probabilities of input bits a_i and b_i being 1. In an SN, the probability of each single bit being 1 is equal to the value of the SN and thus the AND gate multiplies its two input SNs.

Addition is performed by a MUX in both the unipolar and bipolar forms, however it introduces a scaling factor that can be controlled by the selection input S of the MUX. S can be considered a unipolar SN with value s so the MUX then computes $a \cdot s + b \cdot (1 - s)$. For regular addition, s is set to 0.5, but different values can be used to perform multiply-accumulate operations efficiently. This scaling is inherent to SC, as the range of values is bound to $[0, 1]$ or $[-1, 1]$ depending on the representation form, and the result of an addition has to lie within this range.

B. Types of Stochastic Circuits

Most non-trivial SC systems consist of several connected components, each of which usually implements some basic arithmetic operation. For example, stochastic digital filter circuits such as in [21] commonly consist of multipliers to

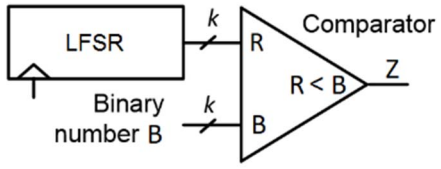


Fig. 1: Stochastic number generator (SNG) with an LFSR as the pseudorandom number source.

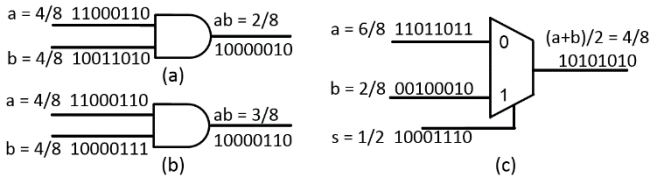


Fig. 2: Typical SC operations: a) accurate unipolar multiplication, b) inaccurate unipolar multiplication, c) scaled addition.

calculate products of filter coefficients and input values, and scaled adders to calculate the average of these products. Stochastic NNs are layered networks of multipliers, adders, circuits for pooling operations (usually maximum operations) and circuits for activation functions, such as *tanh* [3] or *ReLU*. While the overall structure of these systems is often determined by their application, the inner structure of the stochastic circuits for each component can be changed arbitrarily, as long as the component still implements its target function. Therefore, two stochastic systems with identical overall structure can have vastly different properties and limitations depending on the implementation details of their components.

We propose to split stochastic circuits into two groups with significantly different limitations. Additional variations between circuits within each group exist, but are of minor impact compared to the inter-group differences. We define these groups in the following way:

- **Strongly stochastic circuits**: Basic strongly stochastic circuits receive only SNs as inputs and compute a single basic arithmetic operation with only SNs as outputs. A strongly stochastic circuit consists exclusively of basic strongly stochastic circuits. Examples for this group can be found in image detection [11] and digital filter applications [21].
- **Weakly stochastic circuits**: Basic weakly stochastic circuits receive only SNs as inputs and compute a single basic arithmetic operation with at least one binary output, or vice-versa. A weakly stochastic circuit includes at least one basic weakly stochastic circuit. An example of this type is the APC adder in [19].

Several major differences exist between these groups. Strongly stochastic circuits usually have implementations that are more compact and have high tolerance of noisy input data. However, they are subject to the limitations of SC regarding the possible range of input and output values, as they can only lie within the interval $[-1, 1]$. Internally, values outside of this range can be used, for example in counters that track internal information, such as the state counter in the FSM of an activation function [3] or the counters in the maximum circuit in [15]. Stochastic adders of this group always include

scaling, which can cause problems in addition-heavy applications, as will be shown in section IV.

The major advantage of weakly stochastic circuits is the absence of the restriction to the range $[-1, 1]$ in either their inputs or outputs. This advantage is most noticeable for adders, as weakly stochastic adders can have binary outputs larger than 1, which removes the need for scaling. They are therefore a good candidate in systems with large sums, such as convolutional NNs. However, their binary elements often require more area than most strongly stochastic circuits. In addition, weakly stochastic circuits may lose tolerance to noise, if some of the inputs are in binary format.

III. Presence of Bias

A. Theoretical Bias in SC

The central concept of SC is the computation of a target function f via probabilistic methods. Input variables of f are transferred from their original deterministic domain, in general the set of real numbers, into random variables. Formally, this can be described as transforming the real valued vector of input variables $x = (x_1, \dots, x_n)$ into a vector $X = (X_1, \dots, X_n)$ of random variables with $x_i = \mathrm{E}(X_i)$ for all its elements, where E denotes the expectation operator. In the following sections, we will refer to the value $f(x)$ as the "exact" result of a computation, which can for example be obtained using conventional floating-point arithmetic. In previous work on SC, it is generally assumed that the transformation of the inputs from real values to random variables does not affect the behavior of the target function, so that

$$f(x) = \mathrm{E}\big(f(X)\big) = f\big(\mathrm{E}(X)\big) \tag{1}$$

holds after this transformation. However, this is generally not the case. Jensen's inequality states that for any convex function

$$\mathrm{E}\big(f(X)\big) \geq f\big(\mathrm{E}(X)\big) \tag{2}$$

holds (the converse relation holds if f is concave). In fact, equality in (2) is valid if f is affine, i.e., its mapping has to satisfy additivity and homogeneity, or X is constant. For example, the MUX-based scaled addition $g(a, b) = \frac{a+b}{2}$ is an affine function, as it satisfies the two requirements:

Additivity:

$$g\big((a, b) + (c, d)\big) = g(a + c, b + d) = \frac{a + c + b + d}{2}$$
$$= \frac{a + b}{2} + \frac{c + d}{2} = g(a, b) + g(c, d)$$

Homogeneity:

$$g\big(c \cdot (a, b)\big) = g(ca, cb) = \frac{ca + cb}{2} = c \cdot g(a, b)$$

On the other hand, the function $h(x) = x^2$ is not affine, as it does not meet the requirements. For example:

$$h(x + y) = (x + y)^2 \neq x^2 + y^2 = h(x) + h(y)$$

An affine function preserves points, straight lines and planes, which can often be a useful criterion to quickly categorize a function by looking at its graph. It can be seen easily, for example, that $h(x)$ does not preserve straight lines.

As SNs are not generally constants, only the requirement of f being affine can be met in SC in order to satisfy (2). This in turn implies that even simple non-affine functions such as $f(x) = x^2$ cannot be implemented precisely in SC and instead will result in a biased stochastic implementation.

The theoretically expected value of a stochastic implementation of f can be calculated by

$$\mathrm{E}(f(X)) = \int_{-\infty}^{\infty} f(x)\xi(x)\mathrm{d}x \tag{3}$$

where $\xi(x)$ is the density function of X. It has been shown in [16] that SNs can be modelled as normally distributed random variables. For example, for the function $f(x) = x^2$ and a bipolar SN of length 256 with expected value 0.5 as input, the expected value of f is 0.2539, which implies a bias of 0.0039. This may not seem significant compared to the normal inaccuracies of SNs at first, as the standard deviation σ of an SN with the same parameters is 0.054. However, this bias has two key distinguishing properties:

1. While other inaccuracies in SC such as rounding errors or random fluctuations are distributed equally around the mean, the bias here is always positive and can therefore accumulate in subsequent operations like addition where other types of errors cannot.
2. The size of the bias depends on the target function. For example, it is shown in [15] that the maximum function has relatively large bias. For the maximum of four bipolar SNs of length 256, the bias can be as high as 0.056 (if all SNs have expected value 0).

Strongly stochastic circuits are always subject to the forgoing bias, regardless of their internal structure. This is a direct consequence of the probabilistic nature of SNs. The value of the bias varies for different input values, as it generally depends on the mean and standard deviation of the input SNs. It is therefore not feasible to counteract the bias by measures such as introducing additional constants in a circuit.

B. Practical effect of Bias

In order to assess the practical effect of the bias, an accurate model for the distribution of LFSR-generated SNs has to be found first. Commonly, individual bits of SNs are considered to be independent of each other and are treated as Bernoulli random variables. Dependencies within an SN (autocorrelation) introduced by the sequential nature of LFSRs are usually ignored. However, it has been shown that they can lead to systematic errors in some circuits [17]. Furthermore, while the theoretical model from [16] is easy to use, it does not consider LFSR-generated SNs, which generally have slightly smaller standard deviation than the model assumes and it can therefore overestimate the bias.

In the following, we will analyze the distribution of these SNs at different circuit levels and the bias they cause in various functions. Table 1 shows the average standard deviations of output SNs after specific operations have been performed on LFSR-generated input SNs. Each operation was simulated on 100 random input values, each of them simulated independently 5,000 times with an SN length of 256. It can be seen that after only one operation, the distributions of the SNs are already close to the distributions of SNs generated with a highly accurate, software-based RNG of the Mersenne

Table 1: Standard deviation σ of LFSR-based and MT-based SNs after performing various SC operations.

Operation	LFSR-based σ	MT-based σ
Multiplication	0.034	0.057
Squaring	0.047	0.053
Addition	0.049	0.057
Stanh($4x$) [3]	0.080	0.078

Twister (MT) type [13]. Therefore, the bias described in the previous section is also present in circuits with LFSR-generated input SNs.

According to (3), the bias of a stochastic circuit depends on the implemented function and the distribution of its input values. We performed simulations on several stochastic circuits affected by it (i.e., implementing non-affine functions) to determine their bias. The results of these simulations together with the theoretical analysis using (3) show that fortunately, most typical SC functions have a very small bias that is not expected to impact practical implementations in a significant way. The only exception is the maximum function, which shows a comparatively large bias that can lead to noticeable inaccuracies in applications, as we will show in section IV.A.

Table 2 shows some samples from our simulations, which consisted of 50,000 independent repetitions for each operation with SNs of length 256. As many complex non-affine functions used so far (e.g., *tanh*) do not have directly corresponding SC implementations, the affected functions are currently restricted to polynomials of degree two or more and the maximum function, for which a corresponding stochastic circuit exists [15]. We have further noticed that decorrelation techniques such as isolation [5] reduce, or even remove the bias for correlation-affected functions such as squaring (simulations in table 2 were performed with regeneration instead of isolation). At this point, we cannot explain this effect, it will be investigated in future work.

In summary, our analysis shows that exact SC implementations are limited to affine functions in theory. However, the extent of the bias caused by non-affine functions is very small with the maximum function being the only exception that we found. This may also be the reason why this limitation has so far not been recognized, as many proposed SC applications, e.g., in image processing, either implement affine functions or are only affected by a very small bias that can be easily missed due to the inherent random behavior of stochastic circuits. Nevertheless, SC designs that include an affected function should be examined with regard to their bias.

Table 2: Bias of various functions; here P1 = $(x^2 + 0.5x)/2$ and P2 = $(x^3 - 0.4x^2 + 0.3x - 0.8)/4$.

Operation	Simulated bias	Theoretical bias
Squaring	0.0038	0.0039
Cubing	0.0044	0.0044
Maximum	0.0643	0.056
P1	0.0019	0.0020
P2	0.0008	0.0008

This can be done during the design process either by calculating (3) directly or by simulation. If the bias exceeds the acceptable level for the affected application, SN length can be increased to reduce it.

IV. Limits of Stochastic Addition

A. Strongly Stochastic Addition in Neural Networks

A major limiting factor of strongly stochastic circuits is scaled addition. Especially in applications with large sums, such as digital filters and NNs, the scaling often causes a significant reduction in the absolute value of the result. This in turn results in more noise in the output SNs. Moreover, it becomes much harder to compare results of such large additions directly with each other in many cases. For example, two bipolar SNs X_1 and X_2 of length 100 bits and values 0.5 and 0.3, respectively, are clearly distinguishable in most cases, as their standard deviations are less than 0.1. However, two bipolar SNs $\widehat{X_1}$ and $\widehat{X_2}$ of the same length with values 0.05 and 0.03 cannot be reliably distinguished anymore, even though the relative difference is identical in both cases.

NNs are especially affected by this problem. Even relatively small NNs can already include additions of several hundred values. At the same time, the relation between values is important in these networks, e.g., in the processing of feature maps in convolutional NNs or in the final classification decision, where often the position of the largest output is the deciding factor instead of the specific output value. Moreover, convolutional NNs often operate on sparse matrices internally with most values equal or close to 0 and only a few significantly larger entries. Stochastic additions with large scaling factors can in this case lead to results that are indistinguishable from 0 in the context of SC and thereby cause a major loss of information in the network.

We have performed an analysis of a typical computation of a convolutional NN using the LeNet structure, which is often used for smaller image recognition tasks and has been the basis of a previous SC implementation [19]. The task of

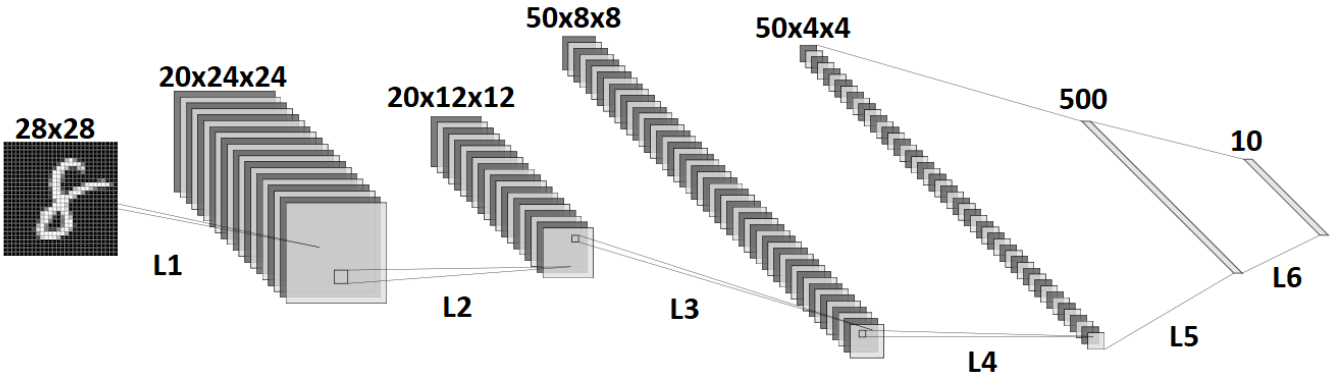


Figure 3: Structure of the convolutional NN used in simulations. Numbers above the layers describe the data size.

Table 3: Implementation details of the NN in Fig. 3.

Layer ID	L1	L2	L3	L4	L5	L6
Layer type	Convolution + activation	Max-pooling	Convolution + activation	Max-pooling	Fully connected	Fully connected
Filter size	5x5	2x2	5x5	2x2	–	–
Implementation	Strongly stochastic convolution and $\tanh(11x)$ activation after [3] and [19]	Strongly stochastic after [15]	Strongly stochastic convolution and $\tanh(17x)$ activation after [3] and [19]	Strongly stochastic after [15]	Strongly stochastic multiplication, weakly stochastic addition (parallel counter)	Fully binary
Distrib. analysis	Figures 5 + 6	Figure 7	Figures 8 + 9	–	–	–

the NN is handwritten image recognition using the MNIST [10] dataset, which is often used as a standard benchmark to evaluate an NN. The details of the stochastic implementation are given in Fig. 3 and table 3. The SN length used was 4,096. The reference implementation is a conventional floating-point software-based network (SBN) with input values restricted to $[-1, 1]$ and classification accuracy 98.26% that provides us with exact reference values.

The goal of our analysis here is to examine the distributions of output values of each layer in the network and compare the distributions of the SBN with those of a partially stochastic implementation. In layers with very narrow SN value distributions, the stochastic network potentially loses a large amount of information, as the random fluctuations gain significance, and small differences between values that are present in the SBN are lost because of this. This information loss should then be visible in the output value distribution of the subsequent layer.

The following distributions were obtained during classification of the input image showing the digit "8" in Fig. 3 as a 28×28 array of grayscale pixels. The input values denote pixel intensity and range from 0.00 (black) to 1.00 (white). The initial distribution of input values is shown in Fig. 4. The width of the histogram intervals has been set to 2σ of an SN with mean 0 and length 4,096, which means that random fluctuations of SNs within the same interval are very likely to dominate over the actually important difference of their mean values.

The vast majority of input values lie in the interval $[0, \frac{1}{16}]$ because the input images have a black background (encoded by input value 0.00). As explained previously, this can cause problems in the first convolutional layer (before activation),

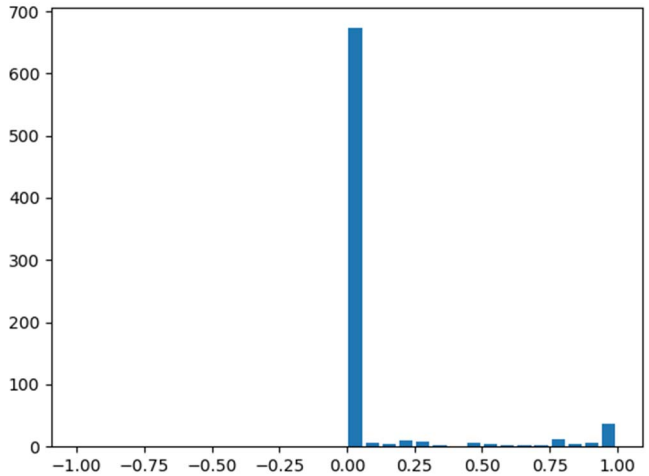


Figure 4: Distribution of input values of the simulated NN.

as the significant values are downscaled during the addition with small values. Indeed, the distribution of the output values of the convolution operation in L1 (Fig. 5) confirms this: Almost all of the output values in the network after the first convolutional operation are concentrated in $[-\frac{1}{16}, \frac{1}{16}]$. As noted before, important differences within each interval cannot be reliably tracked and conserved by the stochastic implementation. This becomes especially clear in the output distribution of the subsequent activation function in the same layer (Fig. 6), which is implemented by $\tanh(11x)$ according to the formula in [19].

Upscaling is commonly used in the activation function in SC implementations of NNs to counteract the downscaling of the addition, however it also magnifies the random fluctuations. It can be seen in Fig. 6 that the distributions of the SBN and the SC implementations differ significantly after the activation part of L1. In fact, the distribution of the SC implementation resembles a normal distribution with mean 0, which suggests that not all information of the reference SBN is retained by the SNs at this point in the network. The subsequent max-pooling layer L2 only exacerbates this problem, as can be seen in Fig. 7.

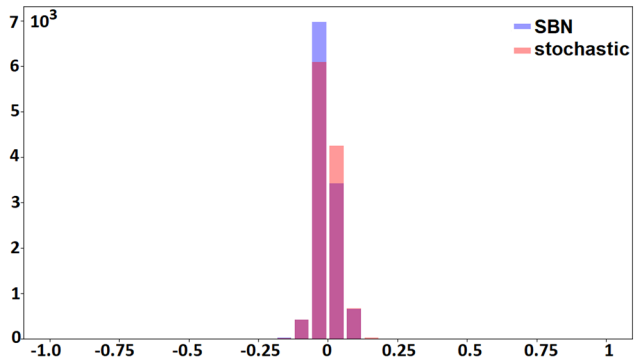


Figure 5: Distribution of output values of the convolution operation in L1.

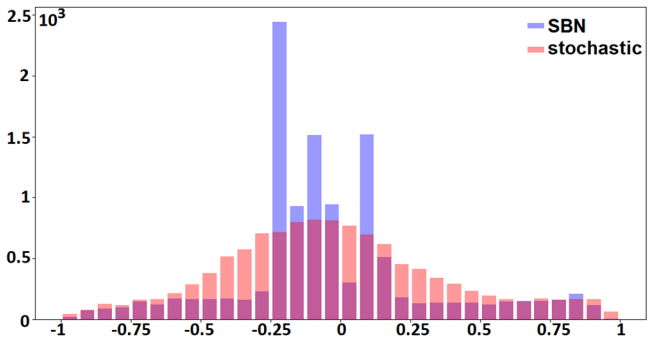


Figure 6: Distribution of output values of L1 (convolution and activation function).

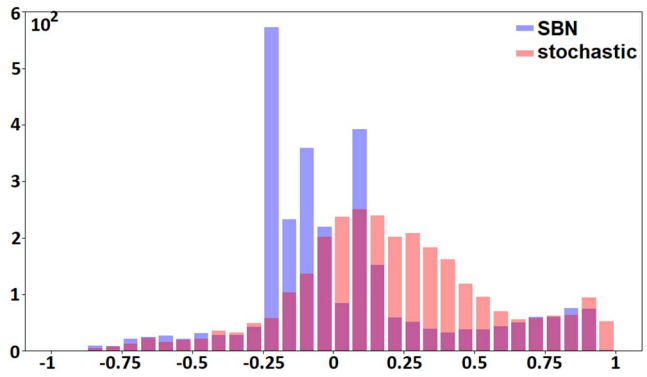


Figure 7: Distribution of output values of L2.

Because the input distribution of the stochastic max-pooling layer resembles a normal distribution, the mean of the output distribution is shifted to the positive side, as the four inputs of each maximum operation likely contain at least one positive value. Moreover, the bias effect described in section III causes a further slight shift in the same direction. Overall, it is clear that much of the information that the exact SBN implementation contains is lost at this point in the SC implementation.

The second set of convolution, activation function and max-pooling operations increases this effect even more, to the point that the overall output of the stochastic network becomes essentially random. Because of the downscaling effect of the second convolutional layer, its outputs lie exclusively in the interval $[-\frac{1}{16}, \frac{1}{16}]$ (Fig. 8). While the overall distributions of the SC implementation and the SBN seem to be very similar for L3, L4 and L5, detailed analysis reveals that the specific patterns in the feature maps of L3 are not well conserved in the SC implementation. In some cases, such as the bottom example in Fig. 9, the basic pattern in the feature map of the SBN (left) can still be partially recognized in the SC counterpart. However, in many cases, such as the top example in Fig. 9, the pattern is lost and with it the information that the SBN contains.

A convolutional NN with the structure given in Fig. 3 includes additions of 26 values in the first convolutional layer and additions of 501 values in the second convolutional layer. Subsequent fully connected layers include even larger sums. In our simulations of an SBN with unscaled addition, the largest absolute output value of the first convolutional operation was ≈ 5.6 and after the second convolutional operation ≈ 20.8 when the network was subject to the restrictions of SC (e.g., requiring all input values to be in $[-1, 1]$).

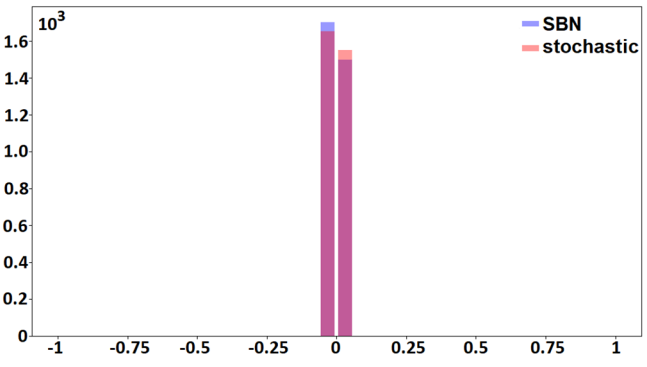


Figure 8: Distribution of output values of L3.

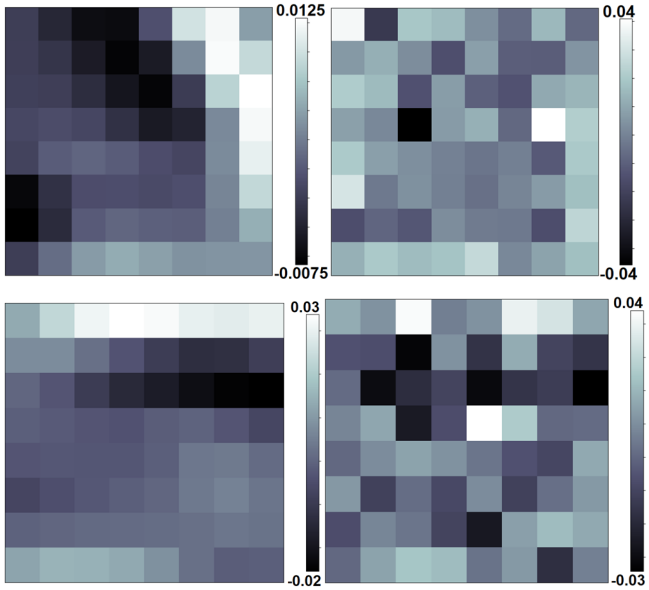


Figure 9: Examples of feature maps in L3: SBN (left) vs SC implementation (right).

The corresponding average values were ≈ 0.1 and ≈ 0.6. The respective maximum values in a strongly stochastic implementation would therefore be ≈ 0.215 and ≈ 0.042, and the average values ≈ 0.004 and ≈ 0.001. This results in a loss of most relevant information at the second convolutional layer for an SN length of 4,096, as the corresponding standard deviation of an SN with this length is ≈ 0.016 and therefore is significantly larger than the average value at that point in the network.

While it is in theory possible to counteract the accuracy loss due to scaling by increasing the SN length, this is not practically feasible for the high scaling factors commonly appearing in NNs. The SN length has to be increased in proportion to the scaling factor in order to retain the pre-addition precision. However, in NNs such as the one analyzed in this section, multiple subsequent layers include scaled addition and their scaling factors are therefore multiplied. SN lengths of magnitude 10^5 or even 10^6 and more (depending on the exact network structure) would be needed in this case. For example, the specific network of this section still only reached a classification accuracy of $\approx 70\%$ when SNs of length 2^{17} were used in our simulations. As the computation time in SC depends directly on the SN length, this leads to unacceptably long computations.

B. Strongly Stochastic Addition in Digital Filters

Digital filters have been considered a promising field for stochastic implementations [21] because they also consist mainly of multiplications and additions. In a finite impulse response (FIR) filter for example, input signals are multiplied with filter coefficients followed by the calculation of the average of the products. Strongly stochastic adders scale down the result by the number of inputs and so are inherently calculating an average. Existing stochastic implementations of FIR filters [9][21] are therefore indeed very compact. However, extensive simulations performed by the authors of [21] show that a very high SN length is needed for the stochastic filters to reach acceptable accuracy. These authors state that in order to reach the accuracy of an exact 267-tap binary FIR filter (corresponding to an average of 268 values) with a precision of 13 bits, an SN length of 2^{22} is needed. A

"possibly still useful degraded performance" ([9]) can be achieved with an SN length of 2^{18}.

Further implementations of a 31-tap FIR filter (corresponding to an average of 32 values) in [17] use an SN length of 2^{16} to produce results comparable to the exact reference filter. The results reported in the cited works show that strongly stochastic filter implementations suffer from the same effects as strongly stochastic NNs: large downscaling factors lead to highly degraded accuracy. If the input signal of a filter is not constant, its averaging operation leads to a systematic downscaling of the filter's output compared to the original input. As a result, these outputs tend to be closer to the center of the SN value range ($[-1, 1]$ for bipolar and $[0, 1]$ for unipolar SNs), where their standard deviation is the largest. This combination of decreasing the absolute value while increasing the inaccuracies of the output leads to the need for excessive SN lengths in this type of application.

C. Weakly Stochastic Adders

A possible solution to the problem of reduced accuracy due to high downscaling factors is provided by weakly stochastic adders. In contrast to the bias limitation shown in section III, the problems due to high scaling factors in NNs are known, and many proposed stochastic NN implementations therefore employ weakly stochastic adders. These adders can output binary numbers and are therefore not subject to the limited range of SNs. An implementation of a weakly stochastic adder, called an approximate parallel counter (APC) is given in [19]. It receives 16 SNs as inputs, approximately counts the number of 1s received per clock cycle using conventional full adders and outputs the result as a 4-bit binary number. By accumulating the outputs over the full SN length n, all inputs are summed up without scaling the result.

The use of weakly stochastic circuits has several disadvantages however: The error tolerance of the circuit is worse than that of strongly stochastic implementations, as errors in its binary part can affect the output significantly. Secondly, the circuit cannot be connected to a subsequent circuit with stochastic inputs without additional binary-stochastic conversion elements, which are expensive and cause significant area overhead. In [19] this is avoided by combining the adder with a subsequent weakly stochastic circuit for the *tanh* function, which receives binary inputs and produces a stochastic output.

In applications with unavoidable, large-scale additions, weakly stochastic adders can provide a good solution to the scaling problem. The experimental data reported in [19] suggests that a convolutional NN based on weakly stochastic adders performs much better than an implementation using strongly stochastic MUX-based adders. The authors of [19] have repeated their simulations for the MUX-based CNN configurations on a seemingly identical network in [12], however with significantly different results, e.g., 3.06% inaccuracy in [19] versus 8.68% inaccuracy in [12], both for SN length 1024. We were not able to reproduce the results reported in those works for the MUX-based network configurations and our detailed analysis in section IVA. implies that such an accuracy is impossible to reach for a MUX-based implementation of this specific CNN configuration with SN length 1024. Our results suggest that the improvement offered by weakly stochastic designs such as APC is even more significant and can, with some trade-off in area, provide a solution for large-scale additions in SC.

V. Conclusion

We proposed to classify stochastic circuits as strong or weak, in order to distinguish between significantly different SC implementations of the same systems. These two types have different properties with regard to value range limitations, error tolerance and area consumption and offer a way to further categorize stochastic circuits according to these properties. We have further shown that stochastic implementations of non-affine functions suffer from a theoretically provable bias due to the probabilistic nature of SC. Our simulations confirm that this bias leads to noticeable inaccuracies in some applications such as convolutional NNs, which are considered a promising field for SC. However, we have also found that the effect of this bias is very small and appears to be negligible in the majority of cases. We therefore conclude that it is not a limiting factor for SC, in general.

Stochastic addition poses a more serious challenge in SC, however. Strongly stochastic circuits include an unavoidable downscaling factor that can significantly degrade the accuracy of the result, as we have demonstrated for the example of a CNN. With a trade-off in area, weakly stochastic adders with stochastic inputs and binary outputs can provide a solution for these applications and make their SC implementation practical.

Other SC applications such as image processing algorithms like gamma correction and edge detection are largely unaffected by the examined limitations, as they only include additions with very small scaling factors, so the implemented functions show little or no bias. These applications can therefore fully benefit from the high error tolerance and low hardware cost that strongly stochastic circuits offer.

Acknowledgment

This work was supported in part by Deutsche Forschungsgemeinschaft (DFG), project number PO 1220/12-1.

References

[1] A. Alaghi and J.P. Hayes. STRAUSS: Spectral transform use in stochastic circuit synthesis. *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems,* 34: 1770-1783, 2015.

[2] A. Alaghi, C. Li and J.P. Hayes. Stochastic circuits for real-time image-processing applications. *Proc. of Design Automation Conference*, article 236, 2013.

[3] B.D. Brown and H.C. Card. Stochastic neural computation I: Computational elements. *IEEE Transactions on Computers*, 50: 891-905, 2001.

[4] V. Canals, A. Morro, A. Oliver, M. L. Alomar and J. L. Rosselló. A new stochastic computing methodology for efficient neural network implementation. *IEEE Transactions on Neural Networks and Learning Systems*, 27: 551-564, 2016.

[5] T.-H. Chen and J. P. Hayes. Analyzing and controlling accuracy in stochastic circuits. *Proc. of IEEE International Conference on Computer Design,* 367–373, 2014.

[6] J. Chen and T. Hu. A novel FIR filter based on stochastic logic. *Proc. of IEEE International Symposium on Circuits and Systems*, 2050 - 2053, 2013.

[7] A. Coninx, P. Bessière, E. Mazer, J. Droulez, R. Laurent, M. A. Aslam and J. Lobo. Bayesian sensor fusion with fast and low power stochastic circuits. *Proc. of IEEE International Conference on Rebooting Computing (ICRC)*, 1-8, 2016

[8] B.R. Gaines. Stochastic computing systems. *Advances in Information Systems Science*, 2: 37-172, 1969.

[9] H. Ichihara, T. Sugino, S. Ishii, T. Iwagaki and T. Inoue. Compact and accurate digital filters based on stochastic computing. *IEEE Transactions on Emerging Topics in Computing*, 7: 31-43, 2019.

[10] Y. LeCun, C. Cortes and C. Burges. The MNIST Database. http://yann.lecun.com/exdb/mnist/

[11] P. Li and D.J. Lilja. Using stochastic computing to implement digital image processing algorithms. *Proc. of IEEE International Conference on Computer Design,* 154-161, 2011.

[12] J. Li, A. Ren, Z. Li, C. Ding, B. Yuan, Q. Qiu and Y. Wang. Towards acceleration of deep convolutional neural networks using stochastic computing. *Proc. Asia and South Pacific Design Automation Conference*, 115-120, 2017.

[13] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 8.1: 3-30, 1998.

[14] A. Naderi, S. Mannor, M. Sawan and W. J. Gross. Delayed stochastic decoding of LDPC codes. *IEEE Transactions on Signal Processing*, 59: 5617-5626, 2011.

[15] F. Neugebauer, I. Polian and J.P. Hayes. On the maximum function in stochastic computing. *Proc. of ACM International Conference on Computing Frontiers,* 59-66, 2019.

[16] F. Neugebauer, I. Polian and J.P. Hayes. Framework for quantifying and managing accuracy in stochastic circuit design. *Proc. of Design, Automation and Test in Europe Conf*., 31-36, 2017.

[17] F. Neugebauer, I. Polian and J.P. Hayes. S-box-based random number generation for stochastic computing. *Microprocessors and Microsystems*, 61: 316-326, 2018.

[18] M. Parhi, M.D. Riedel and K.K. Parhi. Effect of bit-level correlation in stochastic computing. *Proc. of IEEE International Conf. on Digital Signal Processing*, 465-467, 2015.

[19] A. Ren, J. Li, Z. Li, C. Ding, X. Qian, Q. Qiu, B. Yuan and Y. Wang. SC-DCNN: Highly scalable deep convolutional neural network using stochastic computing. *ACM SIGOPS Operating Systems Review,* 51: 405-418, 2017.

[20] R. Wang, J. Han, B.F. Cockburn and D.G. Elliott. Stochastic circuit design and performance evaluation of vector quantization for different error measures. *IEEE Transactions on VLSI Systems,* 24: 3169-3183, 2016.

[21] R. Wang, J. Han, B.F. Cockburn and D.G. Elliott. Design, evaluation and fault-tolerance analysis of stochastic FIR filters. *Microelectronics Reliability*, 57: 111-127, 2016.

Design of a 16-bit Adiabatic Microprocessor

Rene Celis-Cordova[1], Alexei O. Orlov[1], Tian Lu[2], Jason M. Kulick[2], and Gregory L. Snider[1]
[1]Electrical Engineering Department, University of Notre Dame
275 Fitzpatrick Hall, Notre Dame, IN 46556, USA.
[2]Indiana Integrated Circuits
1400 E Angela Blvd, South Bend, IN 46617, USA.

***Abstract*—Heat production is one of the main limiting factors in modern computing. In this paper, we explore adiabatic reversible logic which can dramatically reduce energy dissipation and is a viable implementation of future energy-efficient computing. We present a 16-bit adiabatic microprocessor with a multicycle MIPS architecture designed in 90nm technology. The adiabatic circuits are implemented using split-rail charge recovery logic, which allows the same circuit to be operated both in adiabatic mode and in standard CMOS mode. Simulations of a shift register show that energy dissipation can be much lower when operating in adiabatic mode compared to its CMOS counterpart. We present a standard cell library with all the necessary components to build adiabatic circuits and implement the subsystems of the microprocessor. The microprocessor has a proposed operating frequency of 0.5 GHz representing a useful implementation of adiabatic reversible computing.**

Keywords— adiabatic logic, reversible computing, microprocessor, CMOS integrated circuits

I. Introduction

Modern microprocessors are limited by heat dissipation. Speeds have been capped around 4 GHz to limit heat generation since 2004 [1]. Such speeds are well below the RC time constant limit of the circuits, sacrificing speed to prevent the chips from melting.

Adiabatic reversible computing can reduce the power dissipation of a circuit by using slowly ramping up and down clocks as voltage supplies. This quasi-adiabatic approach yields the following expression for active power:

$$P_{active} = N\alpha {V_{dd}}^2 Cf \frac{2RC}{T} \quad (1)$$

Where N is the number of gates, α is the activity factor, Vdd is the ramping power supply, C is the load capacitance, f is the operating frequency, RC is the time constant of the gate, and T is the rising/falling time of the power supply. When T is much greater than the RC time constant the active power dissipation can be much lower than its CMOS counterpart. Even though adiabatic computing was dismissed as slow before [2], it has now become attractive since current microprocessors already operate well below their RC limits.

Adiabatic logic presents a viable alternative to reduce active power when compared against other approaches like dark silicon where devices are simply turned off. Circuits dominated by active power such as microprocessors benefit the most from adiabatic logic as it can dramatically reduce energy dissipation.

Adiabatic computing uses reversibility to recover energy supplied to a circuit, and can be implemented as split-rail charge recovery logic (SCRL) [3]. An inverter using adiabatic SCRL logic is shown in Fig. 1(a), where the power supply and ground have been replaced by ramping clocks. As seen in Fig. 1(b), both ramping clocks start at a null state (0 V), then ramp up for time T until reaching a valid logic state, retain the logic value for some period, and ramp back down to the null state recovering energy. It should be noted that since the logic values are not valid when the clocks are ramping up and down or during the null state, the following stages require clocks with different phases.

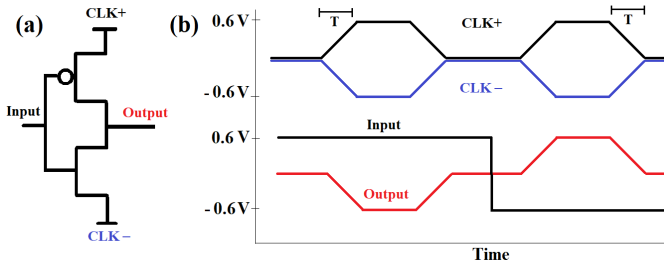


Fig. 1. (a) Adiabatic inverter implemented with SCRL. (b) Timing diagram.

A chain of three SCRL inverters is shown in Fig. 2, which illustrates the use of phased clocks to power different stages of the circuit. This clocking scheme is known as Bennett clocking [4], which consists of clocks that only ramp up once the previous stage has a valid state. Similarly, during the energy recovering process the last level of logic ramps down first, and once the null state is reached the previous stage follows. Bennett clocking ensures that the logical input values of the circuit are valid during energy recovery and presents a straightforward implementation to implement reversibility.

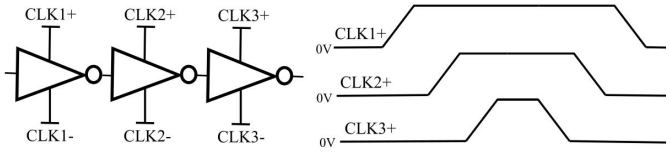


Fig. 2. Timing diagram of three-level Bennett clocking.

Supported by the US Air Force Research Laboratory under STTR AF18B-T013. Clearance number AFMC-2019-0527.

II. CHOICE OF TECHNOLOGY

An adiabatic SCRL implementation allows a circuit to operate both in adiabatic and in standard CMOS mode, providing a direct comparison of the power between the two modes. To evaluate the benefits of adiabatic computing a three-bit shift register is analyzed. The shift register shown in Fig. 3(a) uses transmission gates and inverters to control and shift three bits of information. The transmission gates control the feedback to avoid the uncontrolled erasure of information, and the inverters can hold one bit. The energy dissipation of the middle inverter in the shift register is analyzed as it operates. The same circuit is operated in standard CMOS mode, and in adiabatic reversible mode. As seen in Fig. 3(b) an adiabatic operation with Bennett clocking requires ramping clocks only for the inverters since the transmission gates are simply letting information through. SPICE simulations are performed using the predictive technology model for 90nm developed by Arizona State University (ASU) [5] in order to avoid the use of sensitive foundry specific SPICE models. The simulations use equally sized transistors, with a ratio of W/L=6 and 1 Volt power supply.

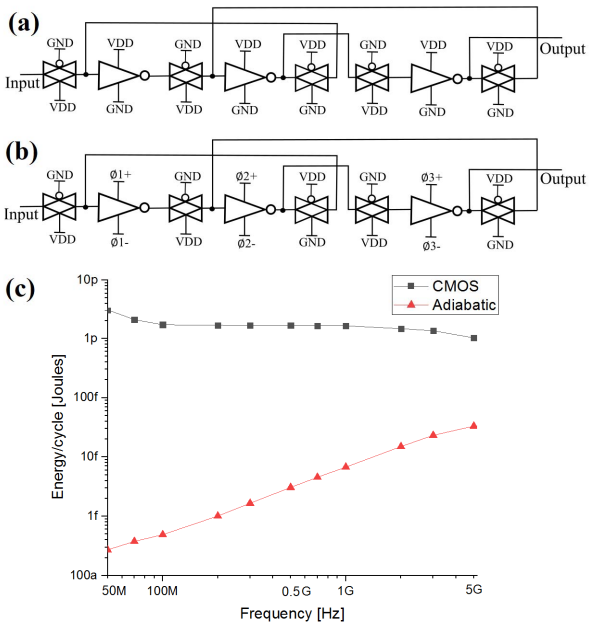


Fig. 3. (a) CMOS shift register. (b) Adiabatic shift register. (c) Energy per cycle SPICE simulation comparing both modes of operation.

The SPICE simulation of the shift register energy over one cycle of operation at different frequencies is presented in Fig. 3(c). The adiabatic energy increases with frequency, as with other adiabatic implementations [6], even though the CMOS energy remains relatively constant. At high enough frequencies, the adiabatic energy dissipation would be the same as in standard CMOS mode. However, the range of frequencies presented show that up to 5 GHz an adiabatic operation of the circuit still shows an energy dissipation orders of magnitude lower than CMOS.

The choice of technology node for an adiabatic circuit involves a number of trade-offs. The 90nm node technology has been used for low power CMOS applications [7] and as shown by the shift register simulations it is a viable technology for an adiabatic implementation. Other advanced node technologies could be used, but an energy analysis must be made in order to evaluate their benefits. To demonstrate this, we simulated the shift register using both the 90nm technology and the 28nm technology using the predictive technology models from ASU. Since both technologies can operate with a 1V power source the same architecture and power supply was used, and the transistors were scaled appropriately keeping the ratio of W/L=6. The results are presented in Fig. 4.

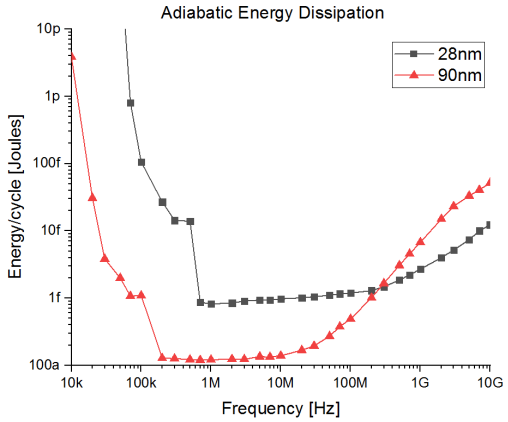


Fig. 4. Comparison of energy dissipated by an adiabatic shift register implemented in 90nm and 28nm.

As seen in Fig. 4, the energy per cycle of the adiabatic shift register changes with frequency depending on the dominating power dissipation mechanism, either passive power dissipation or active power dissipation. At low frequencies passive power dominates, until the energy reaches a minimum point, and then at higher frequencies active power takes over. The 90nm technology presents a lower energy dissipation at low frequencies due to the high passive power of the 28nm node mostly related to leakage. However, at higher operating frequencies where active power takes over, the energy plots cross and above 300 MHz the 28nm technology presents a lower energy. This is the expected behavior as shorter advanced nodes, like 28nm, have a lower bit energy since they use smaller devices which translates into lower active power dissipation.

We propose a microprocessor operating frequency of 0.5 GHz, which represents a useful speed as well as a low enough frequency to benefit greatly from adiabatic logic. For this frequency, the adiabatic energy dissipation as shown by the shift register is still orders of magnitude lower than in CMOS operation. And the 90nm technology is an appropriate choice since both the 90nm and 28nm technologies have a very similar energy dissipation. Therefore, the 90nm technology is a viable option to implement an adiabatic microprocessor.

III. Microprocessor Design

A. Architecture

We present the design of an adiabatic 16-bit datapath microprocessor implemented in 90nm technology using SCRL. The microprocessor is a multicycle RISC processor and it follows the MIPS architecture described by Harris [8]. Only a subset of instructions is implemented but they are enough for universal computation. The instructions are: addition, subtraction, bitwise AND, bitwise OR, set less than, add immediate, branch if equal, jump, load byte and store byte. This microprocessor represents an improvement over previous implementations [9] since it has a larger bit datapath and it is designed using a 90nm technology node.

The microprocessor is composed of a controller unit, an external memory, and a 16-bit datapath with multiple combinational and sequential elements as shown in Fig. 5. Three different Bennett blocks are defined in order to implement adiabatic logic with Bennett clocking, and each

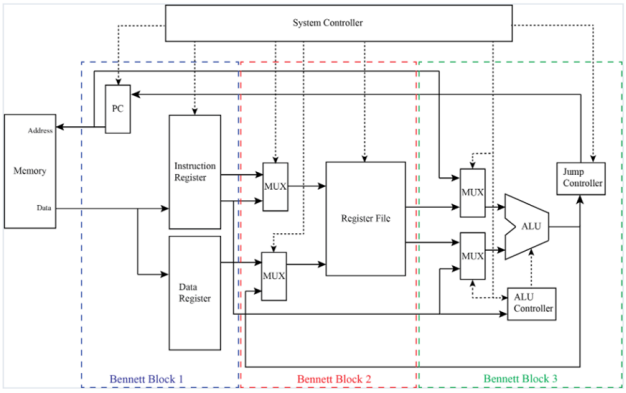


Fig. 5. Adiabatic microprocessor main components showing three separate Bennett blocks to implement adiabatic logic with Bennett clocking.

Bennett zone contains multiple Bennett clock levels. It is desirable to implement the Bennett blocks in the lowest possible number of clocks to reduce complexity in terms of implementation, timing, and testing. The critical path determines the minimum number of Bennett clocks required for energy recovery. The critical path of the microprocessor is between the register file and the Algorithmic and Logic Unit (ALU) output register, which must be optimized for adiabatic computing.

The biggest block in the critical path is the ALU which is composed of an addition unit, bitwise AND, bitwise OR, a comparator, and a result multiplexer. The dedicated comparator unit is introduced into the ALU in order to implement the "set less than" instruction in the lowest possible number of Bennett clocks, in contrast to a regular CMOS implementation which uses the addition unit to perform this instruction [10]. However, introducing a dedicated comparator reduces the complexity of an adiabatic implementation.

The addition unit inside the ALU is the most complex block within the critical datapath, therefore a fast addition architecture is chosen in order to optimize the number of Bennett clocks used. Fast adders using adiabatic logic have shown benefits over standard CMOS, such as the 16-bit carry-look-ahead adder by Lim, J. et al. [11]. Yet in order to reduce the number of Bennett clocks a Kogge-Stone architecture [12] is selected which only requires eight Bennett clocks to perform 16-bit addition. The Kogge-Stone adder uses propagate-generate logic, similarly to other fast adder architectures, in order to optimize the carry out signals generated by bitwise addition.

B. Physical Design

We design a library of standard cells including all the basic gates to be used in the 16-bit microprocessor. Careful design of the standard cells allows them to be easily interconnected and generate all the building blocks of the processor. For example, the design of an adiabatic SCRL NAND gate is shown in Fig. 6(a). It includes inputs and outputs, substrate connections for both n-channel and p-channel devices, and the positive and negative power clocks which follow Bennett clocking. It should be noted that the n and p wells of the devices are connected to DC voltage supplies, while the source and drain terminals of the transistors are connected to the power clocks to implement adiabatic logic.

The generic 90nm process design kit by Cadence (gpdk090) is used to implement the physical layout of the microprocessor. As seen in Fig. 6(b) the physical implementation of the NAND gate includes inputs at the top and output at the bottom which creates a downward flow of information using Bennett Clocking. Previous logical stages are placed above the gate and following stages below. Other gates that share the same Bennett level can simply be placed on the side. The floor-planning of complex circuits is straightforward using standard cells, the only limitation being the power rails which have a specific Bennett clock.

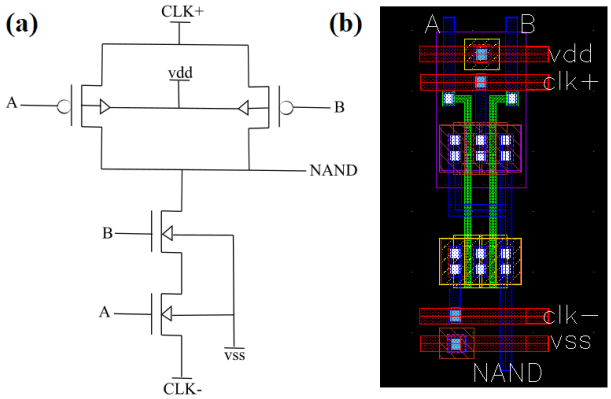


Fig. 6. (a) Adiabatic two input NAND circuit schematic. (b) NAND physical implementation using 90nm technology.

The physical layout for the Kogge-Stone 16-bit adder illustrates the use of the standard cell library and is presented in Fig. 7. The adder is composed of 972 transistors, and it occupies an area of 80 um by 60 um. Eight Bennett levels can be seen from top to bottom, each level contains both a positive and a negative clock. A bit slice shows the path of single bit addition, and the full adder contains 16 bit slices. Only four layers of metal are used to interconnect this fast adder, but another two metal layers are required to interconnect it with the rest of the ALU.

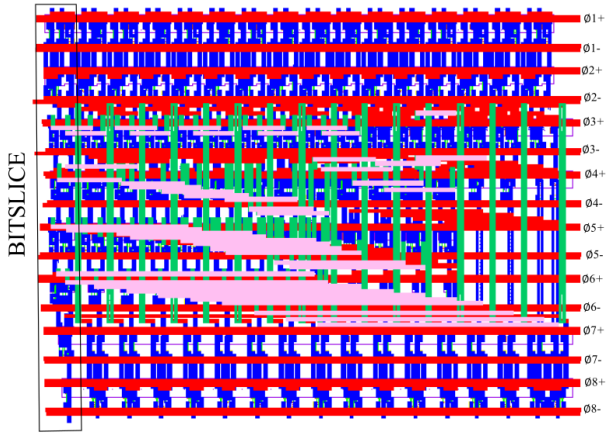


Fig. 7. Adiabatic 16-bit Kogge-Stone adder layout implemented in 90nm technology.

The physical design is verified using a layout versus schematic (LVS) check, and a design rule check (DRC) included in the generic 90nm kit [13]. The LVS requires a schematic with matching architecture, which we also developed for our standard cell library. The design rules used in the DRC verification do not correspond to any foundry and are conservative in terms of the minimum distance between features. This allows the design to be adjusted for a specific foundry fabrication technology if needed. Furthermore, we implement our devices leaving some flexibility to more restrictive design rules, while still passing the DRC checks from the generic model. This extra flexibility might not yield the optimal minimum area for the design but allows the microprocessor to be easily adjusted for future fabrication.

The adder implemented using only eight Bennett levels allows the critical path of the microprocessor to use only 12 Bennett clocks. These levels include the register source operands, multiplexers, the ALU, and the ALU output register. Optimizing the architecture of the ALU for adiabatic computing shows that the number of clocks needed to implement this 16-bit microprocessor can be equal to the number required for an earlier 8-bit adiabatic microprocessor. The only extra complexity of the circuit is related to the increased size of the datapath, and the effort to interconnect the main subsystems of the microprocessor.

This physical implementation realizes adiabatic computing successfully using Bennett clocking and it shows that logic can be compactly placed even with the separate logical Bennett levels.

IV. Conclusion

A 16-bit adiabatic microprocessor is successfully implemented in 90nm technology using split-rail charge recovery logic. The simulation of a subsystem of the microprocessor, a shift register, shows the benefits of using adiabatic logic against standard CMOS and confirms that the 90nm node is a viable option for its implementation. Optimizing the architecture of the microprocessor for adiabatic computing pays off by reducing the number of clocks, and therefore reducing the complexity of the system. The physical design of the microprocessor implements a newly developed standard cell library with Bennett clocking. The final layout is in progress estimated to be complete within three months. The implementation of the microprocessor in 90nm technology with an operating frequency of 0.5 GHz shows a real-life circuit using adiabatic reversible logic and shows a promising future for energy-efficient computing.

References

[1] M. M. Waldrop, “The chips are down for Moore’s Law,” *Nature: News Feature*, vol. 530, iss. 7589, February 9 2016.

[2] C. S. Lent, A. O. Orlov, W. Porod, and G. L. Snider, *Energy limits in computation: A Review of Landauer’s principle, Theory and Experiments,* New York: Springer, 2018.

[3] S. G. Younis and T. F. Knight, "Practical Implementation of Charge Recovering Asymptotically Zero Power CMOS," in *Research On Integrated Systems*, Seattle, WA, 1993, pp. 234-250.

[4] C. S. Lent, M. Liu, and Y. H. Lu, “Bennett clocking of quantum-dot cellular automata and the limits to binary logic scaling,” *Nanotechnology*, vol. 17, pp. 4240-4251, Aug 28 2006.

[5] W. Zhao, Y. Cao, "New generation of Predictive Technology Model for sub-45nm early design exploration," *IEEE Transactions on Electron Devices*, vol. 53, no. 11, pp. 2816-2823, Nov 2006.

[6] P. Teichman, *Adiabatic Logic: Future Trend and System Level Perspective*, New York: Springer, 2012.

[7] M. Meijer, et al., “Ultra-Low-Power Digital Design with Body Biasing for Low Area and Performance-Efficient Operation”, *Journal of Low Power Electronics*, vol. 6, No. 4, 2011.

[8] D. Harris, and S. Harris, *Digital design and computer architecture*, Boston : Morgan Kaufmann Publishers, 2007.

[9] C. O. Campos-Aguillón, et al., "A Mini-MIPS microprocessor for adiabatic computing," 2016 *IEEE International Conference on Rebooting Computing (ICRC)*, San Diego, CA, 2016, pp. 1-7.

[10] N. Weste and D. Harris, “*CMOS VLSI Design: A Circuits and Systems Perspectives,*” 4th ed.: Addison-Wesley, 2010.

[11] J. Lim, et. al., “A 16-bit carry-lookahead adder using reversible energy recovery logic for ultra-low-energy systems,” *IEEE Journal of Solid-State Circuits*, vol. 34 , iss. 6 , Jun 1999.

[12] D. Esposito, et al., " Variable Latency Speculative Parallel Prefix Adders for Unsigned and Signed Operands," *IEEE Transactions on Circuits and Systems I: Regular Papers,* vol. 63 , iss. 8, 2016.

[13] Cadence Reference Manual “*Specification for 90nm Generic Process Design Kit (gpdk090)*”, 2008.

Hierarchical Memcapacitive Reservoir Computing Architecture

Dat Tran, S.J., *Student Member, IEEE*
Department of Electrical and Computer Engineering
Portland State University
Portland, USA
datran@pdx.edu

Christof Teuscher, *Senior Member, IEEE*
Department of Electrical and Computer Engineering
Portland State University
Portland, USA
teuscher@pdx.edu

***Abstract*—The quest for novel computing architectures is currently driven by (1) machine learning applications and (2) the need to reduce power consumption. To address both needs, we present a novel hierarchical reservoir computing architecture that relies on energy-efficient memcapacitive devices. Reservoir computing is a new brain-inspired machine learning architecture that typically relies on a monolithic, i.e., unstructured, network of devices. We use memcapacitive devices to perform the computations because they do not consume static power. Our results show that hierarchical memcapacitive reservoir computing device networks have a higher kernel quality, outperform monolithic reservoirs by 10%, and reduce the power consumption by a factor of $3.4\times$ on our benchmark tasks. The proposed new architecture is relevant for building novel, adaptive, and power-efficient neuromorphic hardware with applications in embedded systems, the Internet-of-Things, and robotics.**

***Index Terms*—memcapacitive, reservoir computing, hierarchical structure.**

I. Introduction

Emerging nanoscale memory devices, such as memristors and memcapacitors, offer properties that make them ideal building blocks for the implementation of neuromorphic architectures. For example, a memcapacitor can directly emulate the synaptic plasticity as observed in biological systems [1].

Reservoir Computing (RC) is a recent brain-inspired computing paradigm that uses an unstructured computing core, i.e., the reservoir, to project an input into a higher-dimensional space [2]. A readout layer is then trained to interpret these high-dimensional dynamics as the desired output signal. RC is able to explain higher-order cognitive functions and the interactions of short-term memory with other cognitive processes [3].

Most reservoirs are assumed to be random and monolithic, i.e., unstructured. Little work has been done on hierarchically-organized reservoirs. For example, hierarchical reservoir systems offered better dynamics than monolithic single-layer reservoir networks for representing an acoustic model for speech recognition [4]. Memristive hierarchical reservoirs outperformed monolithic reservoirs by at least 20% for certain tasks [5]. Hierarchical random Boolean network reservoirs were able to adapt for information transferring [6]. A hierarchical *Echo State Network* (ESN) improved the learning capability for performing general tasks [7].

In this paper, we propose a new hierarchical reservoir in which each cluster node of the reservoir is a small-world power-law memcapacitive network. Our simulation results show that such hierarchical memcapacitive reservoirs (1) outperform monolithic reservoirs and (2) consume significantly less power.

II. Background

A *memcapacitor* (MC) is a nonlinear memory device that describes the relationship between the charge q and the voltage v: $dq = Cdv$. Its capacitance depends on the external applied voltage v as follows:

$$q = C\left(x, v, t\right) v,$$
$$\frac{dx}{dt} = f\left(x, v, t\right),$$

where q is the charge of the device at time t, v is the applied voltage, x is the internal state of the device, C is the capacitance, which depends on the internal state x, and the function $f()$ describes the changes of the internal state x.

In this paper, we will rely on two memcapacitor models: the *Biolek* and the *Mohamed* model. The generic *Biolek* memcapacitor model describes the function of a threshold memcapacitive device [8]. Its capacitance C is a function of the internal state variable x and is related to the applied voltage V_C:

$$q = CV_C,$$
$$\frac{dC}{dt} = f\left(V_C\right) W\left(C, V_C\right),$$

where $f()$ describes the device state change and the window function $W()$ sets the boundary of the internal state x. These functions are described by the following equations:

$$f\left(V_C\right) = \beta\left(V_C - 0.5 * \left[\left|V_C + V_t\right| + \left|V_C - V_t\right|\right]\right),$$
$$W\left(C, V_C\right) = \theta\left(V_C\right) * \theta\left(C_{max} - C\right) + \theta\left(-V_C\right) * \theta\left(C - C_{min}\right),$$

where the step function $\theta()$ limits the capacitance of the device to the range $[C_{min}, C_{max}]$, β is a constant rate of change when $|V_C|$ is greater than the threshold voltage V_t, and C_{min} and C_{max} are the minimum and maximum values of the device

978-1-7281-5221-9/19/$31.00 ©2019 IEEE

capacitance. The capacitance of the *Biolek* model ranges from $C_{min} = 1pF$ to $C_{max} = 100pF$.

The *Mohamed* model describes the memcapacitive response of a metal-oxide junction [9]. Mohamed *et al.* observed that the effect of the growth or the shrinkage of thin filaments exhibited a memcapacitive behavior when the *Behavior Shape Factor* (BSF), the ratio of the tunneling current and the capacitive current, is less than 0.1. The state equations are explained in [9]. The capacitance of the *Mohamed* device ranges from $C_{min} = 1nF$ to $C_{max} = 10nF$. For our purpose, the constants of the model were modified to accommodate a very slow input signal of 1Hz: $K_G = 0.4775, K_S = 0.6, B_G = 2.75, B_S = 2.75, x_{min} = 0.4, x_{max} = 0.9, m_{min} = 0.1, m_{max} = 0.9, d_1 = 5 \times 10^{-10}$, and $d_2 = 5 \times 10^{-10}$.

III. Memcapacitive Reservoir Computing

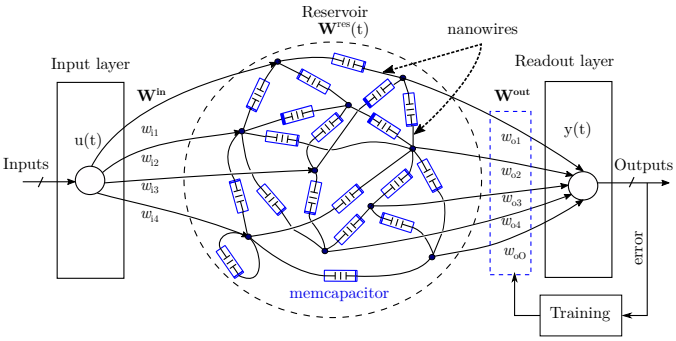


Fig. 1: A memcapacitive reservoir network. The reservoir is an electrical network in which memcapacitors are randomly connected through nanowires. The node voltages, the time-dependent state $x(t)$ of the reservoir, provide signals to the readout layer. The state matrix, $W^{res}(t)$, contains the capacitance of each memcapacitive device at time t. Only the readout layer is trained by a simple gradient descent algorithm.

Reservoir Computing (RC) is a relatively recent, alternative machine learning architecture [2]. Compared to traditional *Recurrent Neural Networks* (RNN), RC avoids the training of large networks with tens of thousands or even more weights. In an RC architecture, only the readout layer is trained by a simple gradient descent algorithm. The reservoir itself is untrained and typically made up from an unstructured, i.e., random, network of some nonlinear devices. In simplified terms, the only constraint on the reservoir is that it must offer "interesting enough" dynamics.

Fig. 1 shows an example of a small-world power-law [10] memcapacitive reservoir, where the memcapacitors interconnect network has a small-world topology and the wires follow a power-law length distribution. Such networks were shown to offer optimal transport characteristics at a low wiring cost [10]. Scaled by a fixed-value weight matrix W^{in}, the input signal $u(t)$ provides stimuli to the reservoir nodes. The time-dependent state $x(t)$ of the reservoir is its node voltages that are determined by the internal capacitance of the memcapacitive devices at time t. The internal capacitance of the memcapacitive devices are represented as the state matrix $W^{res}(t)$, where $W^{res}(t) = [mc_1(t), mc_2(t), mc_3(t), \cdots, mc_n(t)])$. The reservoir state $x(t)$ is defined as:

$$x(t+1) = f\left[W^{res}(t)x(t) + W^{in}u(t)\right],$$

where f is the reservoir node transfer function. The output $y(t)$ is the inner product of the reservoir state $x(t)$ multiplied by the output weight matrix W^{out}:

$$y(t) = x(t)W^{out}.$$

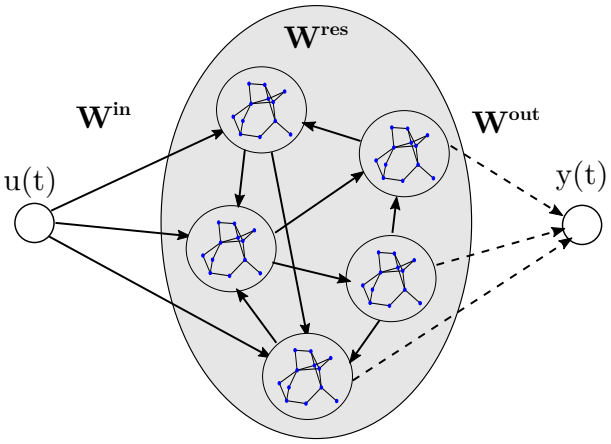


Fig. 2: An example of a hierarchical memcapacitive reservoir. Each cluster is a small-world power-law memcapacitive network.

Fig. 2 illustrates an example of a hierarchical memcapacitive reservoir. Each reservoir node (or cluster node) is a small-world power-law memcapacitive network interconnected as described in [10]. The input signals $u(t)$ are applied to cluster nodes through an input matrix W^{in} with values from a Bernoulli distribution on the interval $\{-\kappa, \kappa\}$. The cluster nodes are connected with random weights w specified by the spectral radius λ of the reservoir weight matrix W^{res}. According to Jaeger, $|\lambda| \leq 1$ is an essential condition to maintain the behavior of a single-layer reservoir [2]. When $|\lambda| > 1$, signals circulating within a network are amplified indefinitely, which leads to chaotic behavior. The condition on λ also applies to the cluster nodes within the hierarchical reservoirs. It has been shown that λ could be greater than 1 for hierarchical memristive reservoirs due to signal loss caused by the resistance of memristive devices [5]. The output voltages at each reservoir node are forwarded to a readout layer for training with a linear regression algorithm. Note that only the readout layer is trained.

In this paper, we investigate four hierarchical memcapacitive reservoir topologies: a *Delay-Line* (DL), a *Delay-Line with Feedback* (DLFB), a *Simple Cycle* (SC), and a *Random* (R) topology. A detailed description of the first three hierarchical networks is given in [11]. For the random (R) topology, the cluster nodes are simply connected randomly.

TABLE I: Physical characteristics of mem-devices.

Model †	Device	Type ‡	Values at w	
			w_{max}	w_{min}
Biolek [8]	*	$MemC$	$100pF$	$1pF$
Mohamed [9]	$SiTiO_3/Ti$	$MemC$	$6.5nF$	$1nF$

† These models were modified to exhibit a state volatility, an essential property in reservoir computing.
* This model is not based on a physical device.
‡ Memcapacitors are abbreviated as MemC.

IV. Methodology

A. Device Models

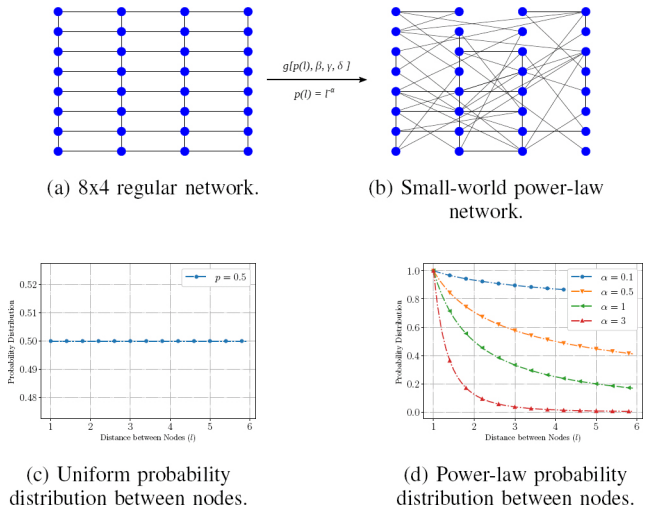
Although memcapacitive behavior has been found in many new nano-composites [12]–[14], we have chosen the generic *Biolek* and *Mohamed* metal-dioxide junction models because they provide a mathematical model that describes the internal state of the device. The device characteristics are listed in Table I. Note that we have modified both models to incorporate state volatility, which is essential for reservoir computing.

B. Small-world Power-law Networks as Cluster Nodes

Although most traditional reservoir typologies are random [5], [15], [16], such topologies are not physically plausible [17], [18]. Most real networks, such as online social networks, cultural networks, brain networks, and the internet typically have the *Small-world* (SW) property combined with a distance-dependent, i.e., spatial, wire length distribution [10]. In such networks, long-distance connections are less likely than short-distance ones, which leads to a low overall wiring cost compared to random or non-spatial SW networks only.

Various studies have shown that brain networks have the SW property [19], [20], which allows to minimize the cost of information processing and to maximize the capacity for growth and adaptation [18].

In this study, cluster nodes are *SW power-law* (SWPL) networks, which is the subset of the SW networks. Both the SW and SWPL networks are similar in describing the fundamental characteristics of a small-world phenomenon. The main difference, however, is the formation of a network: an SW network arises from a regular ring network whereas an SWPL network is formed from a grid network, shown in Fig. 3. In the original SW model [17], nodes in a regular network have a uniform probability distribution to connect to any other nodes within the network. The uniform distribution is not realistic since each connection, local or global, is associated with a cost, such as wire connection, to the system, whose resources are often limited [10]. In the SWPL model, the decay power law, $q(l) = l^{-\alpha}$, governs the formation of a connection. According to the decay power law, distant nodes have a smaller probability to form connections compared to nearby nodes for the reason that long or global connections require more wire lengths than short or local ones. The formation of an SWPL network reflects the growth of real networks, including biological neural networks, which tend to have more



(a) 8x4 regular network.
(b) Small-world power-law network.
(c) Uniform probability distribution between nodes.
(d) Power-law probability distribution between nodes.

Fig. 3: To create a small-world power-law network from a regular network (a), which has an uniform link probability distribution between nodes (c), the network is transformed into a small-power network (b) by rewiring local connections with global links that follow a power-law distance probability distribution (d).

short connections than long connections due to their limited resources [18].

C. Reservoir Kernel Quality

Kernel quality measures the linear separation property of a reservoir [21]. The linear separation describes how a reservoir disassociates different input patterns, independent of a target output. Kernel quality is based on a set of n reservoir states generated by m different input stimuli. All reservoir states are collected into a matrix M of size $n \times m$. Each column is a state vector corresponding to a particular input. The rank r of this state matrix M is a measure of the computational power of the reservoir. A reservoir has the highest computational power when the matrix M has a maximum rank, or $r = m$. In this case, each column in the matrix M is linearly independent of the other columns.

D. Benchmark Tasks

We use three common benchmark tasks to evaluate our reservoir systems: a spoken digit recognition, MNIST pattern recognition, and CIFAR-10 classification.

The spoken digit recognition is widely used as a benchmark for RC systems [22]. The spoken digit dataset contains the recordings of spoken digits in wave file formats at 8kHz [23]. The dataset is divided into two non-overlapping sets, 1,000 digits for training and 500 digits for testing. The recordings are translated into input vectors of Mel-frequency cepstral coefficients, which is a common pre-processing technique for a speech recognition [24]. Each recording segment of a digit is divided into 3:4:3 ratio regions with 13 Mel frequency cepstral features for each region [25]. In addition, each region

is calculated for its delta and delta-delta coefficients to capture all features. The final result is a 117-coefficient vector. The coefficient vectors are translated into input voltages and scaled to a range of $[-\nu, \nu]$ to avoid saturating the reservoirs.

MNIST is a common pattern recognition benchmark [15], [26]. We flattened the images to 784-pixel vectors and used them as reservoir inputs for training and testing. The amplitudes of the pixel vectors were scaled to a range of $[0, \nu]$ to avoid saturating the reservoirs. To keep the simulation time reasonable, we only used 1,000 digits for training and 400 for testing.

CIFAR-10 is another common machine learning benchmark [27]. Since CIFAR-10 images are high-dimensional color images, we converted training and testing images into grayscale images to reduce the computational complexity. To keep the simulation time reasonable, we only used 1,000 images for training and 400 for testing.

V. Results

With 50 different digits chosen as inputs from the isolated spoken digit dataset, we simulated the four hierarchical reservoirs, i.e., the *Delay Line* (DL), the *Delay Line with Feedback* (DLFB), the *Simple Cycle* (SC), and the *Random* (R) reservoir. Monolithic reservoirs were used as a baseline for comparison. Both monolithic and hierarchical reservoirs were trained and tested on the three tasks (see Section IV-D) for their performance and power consumption. Each experiment was repeated 50 times and we averaged the obtained results.

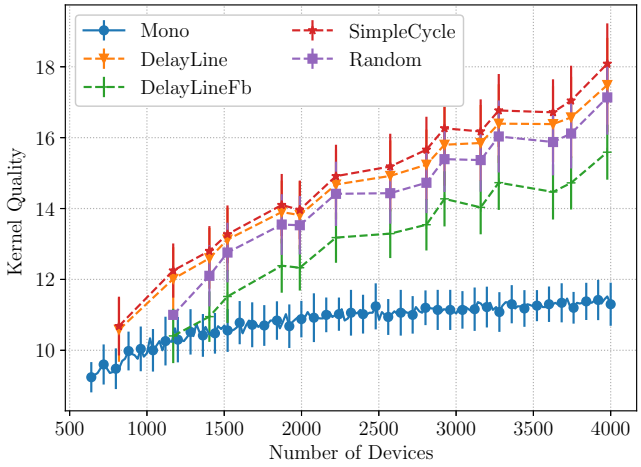


Fig. 4: Kernel quality of monolithic and hierarchical *Biolek* reservoirs.

The simulation results of monolithic and hierarchical reservoirs are shown in Figs. 4 and 5. The results show a common trend in which the hierarchical reservoirs have a high kernel quality compared to monolithic reservoirs. The kernel quality also increases with the number of devices. For example, for a 3,000-device reservoir, the kernel quality of the hierarchical reservoirs is on average $1.4\times$ better than for the monolithic reservoirs.

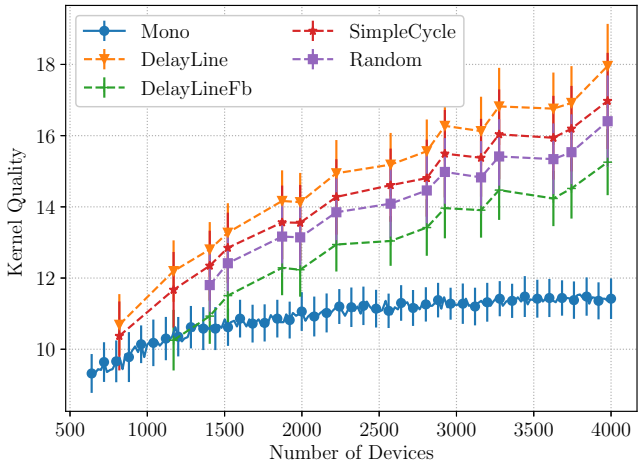


Fig. 5: Kernel quality of monolithic and hierarchical *Mohamed* reservoirs.

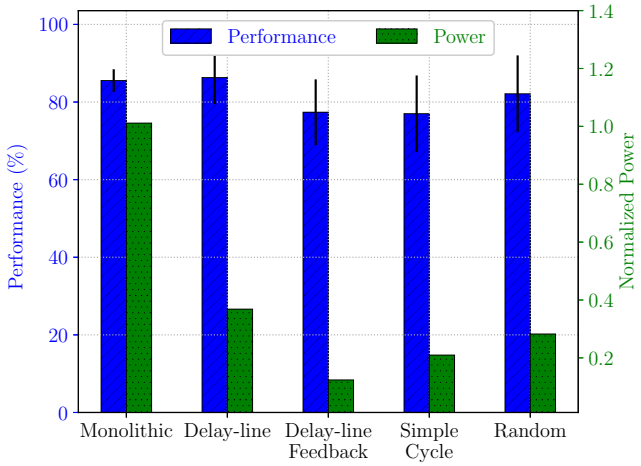


Fig. 6: Performance and power consumption of monolithic and hierarchical reservoirs for the isolated spoken digit task.

The performance and power consumption summary results for all reservoirs are shown in Figs. 6, 7, and 8. The low performance of the reservoirs on the CIFAR-10 task, compared to the-state-of-the-art neural networks (about 94.03% [28]), is due to the dimension reduction of the grayscale from the color images. Maintaining the original dimension of the color image would increase the performance of the reservoirs but also significantly expands the reservoir size. This would lead to impractical simulation times.

For the MNIST pattern task (Fig. 7), the hierarchical reservoirs consumed more power compared to the monolithic ones. We suspect that since the input data from the MNIST images are quite sparse, the monolithic reservoirs have sufficient dynamics to translate the input data into their internal states whereas the hierarchical reservoirs do not increase performance by much, but instead are composed of more devices, which then results in higher power consumption.

For the other two tasks, the isolated spoken digits and the

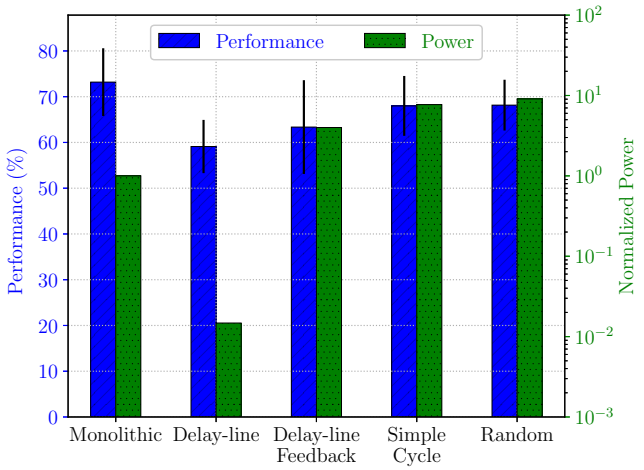


Fig. 7: Performance and power consumption of monolithic and hierarchical reservoirs for the MNIST task.

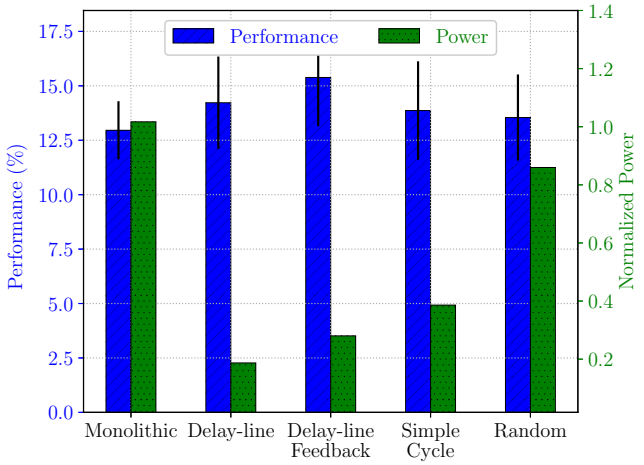


Fig. 8: Performance and power consumption of monolithic and hierarchical reservoirs for the CIFAR-10 task.

CIFAR-10 images, the input signals are more complex and the hierarchical reservoirs yielded both higher performance and power consumption. For the isolated spoken digits, the hierarchical reservoirs on average reduced the power consumption without any performance loss by a factor of $5.1\times$. Similarly, for the CIFAR-10 image task, the hierarchical reservoirs accomplished on average a higher performance by a factor of $1.1\times$ and consumed $3.4\times$ less power.

In Fig. 9 shows the effect of individual memcapacitive models on the performance and the power consumption of different hierarchical topologies. The results show that for the three tasks, on average, the *Mohamed* reservoirs with delay-line and delay-line feedback topologies performed better than the *Biolek* counterparts. However, the *Mohamed* reservoirs consumed more power than the *Biolek* reservoirs on the same hierarchical topology. The reason is that the *Mohamed* device has a higher capacitance range (see Table I) and the *Mohamed* reservoirs employed more devices compared to the *Biolek*

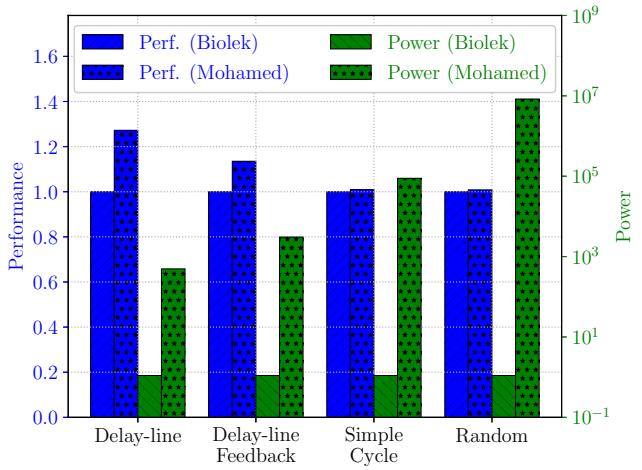


Fig. 9: Average performance and power consumption of memcapacitive hierarchical reservoirs. The performance and power measurements of *Biolek* reservoirs are normalized as the references for the three tasks to compare with the performance and power of *Mohamed* reservoirs for each hierarchical topology.

reservoirs, about $3\times$ on average. We note that both the capacitance range of the device and the reservoir topology contribute to the performance and power consumption of memcapacitive reservoirs. However, a full investigation of these factors is beyond the scope of this paper.

VI. Conclusion

We have shown how networks of nonlinear memcapacitive devices can be used as monolithic and hierarchical reservoirs for reservoir computing. The results from our simulations illustrated that hierarchical memcapacitive reservoir computing device networks have a higher kernel quality, outperformed monolithic reservoirs by 10%, and reduced the power consumption by a factor of $3.4\times$ on our three benchmark tasks. We argue that our new machine learning architecture is relevant for building emerging, adaptive, and power-efficient neuromorphic hardware.

Acknowledgment

This work is **supported** in part by **C-BRIC, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA**. The views expressed are those of the author(s) and do not reflect the official policy or the position of Semiconductor Research Corporation (SRC) program. Approved for Public Release, Distribution Unlimited.

References

[1] Y. Jeong and W. Lu, "Neuromorphic Computing Using Memristor Crossbar Networks: A Focus on Bio-Inspired Approaches," *IEEE Nanotechnology Magazine*, vol. 12, no. 3, pp. 6–18, 2018.

[2] H. Jaeger, "The "echo state" approach to analysing and training recurrent neural networks-with an erratum note," *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, vol. 148, no. 34, p. 13, 2001.

[3] M. Rigotti, D. D. Ben Dayan Rubin, X.-J. Wang, and S. Fusi, "Internal representation of task rules by recurrent dynamics: the importance of the diversity of neural responses," *Frontiers in Computational Neuroscience*, vol. 4, p. 24, 2010.

[4] F. Triefenbach, A. Jalalvand, B. Schrauwen, and J.-P. Martens, "Phoneme recognition with large hierarchical reservoirs," in *Advances in neural information processing systems*, 2010, pp. 2307–2315.

[5] J. Bürger, A. Goudarzi, D. Stefanovic, and C. Teuscher, "Hierarchical composition of memristive networks for real-time computing," in *Nanoscale Architectures (NANOARCH), 2015 IEEE/ACM International Symposium on*, IEEE. Boston, MA: IEEE, 2015, pp. 33–38.

[6] P. J. Górski, A. Czaplicka, and J. A. Hołyst, "Coevolution of information processing and topology in hierarchical adaptive random boolean networks," *The European Physical Journal B*, vol. 89, no. 2, p. 33, 2016.

[7] M. Dale, "Neuroevolution of hierarchical reservoir computers," in *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2018, pp. 410–417.

[8] D. Biolek, M. Di Ventra, and Y. V. Pershin, "Reliable SPICE simulations of memristors, memcapacitors and meminductors," *Radioengineering*, vol. 22, no. 4, pp. 945–968, July 2013.

[9] M. G. A. Mohamed, H. Kim, and T. Cho, "Modeling of Memristive and Memcapacitive Behaviors in Metal-Oxide Junctions," *The Scientific World Journal*, vol. 2015, p. 910126, 2014.

[10] T. Petermann and P. De Los Rios, "Physical realizability of small-world networks," *Physical Review E*, vol. 73, no. 2, p. 026114, 2006.

[11] A. Rodan and P. Tino, "Minimum complexity echo state network," *IEEE Transactions on Neural Networks*, vol. 22, no. 1, pp. 131–144, 2011.

[12] J. A. Patel, Z. T. Sandhie, and M. H. Chowdhury, "Ternary Device using Graphene Memcapacitor for Post Binary Era," in *2018 IEEE 61st International Midwest Symposium on Circuits and Systems (MWSCAS)*, IEEE. Windsor, ON, Canada: IEEE, 2018, pp. 1130–1133.

[13] P. Yang, H. J. Kim, H. Zheng, G. W. Beom, J.-S. Park, C. J. Kang, and T.-S. Yoon, "Synaptic transistor with a reversible and analog conductance modulation using a $Pt/HfO_x/n-IGZO$ memcapacitor," *Nanotechnology*, vol. 28, no. 22, p. 225201, 2017.

[14] A. K. Khan and B. H. Lee, "Monolayer MoS_2 metal insulator transition based memcapacitor modeling with extension to a ternary device," *AIP Advances*, vol. 6, no. 9, p. 095022, 2016.

[15] D. Tran SJ and C. Teuscher, "Memcapacitive reservoir computing," in *2017 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*. Newport, RI, USA: IEEE, 2017, pp. 115–116.

[16] K. S. Scharnhorst, J. P. Carbajal, R. C. Aguilera, E. J. Sandouk, M. Aono, A. Z. Stieg, and J. K. Gimzewski, "Atomic switch networks as complex adaptive systems," *Japanese Journal of Applied Physics*, vol. 57, no. 3S2, p. 03ED02, 2018.

[17] D. J. Watts and S. H. Strogatz, "Collective dynamics of "small-world" networks," *Nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[18] A. K. Barbey, "Network Neuroscience Theory of Human Intelligence," *Trends in Cognitive Sciences*, vol. 22, no. 1, pp. 8–20, 2018.

[19] U. Chockanathan, A. Z. Abidin, A. M. DSouza, G. Schifitto, and A. Wismüller, "Resilient modular small-world directed brain networks in healthy subjects with large-scale Granger causality analysis of resting-state functional MRI," in *Medical Imaging 2018: Biomedical Applications in Molecular, Structural, and Functional Imaging*, vol. 10578, International Society for Optics and Photonics. Houston, TX: SPIE, 2018, p. 105780B.

[20] X. Ma, G. Jiang, S. Fu, J. Fang, Y. Wu, M. Liu, G. Xu, and T. Wang, "Enhanced network efficiency of Functional Brain networks in Primary insomnia Patients," *Frontiers in Psychiatry*, vol. 9, p. 46, 2018.

[21] R. Legenstein and W. Maass, "Edge of chaos and prediction of computational performance for neural circuit models," *Neural Networks*, vol. 20, no. 3, pp. 323–334, 2007.

[22] L. E. Suarez, J. D. Kendall, and J. C. Nino, "Evaluation of the computational capabilities of a memristive random network (MN3) under the context of reservoir computing," *Neural Networks*, vol. 106, pp. 223–236, 2018.

[23] Z. Jackson, "Free Spoken Digit Dataset (FSDD)," https://github.com/Jakobovski/free-spoken-digit-dataset, 2016, accessed: 2018-04-09.

[24] J. Bouvrie, T. Ezzat, and T. Poggio, "Localized spectro-temporal cepstral analysis of speech," *Reconstruction*, vol. 7192, no. 6474, p. 5755, 2008.

[25] P. Clarkson and P. J. Moreno, "On the use of support vector machines for phonetic classification," in *Acoustics, Speech, and Signal Processing, 1999. Proceedings., 1999 IEEE International Conference on*, vol. 2, Phoenix, AZ, 1999, pp. 585–588.

[26] D. Tran SJ and C. Teuscher, "Memcapacitive Devices in Logic and Crossbar Applications," *International Journal of Unconventional Computing*, vol. 13, no. 1, pp. 35–57, 2017.

[27] M. Ghayoumi, M. Gomez, K. E. Baumstein, N. Persaud, and A. J. Perlowin, "Local Sensitive Hashing (LSH) and Convolutional Neural Networks (CNNs) for Object Recognition," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2018, pp. 1197–1199.

[28] Y. Abouelnaga, O. S. Ali, H. Rady, and M. Moustafa, "Cifar-10: Knn-based ensemble of classifiers," in *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE, 2016, pp. 1192–1195.

Integrating Motion into Vision Models for Better Visual Prediction

Michael Hazoglou
Department of Electrical and Computer Engineering
University of California San Diego
San Diego, USA
mhazoglou@eng.ucsd.edu

Todd Hylton
Department of Electrical and Computer Engineering
University of California San Diego
San Diego, USA
thylton@eng.ucsd.edu

***Abstract*—We demonstrate an improved vision system that learns a model of its environment using a self-supervised, predictive learning method. The system includes a pan-tilt camera, a foveated visual input, a saccading reflex to servo the foveated region to areas high prediction error, input frame transformation synced to the camera motion, and a recursive, hierachical machine learning technique based on the Predictive Vision Model. In earlier work, which did not integrate camera motion into the vision model, prediction was impaired and camera movement suffered from undesired feedback effects. Here we detail the integration of camera motion into the predictive learning system and show improved visual prediction and saccadic behavior. From these experiences, we speculate on the integration of additional sensory and motor systems into self-supervised, predictive learning models.**

***Index Terms*—computer vision, machine learning, biologically inspired**

I. Introduction

Deep learning models are incredibly successful with specific tasks such as classification problems or achieving superhuman skills playing in games [1]–[5], but these models have well known limitations. For example, classification problems require large amounts of labeled data for supervised training of models and reinforcement learning requires a reward function associated with inputs and actions. A classifier built to distinguish cars from airplanes is often fooled by a collage of features from cars or airplanes and by more subtle, engineered adversarial inputs. Reinforcement learning based models are generally aware of the entire state of the game that they are learning to play. In general these techniques rarely generalize to other similar tasks without using specialized training methods such as transfer learning or meta-learning and usually require more data for training. These limitations are particularly apparent when considering the challenge of creating an intelligent agent capable or interacting with the world. These agents will never have complete information about their environment and must build simplified models based on limited information that are relevant and general to the agent's needs or tasks. We suppose that future computing systems will need such capabilities more generally: that is, computing systems will need to spontaneously learn simplified, relevant models of their data environments with minimal, task-specific configuration and programming. In this contribution, we illustrate such ideas in the context of a vision system that learns its environment by interacting with it.

In previous work [6] we developed a biologically inspired model to address some of these issues, by focusing on a simple problem of visually surveying an environment by emulating saccades; the rapid movements of the eye. This Error Saccade Model (ESM) uses the prediction error of the Predictive Vision Model (PVM) [7] to determine where to move the field of view based on where the total error in a sliding window is maximal. The intuition behind the ESM is that our eyes are naturally attracted to unexpected events. Taking our biological inspiration further we modified the PVM to include a fovea, which is a pit with a high density of cones in the eye (cone cells are photo receptors responsible for detecting color). As compared to PVMs without foveas, we showed that the PVM with a fovea spent significantly more time with its field of vision on complicated (high image entropy) objects. In this article the authors propose incorporating the motion of the camera into the visual prediction by applying a simple linear transformation according to the corresponding movement.

II. The Predictive Vision Model

The Predictive Vision Model (PVM) is a self-supervised hierarchical recurrent neural network that takes an input sequence and predicts the next input. The hierarchy of the PVM is composed of individual units which compress their respective inputs and exchange their hidden contextual information among each other to produce a prediction. Individual units of the PVM perform the calculations listed in Table I. The hierarchy is split into different levels where the next higher adjacent level of the hierarchy predicts the hidden states of the lower level, and the higher level feeds its context down to this lower adjacent level (see Figure 1). The hierarchies are built from rectangular grids at each level with nearest neighbor connections. Between adjacent levels in the hierarchy connections are formed in pyramidal arrangements such that typically 4 units in square arrangement pass their context up to one unit in the superior level. This reduces the number of units at each level resulting in a pyramidal structure. The PVM is trained in an self-supervised manner using backpropagation with a squared error loss on each PVM unit's prediction of its future input signal. For a more

978-1-7281-5221-9/19/$31.00 ©2019 IEEE

Layer	Symbol	Definition
Inputs		
Signal	P_t	Fan-in from inferior level or raw video tile
Integral	I_t	$\tau I_{t-1} + (1-\tau)P_t$
Derivative	$D_{t/t-1}$	$0.5 + (P_t - P_{t-1})/2$
Previous Prediction Error	E_t	$0.5 + (P^*_t - P_t)/2$
Context	C_{t-1}	concat$[H_t \dots]$ from Hidden of self/lateral/superior/topmost units
Output		
Hidden	H_t	$\sigma(W_h \cdot [P_t; D_{t/t-1}; I_t; E_t; C_t] + b_h)$
Predictions		
Predicted Signal	P^*_{t+1}	$\sigma(W_p \cdot H_t + b_p)$
Loss Function		
Square Error	$\mathcal{L}_t$	$\sum_{\{\text{channels, pixels}\}}(P^*_{t+1} - P_{t+1})^2$

TABLE I

SUMMARY OF THE PVM UNIT. EACH UNIT CONSISTS OF A THREE LAYER PERCEPTRON WITH SIGMOID ACTION NEURONS. THE INDICES REPRESENT THE TIME STEP FOR THE RESPECTIVE VALUE. FOR MORE DETAILS SEE [7].

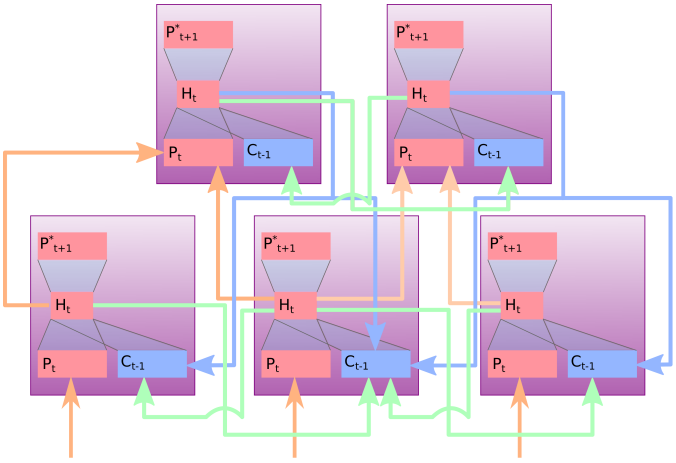


Fig. 1. An example of the PVM hierarchy used in [7]. The purple boxes represent the PVM units. Orange arrows represent the inputs into the units. The blue arrows show the flow of context from superior units while the green arrows show the flow of lateral context. The primary signal (bottom large red box) and context (in blue) is compressed to generate the (hidden) state of the unit which is shown as the smaller red box from which a prediction (top large red box) is made for the next input the unit will receive. Since the inputs to the units are fed one at a time, time derivatives, error in the previous prediction and time averages (integrals) are also factored in the hidden state calculation.

detailed discussion of the Predictive Vision Model see [7]. We reimplemented the PVM in previous work to run on Nvidia graphics cards see [6] for the detailed algorithm. All the code is available at https://github.com/mhazoglou/PVM_PyCUDA including scripts that were written for data collection and training.

III. The Error Saccade Model and Fovea

The fovea-like structure is created by splitting units of the PVM in the central region of lowest level into smaller and more numerous units. As the PVM units are split they remain connected to the units to which they were originally connected.

The error saccade model utilizes forced damped harmonic oscillation with gaussian white noise to emulate the movements of saccades. The total squared error in a sliding window of fixed height and width is calculated across the prediction error of the input level of the PVM hierarchy, the window with the largest error is used as the equilibrium point if the total squared error is larger than a threshold value. The threshold is equal to the time average of the maximum total square error in these sliding windows. This means that as the predictions get better/worse the movement will be more/less sensitive to prediction error. The error saccade model is complemented by the foveal region as the effect of having a lower density of PVM units in the periphery of the field of view results in higher prediction error, which induces saccades to that point. The foveal region has a higher density of PVM units meaning the prediction will have more detail which is beneficial for detailed (high image entropy) scenes. This means that a detailed object will likely stay in the fovea as any drift to the peripheral lower density region will cause a spike in error which will recenter fovea on the object. As we observed in our previous work, the model replicates certain features of saccades and smooth pursuit eye movements observed in humans [6], [8] as the models with the fovea spent more time in regions with higher image entropy than models with uniform densities.

IV. Data Collection

A Playstation Eye web camera mounted on a PhantomX turret from Trossen Robotics with Dynamixel AX-12A robot actuators was used in the work described here. To obtain data a controller reads the pose of the servos and writes the values to a serial connections when pinged. Videos were collected from the camera at a resolution of 320 by 240 pixels with a frame rate of 120 frames per second using a buffer size of a single frame. Using a buffer size of one frame is key for synchronizing the servo poses and frames, as only the most recent frame is needed when the controller pings for the pan-tilt pose. Data were collected as sets of a thousand frames and corresponding servo positions; fifty sets (50,000 frames) were used for model training and fifty different sets were used for testing. For each data set, the turret motion was randomly selected from a set of twenty pan and twenty tilt predefined trajectories (four hundred possible trajectories in total) involving oscillation, reflective motion and resetting position, running at different rates and in reverse. Each frame of the video was scaled down to 128 by 96 pixels for training and testing. All video was recorded using the wider field of view setting of 75° on the camera.

V. Incorporating Motion into the Prediction

The work on the error saccade model and fovea described in section III motivated us to create a physical demonstration using the camera and turret previously described. We quickly learned that the movement of the camera induced errors in the prediction from the PVM that would influence its own motion, causing it to very frequently get caught in fixed trajectory loops. With the turret having an unloaded rotation speed of 300°/sec and the phenomenon of saccades we were emulating

achieving speeds in excess of that (1000°/sec), we knew that integrating the effect of the motion into the visual prediction was necessary to prevent the endless loops and improve overall performance.

We decided to use a high frame rate (120 frames/sec) sufficient to remove motion blur and to enable us to focus on the effects of perspective changes caused by the movement of the camera. In general, the turret creates both displacement and rotation of the camera, but in the model described here we ignore the effects of camera displacement. By ignoring displacement in camera position we also ignore any effects of occlusion as the camera moves, such as moving beyond a corner and seeing something behind it. With this approximation, the change in perspective can be described as a rotation in three dimensions. We choose the x and y directions in the image plane (positive y-axis being oriented down the vertical with positive x-axis directed right along horizontal direction), and the z-direction along the center line of the camera's field of view. The transformation $T(\phi_2, \theta_2; \phi_1, \theta_1)$ can be expressed in terms of Euler angles depending only on the initial pan and tilt angles θ_1 and ϕ_1, respectively and the final pose's pan and tilt angles θ_2 and ϕ_2.

$$T(\theta_2, \phi_2; \theta_1, \phi_1) = R_x(\phi_2) R_y(\theta_2) R_y^{-1}(\theta_1) R_x^{-1}(\phi_1) = R_x(\phi_2) R_y(\theta_2 - \theta_1) R_x^{-1}(\phi_1) \quad (1)$$

Where R_x and R_y denote a rotation about the x or y axis respectively. Explicitly these rotation matrices are written as

$$R_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & \sin(\alpha) \\ 0 & -\sin(\alpha) & \cos(\alpha) \end{pmatrix} \quad (2)$$

$$R_y(\alpha) = \begin{pmatrix} \cos(\alpha) & 0 & \sin(\alpha) \\ 0 & 1 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) \end{pmatrix} \quad (3)$$

Eq. (1) can be written as

$$\begin{pmatrix} \cos\theta_{21} & \sin\phi_1 \sin\theta_{21} & -\cos\phi_1 \sin\theta_{21} \\ -\sin\phi_2 \sin\theta_{21} & Q & \\ \cos\phi_2 \sin\theta_{21} & & \end{pmatrix} \quad (4)$$

$$Q = \cos\theta_{21} \begin{pmatrix} \sin\phi_2 \sin\phi_1 & -\sin\phi_2 \cos\phi_1 \\ -\sin\phi_1 \cos\phi_2 & \cos\phi_1 \cos\phi_2 \end{pmatrix} + \begin{pmatrix} \cos\phi_2 \cos\phi_1 & \cos\phi_2 \sin\phi_1 \\ \sin\phi_2 \cos\phi_1 & \sin\phi_2 \sin\phi_1 \end{pmatrix} \quad (5)$$

with $\theta_{21} = \theta_2 - \theta_1$. This gives us the affine transformation on the horizontal and vertical positions (after rescaling by the focal length in it's pixel equivalent) of each pixel in the input image by applying the transformation in Eq. (1) gives,

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = T(\theta_2, \phi_2; \theta_1, \phi_1) \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}. \quad (6)$$

As the mapping does not map directly onto pixel positions, interpolation can be used to find the appropriate pixel value or the pixel position can be rounded instead. The later method was used instead of interpolation, which may cause the transformed image sent to the PVM and the respective prediction to looked as if it is aliased. As the original image is translated, rotated and rescaled some pixels are not mapped from the input to the transform image leaving it incomplete. These blank pixels are filled by extending the edges of the original image such that they fill the entire frame.

VI. Results and Discussion

Two identically structured PVMs were constructed to compare the effect of integrating motion on the PVM. The structure is an eight level hierarchy of rectangular grids of PVM units with size 64 by 48, 32 by 24, 16 by 12, 8 by 6, 4 by 3, 3 by 2, 2 by 1 and 1 by 1 with each unit having a hidden size of 5 and the top most unit sending context to all lower units. The input into the model is a 128 by 96 pixel video sequence where 2 by 2 tiles of each frame are fed into to the corresponding unit in the lowest level of the PVM hierarchy. The mean square error versus frame count in the PVM without any motion integration is shown in Fig. 2 and the PVM with motion integration is shown in Fig. 3 for both the training and testing sets smoothed across a window of one epoch (50,000 frames). The beginning and end of the training data sets are zero padded with 25,000 frames to maintain the total frame count (resulting in artifacts at the beginning and end of the training error curves in Fig. 2 and Fig. 3). Comparing the two models shows that motion integration substantially improves the PVM's ability to predict the next video frame. Qualitatively, predicted image features are sharper and colors are less faded with motion integration than without it. Improved performance may also be due to permutation of noise due to the transformation [9]. Both models struggle with predictions when very fast movements are involved as even the model with motion integration has no way to predict what lies outside it's field of view. Motion integration improving prediction should not be unique to just the PVM it should improve any other self-supervised visual prediction model [10], [11].

We applied the error saccade model with the PVM on the turret in a demonstration to compare the PVM with and without motion integration. One of the main issues with the behavior of the turret movement is that it can get caught in an infinite loop on a static background as a rapid back and forth between two fixation points is possible due to the induced errors caused by the movement of the camera. This is demonstrated in this video https://bit.ly/2QbEfgs where the turret keeps saccading between the two tables inducing high prediction errors. When motion is integrated into the model prediction the movement is more robust to endless loops, as seen in https://bit.ly/2Wg0w20.

VII. Conclusion

Here we have demonstrated an improved close-loop vision system that interacts and models its environment in an self-

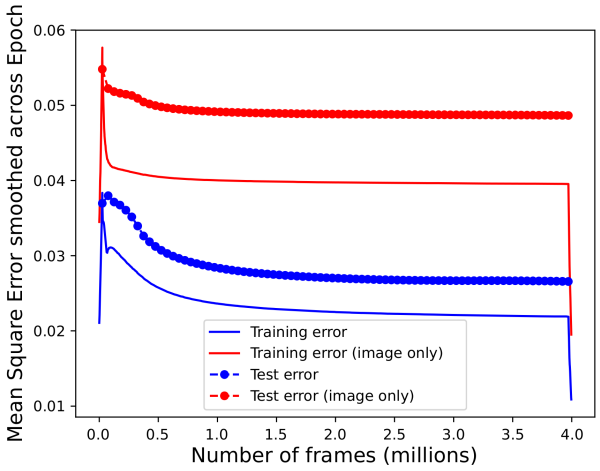


Fig. 2. Training and testing error for the PVM trained without any motion integration. The training error is smoothed over an epoch's worth of frames with zero padding (50,000 frames) for comparison with mean testing error after that epoch. The values in red are the mean squared error for the prediction of the image only, in other words the prediction error of just the lowest level of the PVM hierarchy. The values in blue are the mean squared error for all levels of the hierarchy.

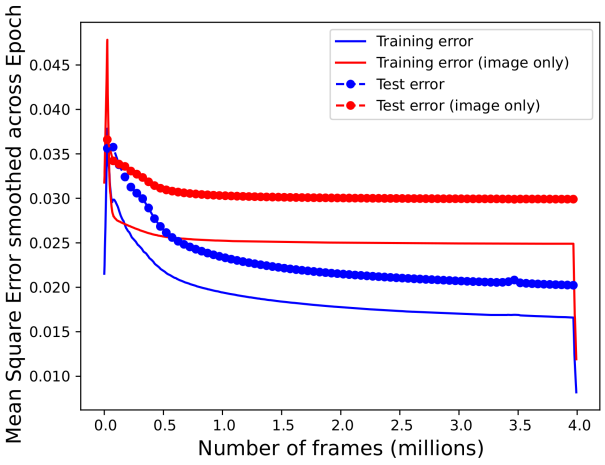


Fig. 3. Training and testing error for the PVM train with motion integration. The training error is smoothed over an epoch's worth of frames with zero padding (50,000 frames) for comparison with mean testing error after that epoch. The values in red are the mean squared error for the prediction of the image only, in other words the prediction error of just the lowest level of the PVM hierarchy. The values in blue are the mean squared error for all levels of the hierarchy.

supervised fashion. The addition of motion integration substantially improved the PVM's prediction ability and when combined with the error saccade model does not have the pathology of being caught in endless loops. We speculate that stereoscopic vision with vergence movements would have the advantage of being able to judge distances and improve the predictive ability of a vision model especially when movement and occlusion are relevant as a three dimensional representation of a scene would be more informative than a representation from a single two dimensional image.

References

[1] V. Mnih, N. Heess, A. Graves, and K. Kavukcuoglu, "Recurrent models of visual attention," *CoRR*, vol. abs/1406.6247, 2014. [Online]. Available: http://arxiv.org/abs/1406.6247

[2] M. Campbell, A. J. Hoane Jr, and F.-h. Hsu, "Deep blue," *Artificial intelligence*, vol. 134, no. 1-2, pp. 57–83, 2002.

[3] B. L. Lewis, "In the game: The interface between watson and jeopardy!" *IBM Journal of Research and Development*, vol. 56, no. 3.4, pp. 17–1, 2012.

[4] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," *ArXiv e-prints*, Dec. 2013.

[6] M. Hazoglou and T. Hylton, "Saccadic predictive vision model with a fovea," in *Proceedings of the International Conference on Neuromorphic Systems*, ser. ICONS '18. New York, NY, USA: ACM, 2018, pp. 2:1–2:6. [Online]. Available: http://doi.acm.org/10.1145/3229884.3229886

[7] F. Piekniewski, P. A. Laurent, C. Petre, M. Richert, D. Fisher, and T. Hylton, "Unsupervised learning from continuous video in a scalable predictive recurrent network," *CoRR*, vol. abs/1607.06854, 2016. [Online]. Available: http://arxiv.org/abs/1607.06854

[8] A. L. Yarbus, "Eye movements during perception of complex objects," in *Eye movements and vision*. Springer, 1967, pp. 171–211.

[9] D. D. Thang and T. Matsui, "Image transformation can make neural networks more robust against adversarial examples," *CoRR*, vol. abs/1901.03037, 2019. [Online]. Available: http://arxiv.org/abs/1901.03037

[10] W. Lotter, G. Kreiman, and D. D. Cox, "Deep predictive coding networks for video prediction and unsupervised learning," *CoRR*, vol. abs/1605.08104, 2016. [Online]. Available: http://arxiv.org/abs/1605.08104

[11] A. Canziani and E. Culurciello, "Cortexnet: a generic network family for robust visual temporal representations," *CoRR*, vol. abs/1706.02735, 2017. [Online]. Available: http://arxiv.org/abs/1706.02735

Fast solution of linear systems with analog resistive switching memory (RRAM)

Zhong Sun
DEIB, Politecnico di Milano and IU.NET
Milano, Italy
zhong.sun@polimi.it

Giacomo Pedretti
DEIB, Politecnico di Milano and IU.NET
Milano, Italy
giacomo.pedretti@polimi.it

Daniele Ielmini
DEIB, Politecnico di Milano and IU.NET
Milano, Italy
daniele.ielmini@polimi.it

***Abstract*—The in-memory solution of linear systems with analog resistive switching memory in one computational step has been recently reported. In this work, we investigate the time complexity of solving linear systems with the circuit, based on the feedback theory of amplifiers. The result shows that the computing time is explicitly independent on the problem size *N*, rather it is dominated by the minimal eigenvalue of an associated matrix. By addressing the Toeplitz matrix and the Wishart matrix, we show that the computing time increases with log(*N*) or $N^{1/2}$, respectively, thus indicating a significant speed-up of in-memory computing over classical digital computing for solving linear systems. For sparse positive-definite matrix that is targeted by a quantum computing algorithm, the in-memory computing circuit also shows a computing time superiority. These results support in-memory computing as a strong candidate for fast and energy-efficient accelerators of big data analytics and machine learning.**

Keywords—in-memory computing, linear system, resistive memory, time complexity

I. INTRODUCTION

In recent years, in-memory computing with crosspoint resistive memory array has gained an enormous attention, thanks to its potential in providing a high time/energy efficiency for a wide range of data-intensive computations [1]-[6]. By exploiting analog physical rules and massive parallelism in crosspoint architecture, arrays of resistive switching memory (RRAM) devices have been extensively used to accelerate matrix-vector multiplication (MVM) in various applications, such as training and inference of deep neural networks [3], [4], signal and image processing [5], [6], and iterative solution of linear systems [7]. Recently, a crosspoint RRAM circuit has also been shown to solve linear systems in one step, by combining feedback loops in the circuit with operational amplifiers [8]. A single-crosspoint-array circuit solves linear systems of positive matrix, while two crosspoint arrays can address general matrices containing positive, negative and null entries. The ability to solve linear problems in one step is valuable in numerous computing scenarios, such as solving differential equations and calculating eigenvectors, *e.g.*, for PageRank. To benchmark the circuit performance in practical implementations, the computing time complexity of the circuit should be assessed.

II. TIME COMPLEXITY OF SOLVING POSITIVE LINEAR SYSTEMS

A. Crosspoint resistive memory circuit

Solving a linear system means to find out the unknown vector x in the equation

$$Ax = b, \tag{1}$$

where A is an invertible square matrix, b is a known vector. Eq. (1) can be conveniently solved in one step by the circuit shown in Fig. 1a, where the crosspoint RRAM array stores matrix A, and the input voltages V_{in} represent the vector $-b$. The crosspoint rows and columns are connected by operational amplifiers (OAs), which provide a virtual ground at the rows through negative feedback. As a result, the output voltages V_{out} have to satisfy the equation by Kirchhoff's current law

$$AV_{out} + V_{in} = 0, \tag{2}$$

which implies that the output voltages constitute the solution to Eq. (1) with $V_{out} = -A^{-1}V_{in} = A^{-1}b$.

B. Time complexity analysis

The circuit in Fig. 1a is able to solve a linear system in one step, with the steady-state output voltages of OAs as solution. To study the circuit dynamic, it is illustrated as a feedback control system shown as a block diagram in Fig. 1b. The crosspoint matrix A plays the role of a feedback network, which weights the output vector x to interact with the input vector b, and the net result is transferred by OAs. According to Kirchhoff's voltage law, the system can be described by an equation in terms of Laplace transformation, namely

$$-U[Ax(s) - b(s)]L(s) = x(s), \tag{3}$$

where $L(s)$ is the open loop gain of OA, and U is a diagonal matrix as $U = diag\left(\frac{1}{1+\sum_i A_{1i}}, \frac{1}{1+\sum_i A_{2i}}, \cdots, \frac{1}{1+\sum_i A_{Ni}}\right)$. For a single-pole OA, namely $L(s) = \frac{L_0}{1+\frac{s}{\omega_0}}$ [9], where L_0 is the DC open loop gain, ω_0 is the 3-dB bandwidth, Eq. (3) becomes

$$x(s) = -L_0\omega_0 U[Ax(s) - b(s)]. \tag{4}$$

The Laplace transformation of Eq. (4) results in a differential equation in time domain, which is

$$\frac{dx(t)}{dt} = -L_0\omega_0 U[Ax(t) - b(t)]. \tag{5}$$

Eq. (5) describes the circuit dynamic towards the steady-state solution. To analyze the computing time, Eq. (5) is converted into a finite difference (FD) algorithm, which reads

$$x(t + \Delta t) = \alpha Ub + (I - \alpha M)x(t), \tag{6}$$

where Δt is the incremental time, α is a small number equal to $L_0\omega_0\Delta t$, M is a matrix defined as $M = UA$, and I is the identity matrix. For the algorithm to be convergent, the spectral radius ρ of matrix $I - \alpha M$ has to be less than 1, which directly implies the minimal eigenvalue (or real part of eigenvalue) of the associated matrix M ($\lambda_{M,min}$) must be positive. Such a requirement is perfectly met by a positive definite (PD) matrix A, thanks to the positive diagonal matrix U.

This article has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 648635).

978-1-7281-5221-9/19/$31.00 ©2019 IEEE

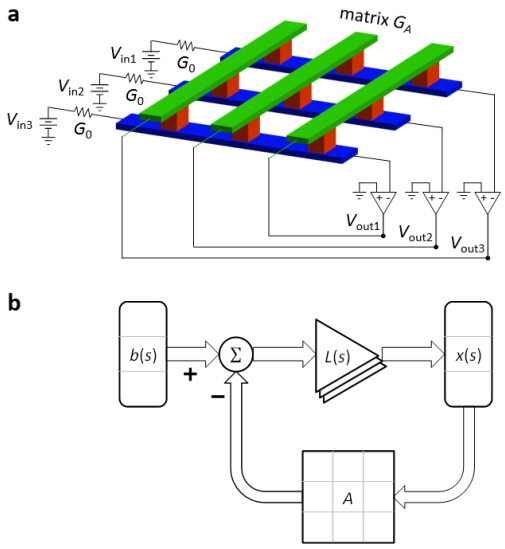


Fig. 1. (a) Crosspoint RRAM circuit for solving linear systems of positive matrix. Matrix A is stored in the crosspoint array with $A = G_A/G_0$. (b) Block diagram of the circuit as a feedback control system.

The FD algorithm is verified by comparing with transient analysis of the circuit in SPICE (Fig. 2). Specifically, 8 discrete conductance levels (120 μS, 80 μS, 60 μS, 50 μS, 30 μS, 20 μS, 15 μS, 10 μS) of RRAM devices from [8] were employed to construct a matrix randomly, under the $\lambda_{M,min}$ constraint. Solving a 3×3 linear system is shown in Fig. 2a, including the matrix normalized by the maximum conductance, an input vector b and the resulting evolvement of solution x. The FD algorithm and circuit simulation results of x are plotted in parallel, showing a high consistency. A 10×10 matrix has also been tested, with the conductance matrix, the input vector and the output dynamic shown in Fig. 2b. Again, the FD algorithm precisely traces the circuit simulation results, thus validating the capability of the algorithm for representing the circuit dynamic and hence for investigating the time complexity.

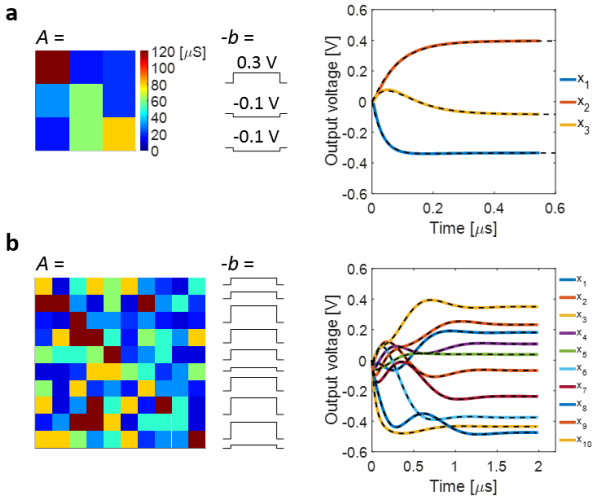


Fig. 2. Simulated solution dynamics of linear systems. (a) A 3×3 linear system case, including the implemented matrix A, the input vector b and the resulting dynamic of solution x. Color lines: transient output voltages in SPICE simulation. Black dash lines: FD algorithm. (b) Same as (a), but for a 10×10 linear system.

The time complexity can be deduced from the convergence analysis of the FD algorithm. The convergence is measured in the A-norm, which is defined as $\|x\|_A = \sqrt{x^T A x}$ [10]. If a linear system is solved with an accuracy ϵ at time t, the A-norm of solution error has to be satisfy

$$\|x(t) - x^*\|_A \le \epsilon, \tag{7}$$

where $x^* = A^{-1}b$ is the precise solution. $\|x(t) - x^*\|_A$ follows

$$\begin{aligned} \|x(t) - x^*\|_A^2 &= \|\alpha U b + (I - \alpha M)x(t - \Delta t) - x^*\|_A^2 \\ &= \|(I - \alpha M)x(t - \Delta t) - (I - \alpha M)x^*\|_A^2 \\ &\le \|(I - \alpha M)\|^2 \|x(t - \Delta t) - x^*\|_A^2 \\ &= \left(1 - \alpha\lambda_{M,min}\right)^2 \|x(t - \Delta t) - x^*\|_A^2 \end{aligned} \tag{8}$$

Repeating the above process iteratively leads to

$$\begin{aligned} \|x(t) - x^*\|_A^2 &\le \left(1 - \alpha\lambda_{M,min}\right)^{2t/\Delta t} \|x(0) - x^*\|_A^2 \\ &< e^{-2\lambda_{M,min} L_0 w_0 t} \|x^*\|_A^2 \\ &= e^{-2\lambda_{M,min} L_0 w_0 t} x^{*T} b \end{aligned} \tag{9}$$

The upper bound of $\|x(t) - x^*\|_A$ satisfying inequation (7) finally reveals the computing time as

$$t \ge \frac{1}{\lambda_{M,min} L_0 w_0} ln \frac{\sqrt{x^{*T} b}}{\epsilon}, \tag{10}$$

which tells the time complexity of solving linear systems with the crosspoint circuit is explicitly independent on matrix size N, instead it is dominated by $\lambda_{M,min}$ of the associated matrix M.

Since U is a diagonal matrix that reflects row sums of matrix A, $\lambda_{M,min}$ is approximately proportional to λ_{min} for matrices with the same size. Therefore, the computing time will scale as $t \propto \frac{1}{\lambda_{min}}$ for a specific N. A series of 3×3 PD matrices with different λ_{min} were generated using the 8 conductance levels. For each matrix, input vectors b were randomly generated and hence the corresponding linear systems were solved by the circuit, with the computing times recorded. The summarized results in Fig. 3a show that the time of solving a linear system increases inversely with the λ_{min} of the matrix. In Fig. 3b, the computing time shows a more precise linear dependence on

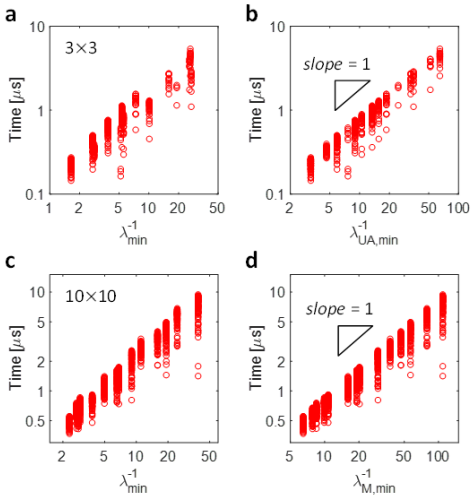


Fig. 3. (a) Computing time for solving linear systems of 3×3 PD matrices with various λ_{min}. (b) Computing time plotted against $\lambda_{M,min}$ of the associated matrix M. (c) Same as (a), but for a series of 10×10 PD matrices. (d) Same as (b), but for the 10×10 matrices.

$1/\lambda_{M,min}$, thus verifying the time complexity in Eq. (10). A series of 10×10 PD matrices have also been tested. Again, while the time of solving a linear system increases roughly with $1/\lambda_{min}$ (Fig. 3c), it shows a more precise linear dependence on $1/\lambda_{M,min}$ (Fig. 3d).

To study the dependence of computing time on the matrix size N, we solved linear systems of a Toeplitz matrix [11], which is defined according to $A_{ij} = \frac{1}{|i-j|+1}, i,j = 1,2,\dots,N$. As N increases, the minimal eigenvalue λ_{min} of the Toeplitz matrix is asymptotically constant, while $\lambda_{M,min}$ decreases linearly with log(N) (Fig. 4a). As a result, the computing time of solving linear systems increases linearly with log(N), namely the time complexity is O(log$N$) (Fig. 4b). In conventional computers, the Toeplitz systems can be solved with the Levinson algorithm, whose time complexity is $O(N^2)$ [11]. Therefore, our approach provides an exponential speed-up for solving such linear systems compared to conventional digital computing.

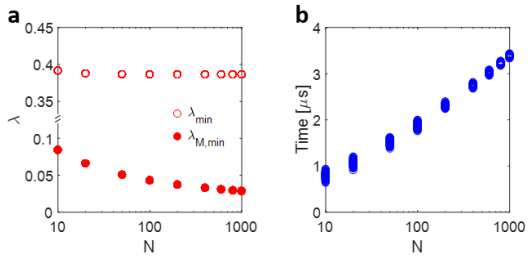


Fig. 4. (a) λ_{min} of the Toeplitz matrix with various sizes, and $\lambda_{M,min}$ of the associated matrices. (b) Computing time of solving linear systems of the Toeplitz matrices. 100 random input vectors b were generated for each size, forming linear systems to be solved. The resulting computing time indicates an O(log$N$) time complexity.

The robustness of the circuit against device variation and analog noise is related to the condition number (κ) of the stored matrix [8]. A well-conditioned matrix whose κ is small can tolerate the circuit imperfections, as the matrix property is stable. Only if the $\lambda_{M,min}$ is positive for a deviated matrix, the solution should be convergent. In Fig. 5, a 100×100 Toeplitz system is solved with or without device deviations in the crosspoint array. Both sets of solution, *i.e.* the 100 output voltages converge with a slight difference of computing time, and the steady-state solution shows relatively small errors. Specifically, the computing time for solving the ideal linear system is 2.126 μs, while it is 2.174 μs for solving the deviated one, which is explained by the reduced $\lambda_{M,min}$ for the latter case.

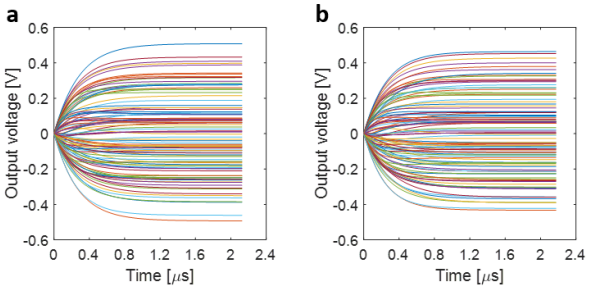


Fig. 5. (a) Transient behavior of solving a 100×100 Toeplitz system without device variations, whose condition number is 19.6. (b) Transient behavior of solving the same linear system, but each crosspoint device is of a variation within ±5%. The $\lambda_{M,min}$ is 0.0429 and 0.0408 for the ideal and the deviated matrix, respectively.

III. TIMECOMPLEXITY OF SOLVING MIXED LINEAR SYSTEMS

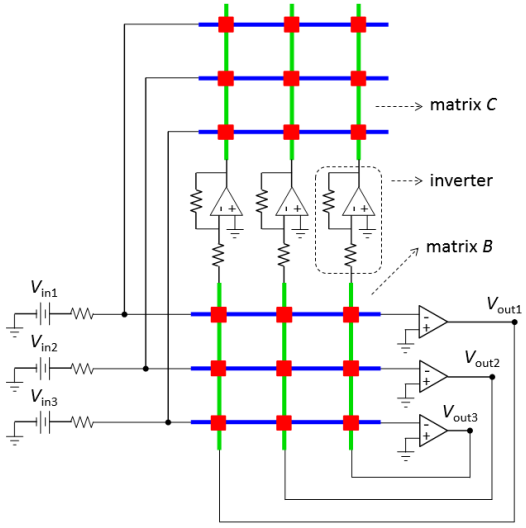


Fig. 6. Crosspoint resistive memory circuit for solving linear systems of general real matrix. Two crosspoint arrays are used to implement the original matrix A, with $A = B - C$.

A. Dual-crosspoint-array circuit

To address mixed matrices containing negative elements, a general circuit was also proposed [8], as shown in Fig. 6. A mixed matrix A can always be split by two positive matrices B and C, namely $A = B - C$. In the circuit, matrices B and C are mapped in two crosspoint RRAM arrays, respectively, and the minus operation is realized by analog inverters between columns of the two arrays. By considering the virtual ground at row lines and applying Kirchhoff's current law, namely $BV_{out} - CV_{out} + V_{in} = 0$, one can easily find out that the output voltages of OAs constitute the solution to a linear system of matrix A.

B. Time complexity analysis

To investigate time complexity of the circuit in Fig. 6, we also analyze the circuit dynamics based on feedback theory of amplifiers. Term the output voltages of OAs and inverters x and y, respectively, and $-V_{in}$ is represented by b, then the following two equations are obtained

$$-U[Bx(s) + Cy(s) - b(s)]L(s) = x(s), \qquad (11\text{-}1)$$

$$-\frac{x(s)+y(s)}{2}L(s) = y(s), \qquad (11\text{-}2)$$

where $U = diag\left(\frac{1}{1+\sum_i B_{1i}+\sum_i C_{1i}}, \frac{1}{1+\sum_i B_{2i}+\sum_i C_{2i}}, \cdots, \frac{1}{1+\sum_i B_{Ni}+\sum_i C_{Ni}}\right)$, and the identical open loop gain $L(s)$ is assumed for all OAs in the circuit. Combining the two equations in Eq. (11) leads to

$$-U\left[Bx(s) - \frac{L(s)}{L(s)+2}Cx(s) - b(s)\right]L(s) = x(s). \qquad (12)$$

By considering a single-pole OA model [9], and then applying Laplace transformation, a second-order differential equation describing the circuit dynamic is obtained,

$$\frac{d^2x(t)}{dt^2} = -\frac{L_0\omega_0}{2}\left[(2UB + I)\frac{dx(t)}{dt} + L_0\omega_0 UAx(t) - L_0\omega_0 Ub\right]. \qquad (13)$$

To analyze time complexity of the circuit, Eq. (13) is converted as a first-order differential equation by assuming

$$z(t) = -\frac{2}{L_0\omega_0}\frac{dx(t)}{dt}. \qquad (14\text{-}1)$$

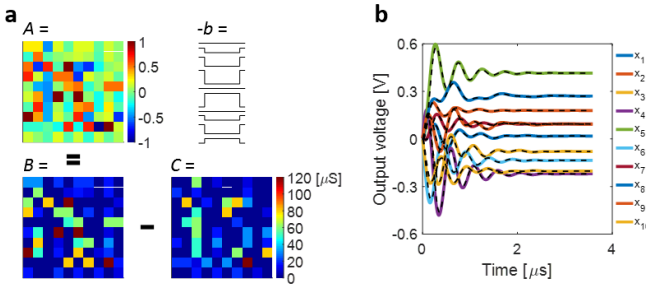


Fig. 7. (a) A 10×10 mixed matrix A and two positive matrices B and C splitting it. Matrices B and C are stored in the two crosspoint arrays in the circuit of Fig. 6, respectively. An input vector b is also provided to form a linear system to be solved by the circuit. (b) Solution dynamic of the constructed 10×10 linear system. Color lines: transient output voltages in SPICE. Black lines: FD algorithm.

As a result, Eq. (13) becomes

$$\frac{dz(t)}{dt} = -\frac{L_0\omega_0}{2}[(2UB + I)z(t) - Ax(t) + Ub]. \qquad (14\text{-}2)$$

The two equations in Eq. (14) can be merged as

$$\frac{d}{dt}\begin{bmatrix}x(t)\\z(t)\end{bmatrix} = -L_0\omega_0\left\{\begin{bmatrix}0 & I/2\\-UA & UB + I/2\end{bmatrix}\begin{bmatrix}x(t)\\z(t)\end{bmatrix} + \begin{bmatrix}0\\Ub\end{bmatrix}\right\}, \qquad (15)$$

Defining $M = \begin{bmatrix}0 & I/2\\-UA & UB + I/2\end{bmatrix}$ as the associated matrix with a size of $2N\times2N$, $X(t) = \begin{bmatrix}x(t)\\z(t)\end{bmatrix}$ and $J = \begin{bmatrix}0\\Ub\end{bmatrix}$, whose sizes are $2N\times1$, Eq. (15) finally becomes

$$X(t + \Delta t) = -\alpha J + (I - \alpha M)X(t), \qquad (16)$$

where I is the $2N\times2N$ identity matrix, α is equal to $L_0\omega_0\Delta t$.

Eq. (16) is same as Eq. (6), hence the discussions for solving positive linear systems can be applied to solving mixed ones. Consequently, the $\lambda_{M,min}$ of matrix M must also be positive. The FD algorithm in Eq. (16) is verified by comparing with SPICE transient simulation for solving a 10×10 mixed linear system. In Fig. 7a, a mixed matrix is randomly generated under the $\lambda_{M,min}$ constraint, then it is split by two positive matrices, which are implemented by the 8 conductance levels. An input vector is also provided to form a linear system to be solved. In Fig. 7b, both SPICE simulation and iterative algorithm results of the solution are plotted in parallel, indicating a precise description of the circuit dynamic by Eq. (16).

C. Solving linear systems of Wishart matrix

The time complexity of solving linear systems of mixed matrix should also be solely determined by $\lambda_{M,min}$. As a case study, solving linear systems of a sample covariance matrix named Wishart matrix is addressed. Wishart matrix is a random matrix defined as $W = \frac{RR^T}{N}$, where R is a $S\times N$ matrix whose entries are IID random variables with zero mean ($\mu = 1$) and one variance ($\sigma = 1$) [12]. Apparently W is a mixed matrix containing both positive and negative entries, as shown in Fig. 8a for a 100×100 Wishart matrix. When solving a linear system of Wishart matrix, W is split by two positive matrices B and C, to be mapped by the two crosspoint arrays in the circuit, respectively. Matrix B is element-wise composed of the positive entries in W, and the other entries are assigned a small number, *e.g.* 10^{-4}, which can be represented by the high resistance state of RRAM device. Matrix C is then calculated according to $W = B - C$.

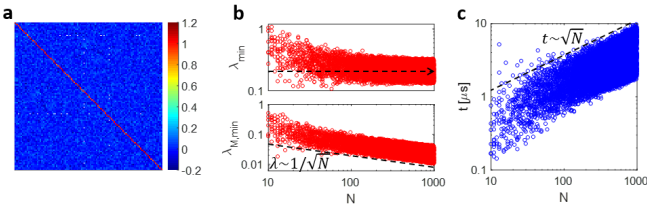


Fig. 8. Computing time of solving linear systems of Wishart matrices with various sizes. (a) A representative 100×100 Wishart matrix. (b) Top panel: λ_{min} of 10,000 randomly-generated Wishart matrices, with sizes from 10×10 to 1,000×1,000. The black dash line indicates that λ_{min} converges to 0.2. Bottom panel: $\lambda_{M,min}$ of the associated matrices, plotted as a function of N. The black dash line evidences that $\lambda_{M,min}$ decreases linearly with $1/\sqrt{N}$. (c) Computing time of solving linear systems of the 10,000 Wishart matrices. The black dash line reveals the computing time increases linearly with $\sqrt{N}$.

It is stated that the λ_{min} of W converges almost surely to $\left(1 - \sqrt{y}\right)^2$ as $N \to \infty$ with $s/N \to y \in (0,1)$ [12]. Such a property is illustrated in the top panel of Fig. 8b, where $y = 0.3$ is assumed and hence λ_{min} is supposed to converge to 0.2. $\lambda_{M,min}$ is also calculated in parallel for each matrix, showing a scaling of $\lambda_{M,min}\sim1/\sqrt{N}$ (bottom panel of Fig. 8b). Together with the generation of a Wishart matrix, a random input vector is also created for solving a linear system by the circuit. The computing time was recorded for each trial, and all the results are summarized in Fig. 8c. As can be observed, computing time increases with $\sqrt{N}$, thus demonstrating the validity of Eq. (10) as time complexity of solving linear system of mixed matrix by the dual-crosspoint-array circuit. Note that the time complexity of classical algorithms, such as Gaussian elimination for solving linear systems in conventional digital computers is $O(N^3)$. Therefore, the $O(N^{1/2})$ time complexity of the crosspoint circuit represents a significant speed-up for solving linear systems in memory.

IV. Comparison with Digital and Quantum Computing

For the matrix product $M = UA$, there is an inequality describing the minimal eigenvalues, namely $\lambda_{M,min} \geq U_{min}\lambda_{min}$, where U_{min} is the minimal eigenvalue, *i.e.*, the minimal element of matrix U. As U_{min} is determined by the maximum row sum of U, the lower bound of $\lambda_{M,min}$ is expected to decrease with N for dense matrices, and thus imposing an N-dependence to time complexity. However, this is not the situation for sparse matrix. Here we concern s-sparse positive-definite matrix, which contains at most s non-zero elements per row. In this case, $\lambda_{M,min} \geq U_{min}\lambda_{min}\sim\frac{\lambda_{min}}{s}$ does not rely on matrix size N, and hence the time complexity of solving linear systems will be $O\left(\frac{s}{\lambda_{min}} ln\frac{1}{\epsilon}\right)$. In conventional computers, the most efficient algorithm for solving linear systems of positive-definite matrix is the conjugate gradient (CG) method that owns a linear time complexity for sparse matrix [13]. A quantum computing algorithm has also been proposed for solving linear systems with a logarithmic time complexity [14], as summarized in details in Table I. Therefore, compared to both digital and quantum computing, the crosspoint RRAM circuit shows a computing time superiority to extremely speed up solving linear systems of sparse positive-definite matrix, which is encountered in a wide range of practical applications [15].

TABLE I. TIME COMPLEXITY COMPARISON

CG method	Quantum computing	In-memory computing
$O\left(Ns\sqrt{\frac{\lambda_{max}}{\lambda_{min}}}ln\frac{1}{\epsilon}\right)$	$O\left(\frac{s^2{\lambda_{max}}^2}{\epsilon{\lambda_{min}}^2}lnN\right)$	$O\left(\frac{s}{\lambda_{min}}ln\frac{1}{\epsilon}\right)$

CONCLUSION

In this work, we study the circuit dynamics based on the feedback theory of amplifier circuits. The results show that, for both single- and dual-crosspoint circuits, the time complexity is solely dictated by the minimal eigenvalue of an associated matrix. For a representative Toeplitz matrix, the time complexity of solving linear systems is shown to be $O(\log N)$, while for the Wishart matrix dataset, the time complexity is $O(N^{1/2})$. These results demonstrate a significant speed-up over the polynomial time complexity of classical digital computers. For sparse linear systems, the time complexity is reasoned as N-independent, evidencing a higher time efficiency than the quantum computing algorithm, thus strongly supporting in-memory computing as a promising approach for data analysis and machine learning.

REFERENCES

[1] D. Ielmini and H. -S. P. Wong, "In-memory computing with resistive switching devices," *Nature Electronics*, vol. 1, no. pp. 333-343, 2018.

[2] Z. Sun, E. Ambrosi, A. Bricalli, and D. Ielmini, "Logic computing with stateful neural networks of resistive switches," *Advanced Materials*, vol. 30, no. 38, p. 1802554, 2018.

[3] M. Prezioso et al., "Training and operation of an integrated neuromorphic network based on metal-oxide memristors," *Nature*, vol. 521, pp. 61-64, 2015.

[4] P. Yao et al., "Face classification using electronic synapses," *Nature Communications*, vol. 8, p. 15199, 2017.

[5] P. M. Sheridan, F. Cai, C. Du, W. Ma, Z. Zhang, and W. D. Lu, "Sparse coding with memristor networks," *Nature Nanotechnology*, vol. 12, pp. 784-789, 2017.

[6] C. Li et al., "Analogue signal and image processing with large memristor crossbars," *Nature Electronics*, vol. 1, pp. 52-59, 2018.

[7] M. Le Gallo et al., "Mixed-precision in-memory computing," *Nature Electronics*, vol. 1, pp. 246-253, 2018.

[8] Z. Sun et al., "Solving matrix equations in one step with cross-point resistive arrays," *PNAS*, vol. 116, no. 10, pp. 4123-4128, 2019.

[9] B. Razavi, Design of Analog CMOS Integrated Circuits. McGraw-Hill, New York, 2001.

[10] G. H. Golub & C. F. van Loan, Matrix Computations, 4th ed., Johns Hopkins Univ. Press, Baltimore, 2013.

[11] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, Numerical Recipes: The Art of Scientific Computing, 3rd ed. Cambridge University Press, New York, NY, 2007.

[12] J. W. Silverstein, "The Smallest Eigenvalue of a Large Dimensional Wishart Matrix," *The Annals of Probability*, vol. 13, pp. 1364-1368, 1985.

[13] J. R. Shewchuk, "An Introduction to the Conjugate Gradient Method without the Agonizing Pain," *Report CMU-CS-94-125, School of Computer Science*, Carnegie Mellon University, Pittsburgh, 1994.

[14] A. W. Harrow, A. Hassidim, and S. Lloyd, "Quantum Algorithm for Linear Systems of Equations," *Physical Review Letters*, vol. 103, p. 150502, 2009.

[15] R. Bhatia, Positive definite matrices. Princeton Univ. Press, Princeton, NJ, 2007.

Designing Crosstalk Circuits at 7nm

Md Arif Iqbal[1], Naveen Kumar Macha, Bhavana T. Repalle, Mostafizur Rahman[2]
Computer Science & Electrical Engineering, *University of Missouri Kansas City, Missouri*
mibn8@mail.umkc.edu[1], rahmanmo@umkc.edu[2]

Abstract*— Crosstalk is an innovative computing technology that utilizes unwanted interferences between interconnects to compute useful logic. Engineering of coupling capacitance, circuit scheme and integration are core features. This paper presents scalability aspects of Crosstalk technology to compete/co-exist with CMOS for digital logic implementations below 10nm. Scalability is a key requirement for any emerging technologies to continue chip miniaturization. Our scalability study is with Arizona State Predictive (ASAP) 7nm PDK and considers all process variation aspects. We present primitive gate designs and their performance under variations. We discuss design constraints to accommodate worst-case variation scenarios. Finally, utilizing primitive gates, we show larger designs such as cm85a, mux, and pcle from MCNC benchmarking suits and detailed comparison with CMOS at 7nm. Our benchmarking revealed, averaging all three above mentioned circuits, 48%, 57% and 10% improvements against CMOS designs in terms of transistor count, power and performance respectively.

Keywords—Crosstalk Computing, Large-scale Circuit Design, Process Variation, ASAP 7nm PDK

I. Introduction

Crosstalk computing provides a scalable alternative solution to CMOS while leveraging CMOS devices and interconnect technologies [1]-[5]. In Crosstalk computing, interference between the metal lines at advanced technology nodes is engineered in an efficient manner to obtain logic functions. Nano metal lines are placed in a compact manner so that whenever signal transitions take place in one of the lines the sum of their Crosstalk interference gets induced in another metal line through coupling capacitance. The strength of the coupling capacitance determines how the charge is going to be induced on the victim metal line and thereby can be engineered to obtain different logic functions. The key components of the Crosstalk computing fabric are metal lines, coupling capacitances, a synchronous clock, and inverter. The coupling capacitances between aggressors and the victim are inversely proportional to the separation of metal lines and directly proportional to the permittivity of the dielectric and lateral area of metal lines, which can be engineered according to required logic function.

Like any other emerging technology, scalability study is also a key requirement for Crosstalk computing. However, to completely assess the effectiveness of any new design methodologies at advanced technology node, a standard process design kit (PDK) with the full set of collateral necessary for schematic entry, layout, design rule checking, parasitic extraction, transistor-level simulation, library generation, synthesis, and automatic placement and routing (APR) is required.

In this paper, we do scalability study with Arizona State Predictive 7nm PDK (ASAP7) [7]. ASAP7 PDK comes with predictive technology models for transistors and design collaterals including libraries and technology files that are required by the CAD tools. Through worst-case process variation analysis, we demonstrate that even at sub 10nm, Crosstalk logic gates function properly. Using these primitive logic gates, we designed larger circuits like cm85a, mux, and pcle from MCNC benchmark suite [8] and compared the results with CMOS at 7nm. Our comparison results show 59%, 62% and 23% reduction in transistor count for cm85a. mux and pcle circuits, respectively. Our simulation results also show potentials for power and performance improvements; on average for the three circuits, reduction in power and latency/performance was 57% and 10%.

The rest of the paper is organized as follows. Section II discusses scalability and Crosstalk Computing Concept and presents implementations of basic logic circuits. Section III discusses the impact of process variation in Crosstalk Computing at 7nm. The behavior of Crosstalk NAND and NOR gate for different process corners is also shown here. Section IV explains the design aspects for larger-scale using Crosstalk logic cells at 7nm. Section V gives the benchmarking results. Finally, Section VI concludes the paper.

II. Scaling Challenges & Crosstalk Design Aspects

Scalability is a major concern for CMOS, and in order for any emerging technologies to compete/co-exist with CMOS the scalability litmus test need to be passed. For CMOS, the scaling has been driven by shrinking transistors first, and challenges are mainly due to the reduction in transistor drive strength, manufacturability and interconnection overhead. Crosstalk computing, utilizing CMOS transistors and processes, proposes an alternative paradigm which puts emphasis on circuit design and integration first. Our benchmarking shows that through design 2-5x density benefits are possible over CMOS with improved power at the same node. Although the prospects are promising, one may question how variability and signal integrity challenges can be passed by a technology that relies on noise. To answer this question, we first discuss Crosstalk's computing aspects and then detail design constraints.

Fig.1 shows the example of two primitive cells (NAND & NOR) constructed in Crosstalk fabric. In any logic cell, the underlying principle is to emulate the behavior of aggressor-victim commonly found in interconnects. During logic computation, the victim net (*Vi*) voltage is controlled electrostatically through coupling capacitances between two aggressors (*Ag1* and *Ag2*) and victim (*Vi*) net. To drive the *Vi* node for next round of logic evaluation, its voltage is discharged to ground through a transistor controlled by *dis* signal after every round of logic evaluation. The *dis* signal also ensures synchronization with the rest of the circuits. Thus, the *Vi* node is connected to an inverter on one end and connected to the drain side of the discharge transistor on the other end. After every computation, the *dis* signal will be turned ON to discharge the *Vi* node. Initially, the *Vi* node is kept floating at 0 (because of the previous discharge cycle), and when the input transitions in *A* and *B* occurs, output summation charge is induced on *Vi*, which in turn drives the inverter acting as a threshold function. This same principle is used while implementing both NAND and NOR gates with the only difference of coupling strengths between inputs and *Vi* net. For NOR gate, the coupling (C_{NR}) is stronger than NAND gates (C_{ND}) and is chosen such that whenever any of the inputs

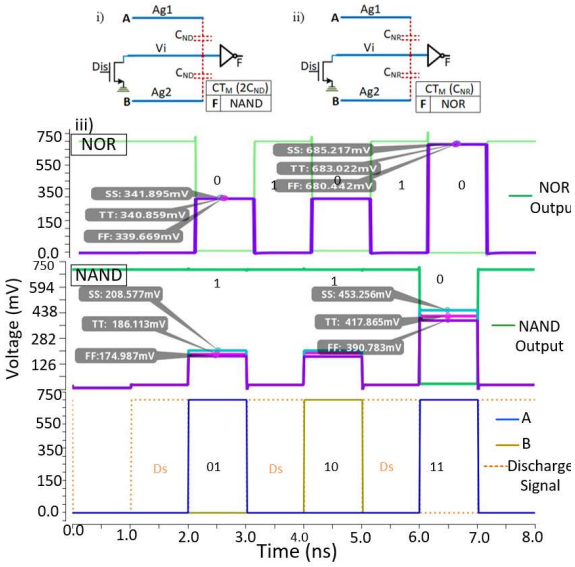


Fig.1 Fundamental Logic Gates: i) NAND ii) NOR iii) Simulation results of NAND and NOR gate with *Vi* node voltage under different process variation

or both the inputs transition (A or B), the *Vi* node gets the summation voltage 1 (the inverter output receives 0, hence NOR gate). For NAND, both the inputs need to transition to 1 to receive the summation voltage 1 on *Vi* node (0 at F). HSPICE simulation results validate the Crosstalk principles (Fig.1 (iii)). The inverter connected to *Vi* makes the logic inverted and ensures full swings for fan-outs. The inverter is one of the key components of the Crosstalk fabric that acts as a threshold function which regenerates the signals and restores them to full swing. Fig.1iii also shows that even though process variations has impact on the victim node voltage (Vi) but CT logic circuits are still able to maintain the functionality by achieving proper output. As discussed in the previous section, if the *Vi* node voltage goes below the switching threshold voltage of 0.3V, the inverter will output logic level '0' and vice versa. Such low switching threshold voltage of the inverter is indeed a key parameter since for any instance if the output voltage does not achieve full swing, the inverter will still be able to respond to incoming voltage.

As mentioned, Crosstalk computing fundamentally relies on interference between nodes for computation. However, transistors play an important role in controlling the interference pattern and ensuring full swing output. Therefore, transistor related variability challenges persist in Crosstalk, but the overall effect is lesser due to the less number of transistors being used.

III. Designing Under Variation at 7nm

Process variation may arise due to various issues and ultimately impact transistor performance, hence the standard in the industry is to name different process corners as FF (Fast-Fast), TT (Typical-Typical) and SS (Slow-Slow) with the first letter refers to NMOS and later one refers to PMOS. By combining FS, FT, etc., other process corners can be obtained; however, FF, TT, and SS are representative of best, nominal and worst-case scenarios. Fig. 2 shows the effect of process variation on the transfer characteristics curve of an unskewed inverter at 7nm. As can be seen from Fig. 2, the inverter has a weak PMOS transistor causing the switching threshold (*Vm*)

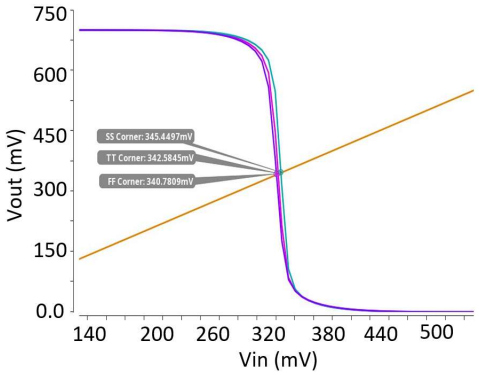


Fig. 2 Voltage characteristics curve of an Inverter for different process variation

to shift more towards zero. In general, the switching threshold voltage is desirable to be equal to exactly half of VDD (0.35V) since this would provide a higher noise margin. For worst-case process variation (SS), *Vm* moves more towards left (Fig. 2) and thereby, increases the undefined region. Under such condition, an incoming signal with noisy zero value would lead to erroneous values at the output.

For Crosstalk computing, with the symmetrical inverter connected to its *Vi* node, the incoming input value, as can be seen from the Fig. 2, should lie within 0V to 0.33V to have perfect logic '1' at the inverter output. However, to have better noise immunity and balanced drive strength, the width of the PMOS needs to be increased since this would shift the switching voltage towards half of the VDD. The properties obtained from Fig. 2 from different process corner is very useful for designing Crosstalk circuits with different fan-ins and indicative of power and performance profile.

We used the ASAP7nm PDK [7] to evaluate Crosstalk circuits at 7nm. ASAP7 PDK is compatible with industry CAD tools for full physical verification (Layout design, DRC, PEX, and LVS). The PDK is a 7nm FinFET technology that comes with transistor models having four different threshold voltage levels. The four devices reported in ASAP7 PDK are SLVT, LVT, RVT, and SRAM in decreasing order of drive strength, for both NMOS and PMOS transistors. The RVT type NMOS transistor has Ion of 37.85uA and Ioff of

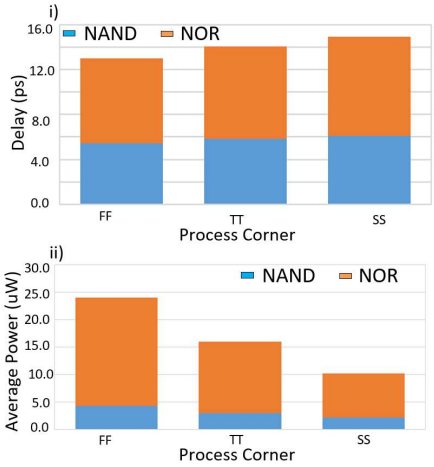


Fig. 3 Impact of process variation on i) Performance and ii) Average power for CT NAND and NOR Gate

0.019uA, providing excellent subthreshold swing of 63.03 mV/decade. Similarly, PMOS transistor also has a similar subthreshold swing of 64.48 mV/decade having Ion of 32.88uA and Ioff 0.023uA. Using these NMOS and PMOS transistor, for an unskewed inverter, high noise margin is found to be 0.33V and low noise margin to be 0.3V with switching threshold voltage at 0.34V.

Fig. 3(i&ii) shows the power and performance results of Crosstalk NAND and NOR gate at three process corners for PMOS and NMOS devices: SS, TT, and FF. As shown in Fig. 3i&ii, the slow transistors result in a slower transition of 5.53ps and 8.77ps for both NAND and NOR gate, respectively, but the functionality remains intact. The delay is minimum for the FF corner, but the power is also highest. The bar graphs show the impact of process variation on performance and power for both NAND and NOR gate with TT being the nominal case.

IV. Designing Large-Scale Circuits In Crosstalk Technology at 7nm

In the previous sections, we have discussed how different Crosstalk primitive logic cells can be achieved and also shown that these cells can be designed at smaller technology nodes with no functionality issue. In this section, we show that Crosstalk cells can be connected in a cascaded manner for large-scale circuit design at smaller technology node.

Fig. 4 shows an example of a large circuit implementation of Crosstalk computing using 7nm ASAP PDK and Fig. 5 shows the simulation results of the circuit. The circuit is 'cm85a' circuit, one of the benchmarking circuits from MCNC suits [8]. The circuit has eleven primary inputs (a-k) and three primary outputs (l-n) as shown in Fig. 4. The netlist

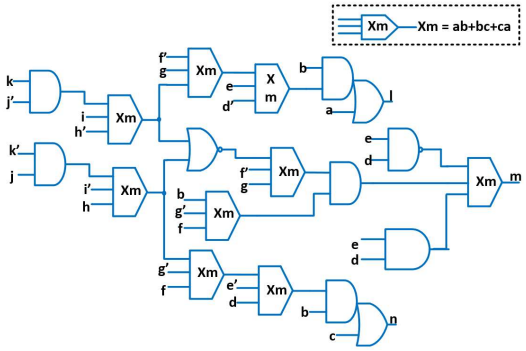


Fig. 4. Schematic of cm85a circuit

of the circuit is obtained from the MCNC and then further simplified to get Crosstalk friendly netlist according to the process explained in [11]. More benefits can be achieved when the Crosstalk circuits is implemented in homogeneous (f= ab+bc+ca) or heterogeneous manner (f=a+bc) [11].

Implementing large circuits using Crosstalk gates, requires attention on maintaining signal integrity and drive strength for the next stage gates. These issues can be addressed by adding inverter at the end of the victim node. The inverter acts as a thresholding function, that is, if the victim node voltage is below a certain voltage limit, the inverter restores the logic level '0' and vice versa. The strength of the inverter depends on the number of fan-out load or type of Crosstalk gates, it is driving. As can be seen in Fig. 4, at the second level, Crosstalk gates are driving two fan-out loads of higher coupling capacitances. To avoid the signal drop at the fan-in of the next stage gates, the inverters are designed as a hi-skewed inverter. Another key issue, specific to Crosstalk circuit, is during each evaluation state, the CT gates need a transition of the input

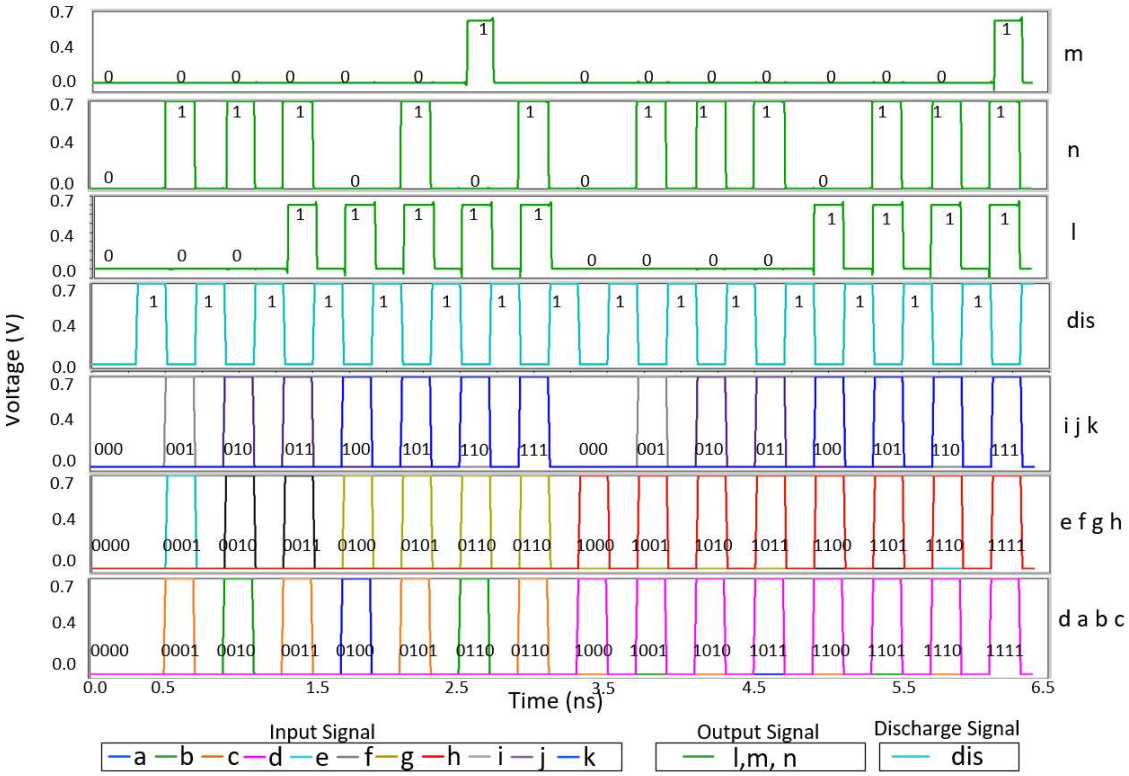


Fig. 5. Simulation results of cm85a circuit design at 7nm

TABLE I. COMPARISON OF DENSITY, POWER AND PERFROMANCE FOR DIFFERENT LARGE-SCALE CIRCUIT

MCNC Benchmark Circuits	I/O		Transistor Count			Average Power (uW)			Performance (ps)		
	CMOS	*Crosstalk*	*CMOS*	*Crosstalk*	*%Reduction*	*CMOS*	*Crosstalk*	*%Reduction*	*CMOS*	*Crosstalk*	*%Reduction*
NAND2	2/1	2/1	4	3	25	0.119	0.075	37	7.78	5.79	25.57
NOR2	2/1	2/1	4	3	25	0.376	0.387	-3	6.56	8.22	-25.3
Cm85a	11/3	11/3	168	69	59.00	60.63	28.95	52.25	21.10	18.93	10.28
Mux	21/1	21/1	506	190	62.45	73.70	29.90	59.43	9.43	12.75	-35.2
Pcle	19/9	19/9	328	252	23.17	288	110	61.80	15.04	7.07	53.00

signal from 0 to 1 for correct logic operation. So, if a logic high is retained on the victim node from the previous operation, it leads to logic failure. For example, in Fig. 4, at third level, a CT-NOR gate, which is an inverting CT-logic gate, is driving the next stage CT-homogenous gate. During discharge (Dis) state, it receives a logic high which is carried to the next evaluation state and thereby prevents the transition of signal from 0 to 1 leading to logic failure. This issue can be resolved by using a pass-gate type circuit-style [2]. In this type of circuit style, a transmission gate is placed between an inverting and non-inverting gate interfaces. During discharge state, input signal coming from the aggressor that is connected to the transmission-gate is discharged to ground and in evaluation state, the input signal is passed through the transmission-gates thus, creating a signal transition from 0 to 1. For CT-circuits with three inputs, especially for CT-OR3 gates, high coupling strength is required for proper logic function. This is due to the case that the victim node requires more charge accumulation to turn on the inverter connected to it, the inverter has a higher threshold voltage. However, since ASAP7 PDK, comes with transistor models with four different threshold voltages, the transistor that requires lower threshold voltage can be used in CT-OR3 gates to avoid higher coupling. Additionally, buffers can be used to maintain the signal strength to drive the gates that are far placed. Using the above-mentioned steps, the cm85a circuit is implemented where only three additional buffers were needed and simulation results in Fig. 5 shows the correct functionality is maintained. As such, any large-scale circuits can be implemented even at smaller technologies using CT-gates in this manner while maintaining correct circuit functionality and achieving improved density, power and performance benefits.

V. COMPARISON AND BENCHMARKING

Table I shows the detailed comparison of density, power, and performance for different circuits between Crosstalk and CMOS technology. For comparison, both Crosstalk and CMOS circuits are simulated using ASAP7 PDK and keeping nominal VDD at 0.7V. The benefits are significant in all aspects of Crosstalk logic-based implementations. In terms of transistor count, the highest reduction was for the mux circuit, it was 62%. For cm85a and pcle circuits the reduction in transistor count is 59% and 23% respectively. Crosstalk circuits show on average 58% power benefits over CMOS counterparts. The benefits are primarily due to the reduction in transistor count. For primitive cells, transistor count reduction is 25% for both NAND and NOR gates. However, the reduction in average power for the mux circuit is not much even though transistor count reduction is maximum compared to other circuits. This is because mux circuit implementation requires many pass-gate type circuit styles circuit which results in more switching activities, hence, less power reduction. On the contrary, for pcle circuit, power reduction is more because it requires less number of buffer and pass-gate type circuit styles that means less switching activity. The effect can also be seen for performance (Table I) where CMOS technology shows better performance than Crosstalk. However, for cm85a and pcle circuits, Crosstalk circuits have 10% and 53% improvement in performance respectively.

VI. CONCLUSION

In this paper, we presented Crosstalk computing's scalability aspects. Using ASAP7nm PDK, we have shown that for both best case and worst case process variations, Crosstalk circuits can be designed to function properly, and benefits over CMOS can be achieved. We have also shown the implementation of three MCNC benchmark circuits and compared density, power and performance results with respect to CMOS at 7nm. Our results show significant benefits over CMOS; for the best case, there is 62%, 62% and 53% reduction in density, power, and performance, respectively.

REFERENCES

[1] N. K. Macha, et al., "A New Concept for Computing Using Interconnect Crosstalks,"IEEE ICRC 2017 Washington, DC, USA.

[2] N. K. Macha, et al., "Crosstalk based fine-grained Reconfiguration Techniques for Polymorphic Circuits," IEEE/ACM NANOARCH 2018.

[3] N. K. Macha, et al., "A New Paradigm for Fault-Tolerant Computing with Interconnect Crosstalks," 2018 IEEE ICRC, Washington, DC, USA.

[4] N. K. Macha, et al., "A New Paradigm for Computing for Digital Electronics under Extreme Environments," IEEE Aerospace Conference 2019

[5] R. Desh, et al.,"A Novel Analog to Digital Conversion Concept with Crosstalk Computing," IEEE/ACM NANOARCH 2018.

[6] Clark LT., et al., "Design flows and collateral for the ASAP7 7nm FinFET predictive process design kit." 2017 IEEE MSE.

[7] Clark LT, et. al., G. "ASAP7: A 7-nm finFET predictive process design kit," Microelectronics Journal. 2016 Jul 1;53:105-15.

[8] Yang, Saeyang. Logic synthesis and optimization benchmarks user guide: version 3.0. Microelectronics Center of North Carolina (MCNC), 1991.

[9] K. J. Kuhn, "Considerations for Ultimate CMOS Scaling," in IEEE Transactions on Electron Devices, vol. 59, no. 7, pp. 1813-1828, 2012.

[10] K. J. Kuhn, "CMOS transistor scaling past 32nm and implications on variation," 2010 IEEE/SEMI ASMC, , pp. 241-246.

[11] Iqbal, A., et. al., "A Logic Simplification Approach for Very Large Scale Crosstalk Circuit Design," IEEE/ACM NANOARCH 2019.

Integrated Photonics Architectures for Residue Number System Computations

Jiaxin Peng, Yousra Alkabani, Shuai Sun, Volker J. Sorger, and Tarek El-Ghazawi
Department of Electrical and Computer Engineering
The George Washington University
Washington D.C., USA
Email: {vitapp, yousra, sunshuai, sorger, tarek}@gwu.edu

***Abstract*—Residue number system (RNS) can represent large numbers as sets of relatively smaller prime numbers. Architectures for such systems can be inherently parallel, as arithmetic operations on large numbers can then be performed on elements of those sets individually. As RNS arithmetic is based on modulo operations, an RNS computational unit is usually constructed as a network of switches that are controlled to perform a specific computation, giving rise to the processing in network (PIN) paradigm. In this work, we explore using integrated photonics switches to build different high-speed architectures of RNS computational units based on multistage interconnection networks. The inherent parallelism of RNS, as well as very low energy of integrated phontonics are two primary reasons for the promise of this direction. We study the trade-offs between the area and the control complexity of five different architectures. We show that our newly proposed architecture, which is based on arbitrary size Benes (AS-Benes) networks, saves up to 90% of the area and is up to 16 times faster than the other architectures.**

***Index Terms*—Residue Number System, Optical Computing, Processing-in-Network, Post-Moore's Law Processors**

I. Introduction

Residue number system (RNS) decomposes a binary weighted number system (WNS) into a set of smaller moduli. Arithmetic operations can be performed on each modulus independently and thus enable parallel computations, providing high-performance [1], [2]. RNS can be used in multiple compute intensive applications including cryptography [3], [4], image processing [5], and neural networks [6]–[8]. This makes an efficient implementation of an RNS system essential for use in such critical applications. As the main building block of an RNS system, the WNS to RNS block converts large binary numbers into a set of smaller numbers, whereas the RNS to WNS block reconstructs the large number back (Fig. 1). The RNS compute unit is composed of parallel independent units that perform independent operations in modular arithmetic. The efficiency of RNS depends on the efficiency of each of those blocks; however, in this work we focus on the RNS compute unit.

A traditional RNS computational unit in digital computing systems is implemented using combinational logic, or lookup tables (LUT), or integrating both technologies [9]–[11]. For a large computing system, the LUT should be large enough to store all the information that is needed for conversion and calculation. The execution time for a load operation from the LUT to the residue arithmetic unit limits the computation speed. Alternatively, a set of carry-save adders (CSA), partial sum adders (PSA) and full adders (FA) are utilized to build the RNS adders and multipliers.

This project is supported by the Air Force Office of Scientific Research (AFOSR) Award number FA9550-19-1-0277.

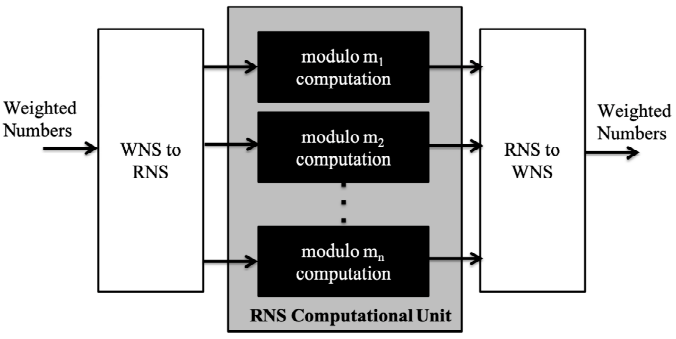


Fig. 1. Building blocks of an RNS system.

Adopting photonics into RNS arithmetic is benefitial as light propagates at high speed, which means the execution time for an operation is fast. Huang et al. proposed several optical methods to implement residue arithmetic unit as spatial maps, where results were encoded as the spatial position of a light beam. [12]. Utilizing electro-optical switches is one of the methods. The light beam is applied at a specific input port of one of the electro-optical switches based on one of the operands. The electro-optical switches are arranged so as to *permute* the position of the input light beam based on the required operation and one of the input operands.

The RNS computational units proposed in [13] utilized a *shifting* operation for RNS addition based on 2×2 switches, and considered such RNS adder as the basic sub-unit to construct a programmable RNS computational unit. Addition, subtraction, multiplication, division, and polynomial computations could be further implemented in such a system. Based on this idea, micro-ring resonators (MRR) [14] and all optical switches (AOS) [15] are proposed to implement the 2×2 switches used to build the RNS adder.

Alternatively, Peng et al. applied the processing-in-network (PIN) concept on RNS arithmetic and implemented the so called all-to-all sparse directional (ASD) architecture for RNS

978-1-7281-5221-9/19/$31.00 ©2019 IEEE

adders and multipliers [16]. In this work, RNS computations are implemented utilizing an all-to-all non-blocking *routing* network. And a hybrid plasmonic-photonic (HPP) 2×2 switch has been proposed as the building block to construct the ASD RNS architecture [17], [18]. This HPP switch has low latency and femto-Joule energy consumption. In addition, it has a compact area (around 20 μm^2), hence it can be integrated in on-chip designs. The residue computational units can thus operate at high-speed with low-power and low-area overheads.

While the optical architectures provide a high speed implementation for both adders and multipliers, they suffer from relatively high demand of optical components or complex logic design. In this work, we explore more architectures based on integrated photonics technology for RNS computational units. These architectures are built using the HPP 2×2 switch. We study architectures that are based on both *shifting* and arbitrary *routing* design concepts.

We introduce one *shifting* architecture that improves upon prior work and reduces the number of required optical components to build it. Moreover, we propose two novel *routing* architectures that are based on Benes [19] and arbitrary size Benes (AS-Benes) [20] networks that utilize even less optical components than the state-of-the-art ASD design. We study the trade-offs between the area and control complexity of the mentioned architectures.

II. Background

A. Residue Number System

RNS represents a number with the residues from a set of relatively prime moduli. Considering a number *X* with a modulo *m*, then *X* could be expressed by its residue *r* as (1).

$$r = |X|_m \tag{1}$$

or $X \equiv r \pmod{m}$. For example, the number *X=64* with modulo $m = 13$ could be represented as $r_X = |64|_{13} = 12$. Similarly, the number *Y=51* with the same modulo could be represented as $r_Y = |51|_{13} = 12$. Both X and Y are represented as 12 using modulus 13. X and Y are congruent modulo-m in this case. The same representation limits the identification of the numbers due to the representation range for a single modulus *m*, which is $0 \sim (m-1)$ for unsigned numbers. Instead of utilizing a single modulus, a set of multiple moduli, $\{m_1, m_2, ..., m_n\}$, is proposed to represent a number. Then a number could be represented as one set of residues, $\{r_1, r_2, ..., r_n\}_{[m_1,m_2,...,m_n]}$ accordingly. For example, assuming there is one set of moduli {13, 16, 19}, number *X* and *Y* would be represented in RNS as $\{12, 0, 7\}_{[13,16,19]}$ and $\{12, 3, 13\}_{[13,16,19]}$ respectively. Assuming the product of the moduli set is *M+1*, i.e. $M = (\prod_{i=1}^{n} m_i - 1)$, where m_i represents *ith* modulus in the moduli set. *M* is named as the dynamic range. The dynamic range for an unsigned number is $[0, M-1]$.

Negative number representation is available in RNS as well with complement notation. A negative number, -*X*, is represented as (2) with respect to modulus *m* in residue domain,

$$r = |m - |X|_m|_m \tag{2}$$

where r is the residue notation for a negative number -*X* in a modulo-m system. From the previous example, $64 = \{12, 0, 7\}_{[13,16,19]}$, then for its negation, $-64 = \{13-12, 16-0, 19-7\}_{[13,16,19]} = \{1, 0, 12\}_{[13,16,19]}$. The dynamic range of a signed number could be either $[-(M-1)/2, (M-1)/2]$ if M is odd, or $[-M/2, M/2-1]$ if M is even for an RNS system.

The selected moduli set should be pairwise relatively prime. Otherwise, the dynamic range will be smaller than the one specified earlier. Within the same dynamic range, two or more numbers might be represented as one expression.

B. Arithmetic of RNS

The arithmetic of RNS is digit-irrelevant, giving it the potential to be implemented and calculated in parallel. For a single modulus in the moduli set, the results of the arithmetic operation are independent on each other. X+Y, for example, could be calculated with residue notation as follows.

$$\begin{aligned} X + Y &= \{12, 0, 7\}_{[13,16,19]} + \{12, 3, 13\}_{[13,16,19]} \\ &= \{12+12, 0+3, 7+13\}_{[13,16,19]} \\ &= \{11, 3, 1\}_{[13,16,19]} \end{aligned} \tag{3}$$

In the decimal number system, X + Y = 64 + 51 = 115, which could be represented as $\{11, 3, 1\}_{[13,16,19]}$ in RNS. It is exactly the same as the result derived from (3). Subtraction and multiplication follow the same rule. The results of $X - Y$ and $X \times Y$ would be identical when derived from these two methods within the chosen range. The subtraction could transform the operation to addition as $X + (-Y)$. -Y could use the negative number representation in the complement notation. Multiplication is similar to addition, as shown in the example below.

$$\begin{aligned} X \times Y &= \{12, 0, 7\}_{[13,16,19]} + \{12, 3, 13\}_{[13,16,19]} \\ &= \{12 \times 12, 0 \times 3, 7 \times 13\}_{[13,16,19]} \\ &= \{1, 0, 15\}_{[13,16,19]} \end{aligned} \tag{4}$$

The property of no carry propagation in RNS arithmetic allows the parallel implementation. Therefore, operations on larger numbers could be decomposed into a set of smaller numbers and be performed in parallel.

C. Nanophotonics RNS Computational Unit

The fundamental component for the integrated photonics RNS arithmetic device is a 2×2 switch with two states that are controlled by an electrical signal, shown in Fig. 2. The two states, bar state and cross state, could be switched by the control signal. In the bar state, the input light source will propagate straight forward through the switch (Fig. 2(a)), whereas in the cross state, the light source will be routed to the opposite output (Fig. 2(b)). A voltage control signal will be applied to the switch in order to set its state.

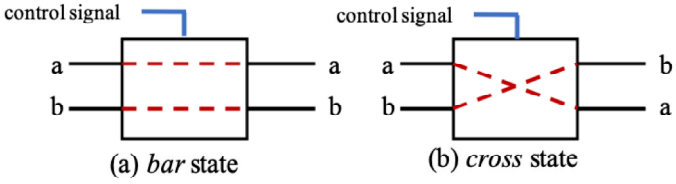


Fig. 2. Conceptual Schematic of Two States of a 2×2 Switch. (a): *bar* state. (b): *cross* state.

The HPP switch is ideal for the RNS computational unit. To be more specific, this hybrid photonic-plasmonic integrated structure is able to combine the advantages from both low-loss photonic waveguide and high light-matter interaction (LMI) energy and speed efficient plasmonics. Although extra loss will arise when integrating two technologies together due to the light mode mismatch, it has been reported that the coupling loss could be engineered to be 0.5 dB or even lower [21]. To explain this, the switching mechanism of the switch needs to be discussed. As a three-waveguide directional coupler, the fundamental modes (TM_1, TM_2 and TM_3 in this case) could be electrically tuned from the *cross* state, in which light couples from one bus waveguide, to the other side of the bus to the *bar* state, in which light stays in the same bus. Due to the coupling theory of the directional coupler, from cross to bar state, the actual coupling length increases dramatically due to the carrier concentration change of the material (i.e. Indium Tin Oxide). It is also worth mentioning that although the coupling distance of the bar state is much longer than the physical length of the switch and the incident light will be forced to stay within the same bus waveguide, a small portion of the light will still be coupled to the other side, thus causing an unbalanced switching (i.e. the extinction ratio (ER) at the bar state will be lower than the ER at cross state). Therefore, the two states performance can be regarded as a trade-off among loss, extinction ratio and footprint that depends on different applications. The HPP switch theoretically could be operated at 400GHz speed. However, the whole system will be limited by the speed of the modulators and photodetectors. In addition, the complex electronic control circuit of the routing based residue adders that can include look-up tables is another bottleneck.

III. Integrated Photonics Architectures of RNS Computational Units

The adder is one of the most critical computational units for an RNS arithmetic system. The RNS adder based on the photonic 2×2 switch is considered to be the basic computation block since other computational units could be implemented using the adders, including subtraction, multiplication and evaluation of polynomials [13]. A residue adder can be implemented based on the shifting property of a residue additive operation, or the routing capability of the adder when considering it as a network. In this work, five residue architectures will be discussed (Table I). Two of those architectures have been previously proposed to be used as residue adders: Tai Adder [13] and the All-to-all Sparse Directional (ASD) adder [16].

TABLE I
Summary of the Architectures of the Residue Adders

Name	Use as RNS Adder
Tai Adder [13]	Yes
Shifting	No
All-to-all Sparse Directional (ASD) Adder [16]	Yes
Benes [19]	No
Arbitrary Size Benes (AS-Benes) [20]	No

The residue additive operation acts like a simple shift right operation when one-hot-encoding is deployed. Tai et.al, proposed an RNS adder with electro-optical component in work [13]. We propose an improved architecture based on the Tai RNS adder, the shifting RNS Adder. Both of these two architectures are implemented using the shifting properties of RNS addition.

While the control logic of these architectures is simple, they require a large number of optical switches, scaling to $O(N^2)$ for a modulo-N system. Despite the shifting-like operation for RNS addition, the RNS adder could be considered as a routing network as well. The all-to-all sparse directional (ASD) RNS is proposed as a non-blocking all-to-all communication network [16]. To reduce the overhead of optical components, we propose a routing network based on Clos and Benes networks [19] or Arbitrary Size Benes (AS-Benes) networks [20]. In the following subsections, more details of the proposed architectures will be explored. Since RNS arithmetic is digit-independent, single modulo-N systems could be designed separately. Furthermore, the multipliers implementation will be explained as well.

A. Shift Operation Based RNS Adders

A modulo-5 RNS addition could be mapped as shown in Fig. 3, which is a shift right operation [12], [13]. This shift operation could be acheived by the optical 2×2 switches. Both the Tai and shifting RNS adders are designed based on this property. The input is shifted several times according to the summand, and the results of the addition operation would be derived at the output ports. These methods require some simple control circuits (e.g. SR-flip flops) to control the electro-photonic switches. The implementation is easy and feasible. Also, it is scalable as the modulo number increases.

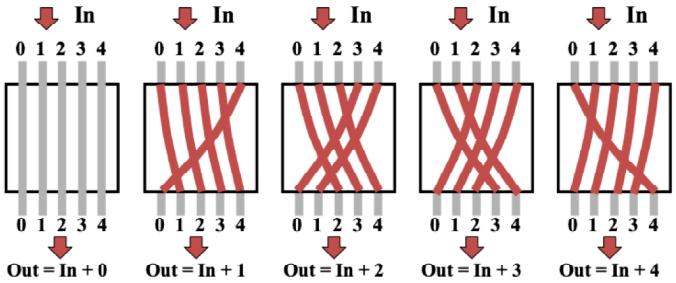


Fig. 3. Modulo-5 Addition Example.

1) Tai RNS Adder: The Tai RNS Adder (shown in Fig. 4) is first proposed in [13]. For an operation $X + Y$, X is represented as the light source, while +Y acts as a control signal for the 2×2 switch. Assuming the default state of these switches is the bar state, the switch states will be changed to cross state if the voltage is applied on them. According to the value of Y, control signals will be applied to different rows in order to change the state to the cross state. If X equals to 2, then X will be translated as light signal at position 2. Lights will propagate straightforward to the output which is the same as the input without any control selection. It acts like "+0" in this case. Fig. 4 shows an example with X=2 and Y=4. The +4 control activates the whole row for +4 as cross state. The input light at position 2 will then be routed to the position 1 at the output, which reflects the result of the operation $|2 + 4|_5$. Furthermore, the result will be routed as desired regardless of the value of X as long as the +4 control signal is set. For a modulo-N system, $N(N + 1)$ 2×2 optical switches are required in total.

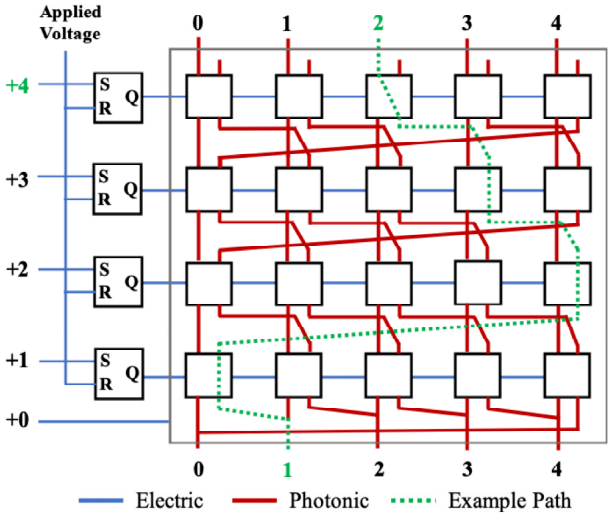


Fig. 4. A Modulo-5 Tai RNS Adder with Bar State as Default [13]. Additive operation $|2 + 4|_5 = |1|_5$ is shown as an example.

Alternatively, if the default state of the switch is cross state, the architecture would be represented as shown in Fig. 5. The design is very similar to the previous one, and requires the same number of 2×2 switches. Based on the switch design, either architecture could be selected. The choice of default state depends on the fundamental design of the switch. In order to save more power, the low loss state should be considered as the default one since most of the time the switch will be kept at the default state. In this case, the selected HPP switch has lower loss in the cross state, hence this latter Tai residue adder performs better in terms of power consumption, resulting from both the light loss and the charging power for changing the state of the switch.

2) Shifting RNS Adder: We propose a different architecture for residue addition based on the shifting property as well (Fig. 3). For a modulo-N system, each set of the diagonally aligned N-1 switches is considered as one level. There are

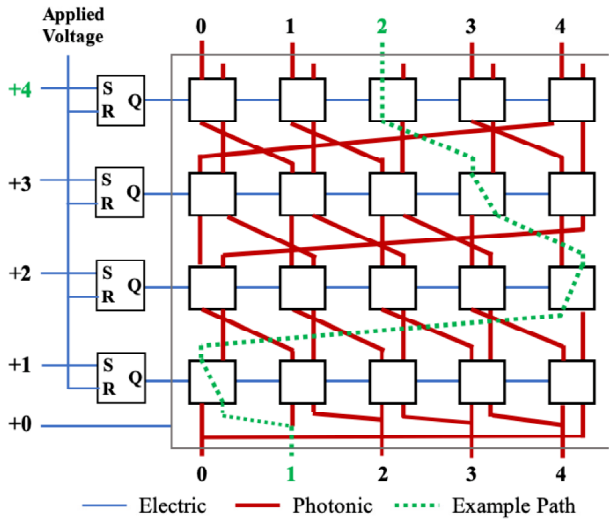


Fig. 5. A Modulo-5 Tai RNS Adder with Cross State as Default [13]. The example shows the operation $|2 + 4|_5 = |1|_5$.

N-1 levels for an RNS adder, and thus there are $(N - 1)^2$ switches in total. The total number of switches required for such architecture is less than the one required by Tai RNS adder. In this design, the default state of the 2×2 switch is assumed to be the bar state. By utilizing the shift operation for RNS addition, switches of each level will be changed to the cross state accordingly. By enabling each level, the corresponding input will be shifted once. Operation X+Y, for example, X is represented as the input light source, while Y specifies how many levels of switches should be under the cross state. To enable the level switching, a set of logic circuits is necessary. Similar to the Tai RNS adder, SR-flip flops act as the controller for the shift level selection.

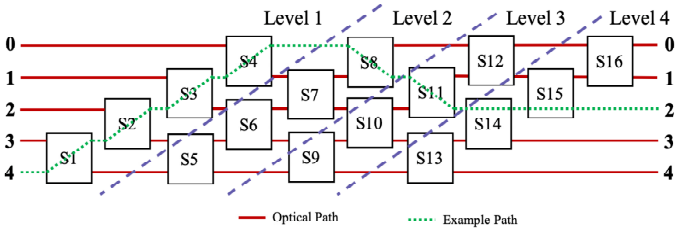


Fig. 6. A Modulo-5 Shifting RNS Adder with Example $|4 + 3|_5 = |2|_5$.

Fig. 6 shows an example of operation 4+3 in a modulo-5 system. 4 is represented as the input light source. For the +3, all the switches of the first three levels will be set as cross state. The input light will be shifted three times. Finally, the input light source will reach the destination at position 2, which is the result of the mentioned operation ($|4 + 3|_5 = |7|_5 = 2$). By setting up the shift level accordingly, the input light will be routed to the desired output destination.

B. Routing-based RNS Adders

An RNS adder could be also considered as a non-blocking routing network. A look-up table (LUT) is needed for storing

the states of switches for different operations. In general, the number of switches in this type of RNS adders is less than the shifting-operation based ones. However, the control unit requires more electrical components with complex design. In addition, loading the states from the LUT is necessary, which adds extra delay.

1) All-to-all Sparse Directional (ASD) RNS Adder: The ASD RNS adder, proposed in [16], considers the system as a routing network instead of computing device. For an operation X+Y, X is represented as the input light source, while the +Y is represented by one set of switch states that are stored in one entry of a LUT. By deriving the pre-calculated states that satisfy the all-to-all non-blocking routing network from the LUT and setting them up accordingly, the input light could be routed to the expected output port.

Fig. 7 depicts the design concept for the ASD RNS adder. Fig. 7 (a) and (b) show a 3-port and 4-port non-blocking routing network without self-communication, i.e. the source 0 could not be transmitted to destination 0. With these two basic units, an N-to-N non-blocking network without self-communication can be derived as shown in Fig. 7 (c). Assuming the (N-2)-port non-blocking network can be routed accordingly, an N-port network can be generated by adding two extra input ports. Two sets of diagonal N-1 switches are added to enable the routing of the two new input ports to any desired output port. To enable self-communication, two more switches are added on the two top and the two bottom waveguides [16].

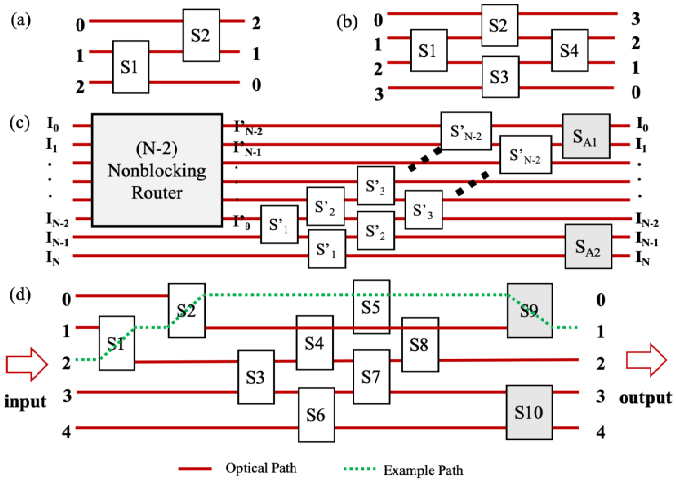


Fig. 7. All-to-all Sparse Directional (ASD) RNS Adder [16]. (a): A 3-port router without self-communication. (b): A 4-port router without self-communication. (c): Conceptual of a modulo-N ASD Adder. (d): A modulo-5 ASD residue adder with example $|2+4|_5 = |1|_5$.

To be more specific, Fig. 7 (d) shows an example of operation 2+4 for a modulo-5 system. Input light is at port 2; states of S1 to S10 are "CCCCBBCCCC" (B/C represents bar/cross state) when 4 is the other summand. The RNS results is 1 for $|2+4|_5$. Every time the summand changes, the system will need to reload all the states in the LUT to route the network successfully. In total, a modulo-N ASD RNS adder requires $(N-2)^2/2+2$ switches if N is odd, or $N(N-2)/2+2$ switches if N is even (N>2).

To derive the states for each switch and store them in the LUT, all possible permutation for the ASD adder should be listed. Based on the selected HPP switch, the states with less bar states requirement for each operation to reduce the power consumption are chosen.

2) Clos and Benes Network-based RNS Adder: Clos network is proposed decades ago for non-blocking communication with any permutations by setting up a series n×k switches [22]. Benes network [19] is one of the special cases of Clos network, utilizing 2×2 switches. Benes network is rearrangeable for any permutation for the all-to-all non-blocking communication. For any source in the network, there is always a path that could be setup to any destination which is not occupied by other sources. Hence, it is very suitable to be considered as an RNS adder architecture.

A regular Benes network is shown in Fig. 8. There are $2log_2N-1$ stages in an N-to-N network, and each stage has N/2 2×2 switches. In the first stage, two inputs are connected to a switch, which separates these two inputs to two sub-networks. Similarly, in the last stage, both sub-networks would be connected to a switch, and thus the output could be derived from either sub-network. In the middle stages, two (N/2)-to-(N/2) subnetworks will be built recursively, until it reaches the smallest component, a 2-to-2 network, which could be easily implemented by a 2×2 switch. The subnetworks are then combined to form a larger network. $Nlog_2N-N/2$ 2×2 switches are required for the implementation.

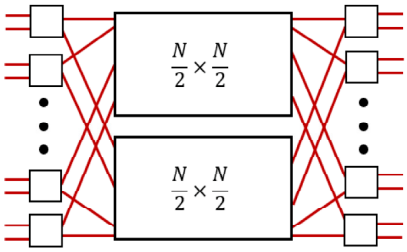


Fig. 8. Conceptual Structure of an N-to-N Benes Network [19], where $N = 2^k$ (k is an positive integer).

3) Arbitrary Size Benes (AS-Benes) Network-based RNS Adder: A regular Benes network is designed for an N-to-N nonblocking network, where $N = 2^k$, k is an integer. To deploy a modulo-N RNS adder, N could be any integer, i.e. N may not be equal to 2^k. A modulo-5 adder, for example, needs at least an N=8 Benes network for implementation. However, only 5 inputs and 5 outputs will be utilized, resulting in wasting resources. Hence, the arbitrary size Benes network [20] is more suitable in this case. An arbitrary size Benes (AS-Benes) network utilizes the same idea of the Benes network, which separates the network into two sub-networks; however, N could be arbitrary size.

For an arbitrary integer N, AS-Benes network is split into two subnetworks, an $\lfloor N/2 \rfloor$-to-$\lfloor N/2 \rfloor$ upper network and an $\lceil N/2 \rceil$-to-$\lceil N/2 \rceil$ lower network. Similar to the Benes network,

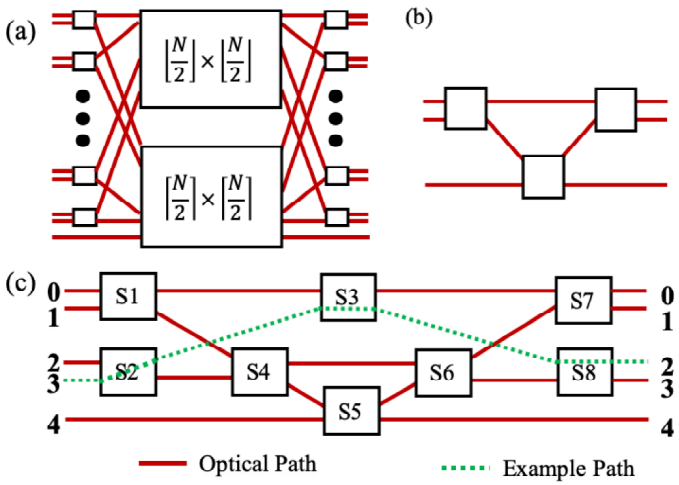


Fig. 9. An arbitrary size Benes Network Schematic [20]. (a):Conceptual design for an AS-Benes network when N is odd. (b):A 3×3 AS-Benes network. (c): A modulo-5 RNS adder built with AS-Benes network. The green lines show the path of operation $|3+4|_5 = |2|_5$.

every two inputs will be connected to a 2×2 switch in the first stage of AS-Benes network. If N is even, then it works exactly the same as Fig. 8; however, when N is odd, it does not follow this pattern. To save the number of switches, the last input will always connect to the sub lower network (Fig. 9 (a)). By splitting the network recursively, we reach either a 1×1 or a 2×2 network. The 1×1 network is considered as a wire while the 2×2 network is easily implemented as the basic switch. Fig. 9 (b) shows a 3×3 network, which combines the smallest networks mentioned above.

For a size N AS-Benes network, the total number of the 2×2 switches, *S(N)*, could be derived by (5), where S(1) = 0 and S(2) = 1.

$$S(N) = 2\lfloor\frac{N}{2}\rfloor + S(\lceil\frac{N}{2}\rceil) + S(\lfloor\frac{N}{2}\rfloor) \quad (5)$$

Fig. 9 (c) shows the example of a modulo-5 RNS network based on the AS-Benes design. Only eight 2×2 switches are required. Similar to the ASD design, for an operation 3+4, 3 is represented as the light source at position 3, while +4 will load the corresponding states of switches from the LUT, which is CCBCCBBB in this case.

C. Implementation of RNS Multipliers

Residue multipliers' design is different from the residue adder design, due to the zero-factor. If any number is multiplied by 0, then the product of such operation will be 0 as well. This special case needs to be designed carefully in the RNS multiplier. A modulo-N RNS multiplier could be decomposed into two parts. The first one contains the situation that there is at least one zero factor in the operation. All input light signals should be routed to the position 0. Hence, this path should be built from not only the input position 0 but also any other input positions. The 2×2 switches is helpful to make the selection. The other part is an (N-1)-to-(N-1) RNS with position range [1, N-1] (Fig. 10(a)). This part needs to be designed to route the input light according to the results of the multiplication operation. It is very easy to implement such schematic with a routing-based modulo-(N-1) adder. The fundamental idea here is different routing path requirement. All the considered routing-based RNS adders have been proven to work well as a rearrangeable non-blocking network. This means that a network can realize all possible permutations between inputs and outputs, which may require rearranging the existing connections. Hence, it is sufficient to build a routing-based RNS multiplier by generating another LUT storing the switch states required for the results of the multiplication.

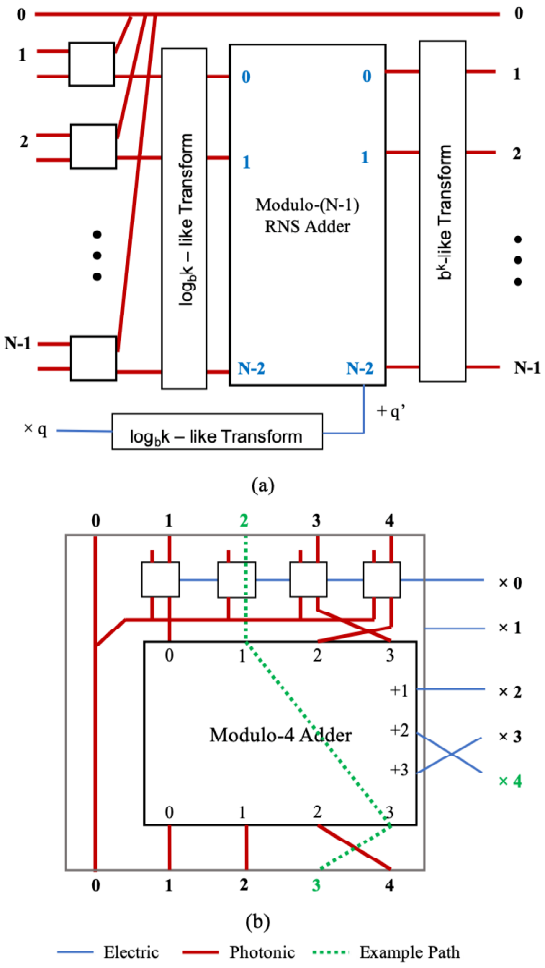


Fig. 10. RNS Multiplier Schematic. (a): A Modulo-(N-1) RNS multiplier implementation. (b): A modulo-5 RNS multiplier with log-transform implementation.

Alternatively, the homomorphic approach could be applied to the RNS multiplier. A modulo-N multiplier could be converted to a $log_b k$-like transform, a modulo-(N-1) additive operation, and a b^k-like transform [10], [12], [13]. If either multiplicand for the multiplication operation is 0, the product will be 0. Fig. 10 (b) shows the transform for a modulo-5 multiplier. By transforming both X and Y in operation $X \times Y$, the modulo-N multiplication could be converted to the modulo-(N-1) addition. This homomorphic approach could be

applied to any RNS adder architecture.

Comparing to the conventional multiplication in binary systems, the other advantage for RNS multiplication is that it could reach the same latency as RNS addition. To combine all these basic subunits, the whole system provides the possibility of single hardware implementation for both addition, subtraction and multiplication. Although it requires a lot of pre-calculation to decide the transform table, it reuses the addition LUT for the routing-based RNS adder.

IV. Overheads for Different RNS Computational Units

There are both pros and cons for the proposed integrated photonics architectures for RNS computational units. Table II shows the tradeoff of a Modulo-N RNS adder with different architectures. Both the Tai and shifting RNS adders are designed by the shifting property for the residue additive operation, and are controlled by a logic circuit. The designs have high scalability; however, the requirements for the optical components are large with complexity $O(N^2)$. The Tai model utilizes the most number of switches; however, the control logic is easy and the light propagation delay is less than the shifting adder. Alternatively, the routing-based RNS adders, including ASD, Benes and AS-Benes, require less optical components. Nevertheless, the tradeoff is the complexity of the control unit implementation. Since the control bits are stored in a LUT, the area of the LUT and the time of reading the LUT are additional overheads in this case.

The moduli set selection is one of the critical parts to build a residue system. All moduli must be pairwise coprime. Otherwise, the representation will not be unique in the range [0, M-1]. Hence, one possibility is to choose the minimum set of all prime numbers to cover the dynamic range. Conventionally, there is another selection set in pure electrical RNS, which is $\{2^k-1, 2^k, 2^k+1\}$, named as three-moduli set. This set of moduli is prime relative to each other. However, the latter selection is not suitable in the proposed nanophotonic RNS architectures due to the large size of a single residue computational unit. Table III shows the RNS units required for different modern binary system with different sizes, including 32-bit, 64-bit, and 128-bit binary numbers. A 32-bit binary system requires a $\{2047, 2048, 2049\}$ moduli set, which requires thousands of optical 2×2 switches. A single optical component has a large area and is not suitable for such a computational system. In addition, it limits the parallelism for the RNS system. Hence, it is not an ideal choice for the proposed nanophotonic RNS computation units.

Instead, the all-prime method performs better. Fig. 11 depicts the number of switches required for a single modulo-N system for different architectures. The highlighted three vertical lines indicate the maximum primes required for 32-, 64- and 128-bit computer systems respectively, which is the largest computation unit for the corresponding residue number system. The routing-based RNS computation unit has larger size than the shifting-based one does, due to the large requirement for optical components. Although the routing-based RNS computational unit schematic requires electrical control circuits with a LUT, the actual area needed is relatively small due to the compact transistor size ($0.01\mu m^2$) [23], compared to that of an HPP switch ($20\mu m^2$).

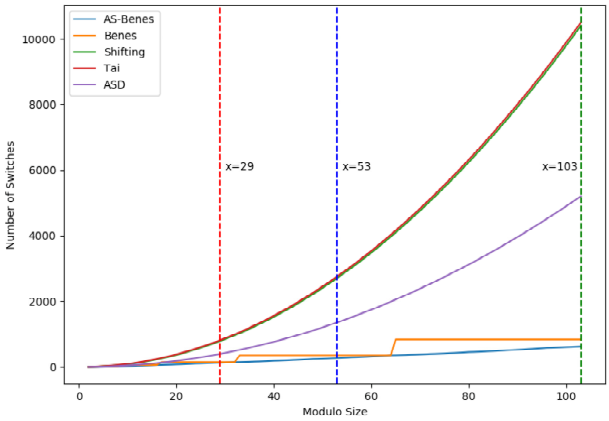


Fig. 11. Number of Optical Switches Required for a Modulo-N System. Highlighted vertical lines specify the largest modulus for 32-bit, 64-bit and 128-bit computer systems, respectively.

The area of an RNS addition computational unit for different size computer systems is shown in Fig. 12, where solid blocks show the area of optical components and hatch blocks depict the ones of electrical components. The major portion of the overall area are from optical components. In general, the routing-based RNS units perform better than shift-based RNS units in terms of area due to the smaller number of optical switches requirement. The AS-Benes architecture requires the smallest area as expected, due to the smallest number of switches required among all the studied architectures. Although the shift operation based residue computational units have simple control units, the area is relatively large. For the 128-bit system, the AS-Benes model saves 90% of the overall area of the Tai model.

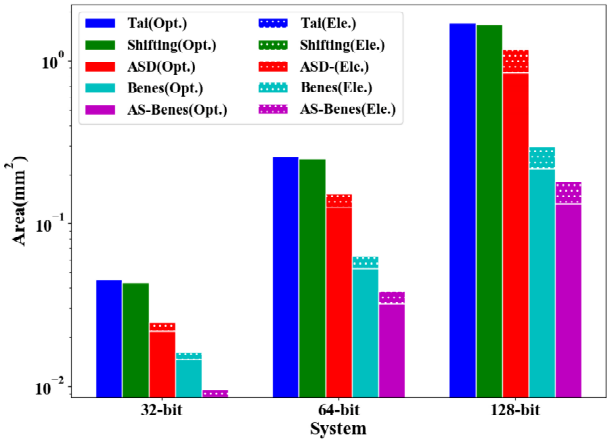


Fig. 12. Area of an RNS Adder on Different Binary Computer Systems with Nanophotonic Architectures. The solid blocks represent the area of optical components, while the hatch blocks represent the area of electrical components.

TABLE II
COMPARISON OF A MODULO-N RNS ADDER BUILT ON DIFFERENT ARCHITECTURES, INCLUDING TAI, SHIFTING, ALL-TO-ALL SPARSE DIRECTIONAL(ASD), BENES, AND ARBITRARY SIZE BENES (AS-BENES).

Parameters	Tai	Shifting	ASD	Benes	AS-Benes
# of switches S(N)	$N(N-1)$	$(N-1)^2$	Odd: $(N-1)^2/2+2$ Even: $N(N-2)/2+2(N>2)$	$Nlog_2N-N/2$	$S(N)=2\lfloor\frac{N}{2}\rfloor+S(\lceil\frac{N}{2}\rceil)+S(\lfloor\frac{N}{2}\rfloor)$ $S(1)=0; S(2)=1$
Control Unit Implementation	Logic Circuit		LUT		
# of Control Unit	N-1		S(N)		
# of LUT (bit)	-		N×S(N)		
max # of stages	$N-1$	$3N-6$ $(N\geqslant 4)$	$ST(N)=ST'(N-2)+4$ $ST'(3)=2, ST'(4)=3$ $ST(2)=1, ST(3)=3, ST(4)=4$	$2(\lceil log_2N\rceil)-1$	$2(\lceil log_2N\rceil)-1$

TABLE III
NUMBER OF SWITCHES REQUIRED FOR DIFFERENT SYSTEM

Binary System	All-Prime		Three-Prime
# of bits	# of primes	max prime	2^k
32	10	29	2048
64	16	53	4194304
128	27	103	8796093022208

The photon time-of-flight (TOF) for one HPP switch is 0.1ps. Fig. 13 shows the maximum photon TOF for each architecture with different modulo sizes. Note that the maximum photon TOF of Benes and AS-Benes architectures are overlapped since the maximum number of stages of these two architectures are the same. Assuming a modulo-103 residue adder, the largest propagation time is around 20ps. The selected HPP switch requires 5.1ps as the response time, which is the timing budget for changing the switch states from cross state to bar state. Modern ring resonators [24] and photodetectors [25] operate at 35GHz and 100GHz, respectively. Such resonators resonate light within 28ps and the photodectectors detect light in less than 10ps. Thus, the micro ring resonator speed would be limited by the modulator and thus it could reach at most 35GHz. By summing the photon TOF, switch response time, ring resonator time and photodetctor response time, the timing budget would less than 100ps. It is suitable for all modern computer systems that operate at $1\sim$ 2 GHz nowadays. Both addition and multiplication can be done in one cycle as long as the LUT selection is done. Benes and AS-Benes models have the lowest delay. The delay of a modulo-103 residue adder with AS-Benes model, for instance, is 13 switches only, which is exactly the calculation time for a residue additive operation. Tai model exhibit moderate delays while the other two models require relatively longer time for calculation.

In addition, the delay of multiplication requires the same time as addition. As the operand size increases, more time is saved by using the residue multiplication. Moreover, the HPP switch needs 5.1ps as the response time in addition to the propagation time, which is the overhead for the electrical control.

As a constraint of the input laser power, the coupled laser power should be less than 100mW to avoid entering

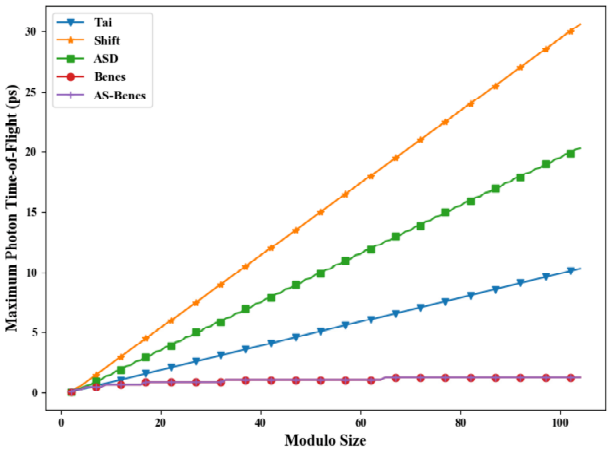


Fig. 13. Maximum Photon Time-of-Flight of Different Nanophotonic Residue Adders with Different Moduli.

the non-linearity region of the Silicon waveguide at 1550nm wavelength [26]. The insertion loss of the HPP switch is 2.1dB and 0.4dB at the bar and the cross states, respectively. Thus, the maximum number of switches would be a function of the number of switches at different states, which varies with the architecture. However, the Tai, shift and ASD models will not fit in this requirement for the nanophotonic switch when the modulo size scales up. Tai residue adder, for example, will reach 42.1dB optical loss for a modulo-103 adder, which is not practical. Optical amplifiers would be required in the middle stages to sustain correct operation, which is an extra overhead. The light loss budget is related to the actual path of the light; however, the number of stages affects the results. In addition, the HPP switch needs 13fJ to switch from cross state to bar state. Hence, the less the number of switches the better. AS-Benes has the fewest number of stages and fewest number of switches among all the discussed architectures. Thus, we conclude that AS-Benes architecture is the best in terms of area, speed and power.

V. CONCLUSION

In this work, we presented three novel architectures for RNS computational units based on shifting and routing. We compared the newly proposed architectures to the ones previously proposed and showed the trade-off between the area

savings and the complexity of the control unit. Moreover, we considered practical implementations of 32, 64, and 128 bit number systems using those architectures, and showed that most of the prior architectures cannot be used to implement such systems except for our newly proposed computational unit based on arbitrary size Benes networks. Evaluation of the area and delay of all networks showed that the AS-Benes based computational unit saves up to 90% of the area and is up to 16 times faster than the other architectures when considering the photon time-of-flight.

REFERENCES

[1] D. Psaltis and D. Casasent, "Optical residue arithmetic: a correlation approach," *Applied Optics*, vol. 18, no. 2, pp. 163–171, 1979.

[2] M. Deryabin, N. Chervyakov, A. Tchernykh, M. Babenko, and M. Shabalina, "High performance parallel computing in residue number system," *International Journal of Combinatorial Optimization Problems and Informatics*, vol. 9, no. 1, pp. 62–67, 2018.

[3] P. M. Matutino, J. Araújo, L. Sousa, and R. Chaves, "Pipelined fpga coprocessor for elliptic curve cryptography based on residue number system," in *2017 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*. IEEE, 2017, pp. 261–268.

[4] J.-C. Bajard, J. Eynard, and N. Merkiche, "Montgomery reduction within the context of residue number system arithmetic," *Journal of Cryptographic Engineering*, vol. 8, no. 3, pp. 189–200, 2018.

[5] P. M. Branch, "The application of the residue number system in digital image processing: Propose a scheme of filtering in spatial domain," *Research Journal of Applied Sciences*, vol. 7, no. 6, pp. 286–292, 2012.

[6] S. Salamat, M. Imani, S. Gupta, and T. Rosing, "Rnsnet: In-memory neural network acceleration using residue number system," in *2018 IEEE International Conference on Rebooting Computing (ICRC)*. IEEE, 2018, pp. 1–12.

[7] N. I. Chervyakov, P. A. Lyakhov, M. V. Valueva, G. V. Valuev, D. I. Kaplun, G. A. Efimenko, and D. V. Gnezdilov, "Area-efficient fpga implementation of minimalistic convolutional neural network using residue number system," in *2018 23rd Conference of Open Innovations Association (FRUCT)*. IEEE, 2018, pp. 112–118.

[8] A. Jain, E. D. Pitchika, and S. Bharadwaj, "An exploration of fpga based multilayer perceptron using residue number system for space applications," in *2018 14th IEEE International Conference on Signal Processing (ICSP)*. IEEE, 2018, pp. 1050–1055.

[9] H. Nakahara and T. Sasao, "A deep convolutional neural network based on nested residue number system," *25th IEEE International Conference on Field Programmable Logic and Applications (FPL),*, pp. 1–6, 2015.

[10] A. R. Omondi and B. Premkumar, *Residue number systems: theory and implementation*. World Scientific, 2007, vol. 2.

[11] K. M. Elleithy and M. A. Bayoumi, "Fast and flexible architectures for rns arithmetic decoding," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 39, no. 4, pp. 226–235, 1992.

[12] A. Huang, Y. Tsunoda, J. W. Goodman, and S. Ishihara, "Optical computation using residue arithmetic," *Applied Optics*, vol. 18, no. 2, pp. 149–162, 1979.

[13] A. Tai, I. Cindrich, J. R. Fienup, and C. Aleksoff, "Optical residue arithmetic computer with programmable computation modules," *Applied optics*, vol. 18, no. 16, pp. 2812–2823, 1979.

[14] L. A. Bakhtiar and M. Hosseinzadeh, "All optical residue arithmetic with micro ring resonators and its application," *Optical and Quantum Electronics*, vol. 48, no. 2, p. 125, 2016.

[15] L. A. Bakhtiar, E. Yaghoubi, S. M. Hamidi, and M. Hosseinzadeh, "Optical rns adder and multiplier," *International Journal of Computer Applications in Technology*, vol. 52, no. 1, pp. 71–76, 2015.

[16] J. Peng, S. Sun, V. K. Narayana, V. J. Sorger, and T. El-Ghazawi, "Residue number system arithmetic based on integrated nanophotonics," *Optics letters*, vol. 43, no. 9, pp. 2026–2029, 2018.

[17] S. Sun, V. K. Narayana, I. Sarpkaya, J. Crandall, R. A. Soref, H. Dalir, T. El-Ghazawi, and V. J. Sorger, "Hybrid photonic-plasmonic nonblocking broadband 5× 5 router for optical networks," *IEEE Photonics Journal*, vol. 10, no. 2, pp. 1–12, 2018.

[18] S. Sun, R. Zhang, J. Peng, V. K. Narayana, H. Dalir, T. El-Ghazawi, and V. J. Sorger, "Mo detector (mod): a dual-function optical modulator-detector for on-chip communication," *Optics express*, vol. 26, no. 7, pp. 8252–8259, 2018.

[19] V. E. Beneš, "Permutation groups, complexes, and rearrangeable connecting networks," *Bell System Technical Journal*, vol. 43, no. 4, pp. 1619–1640, 1964.

[20] C. Chang and R. Melhem, "Arbitrary size benes networks," *Parallel Processing Letters*, vol. 7, no. 03, pp. 279–284, 1997.

[21] S. Sun, A.-H. A. Badawy, V. Narayana, T. El-Ghazawi, and V. J. Sorger, "The case for hybrid photonic plasmonic interconnects (hyppis): Low-latency energy-and-area-efficient on-chip interconnects," *IEEE Photonics Journal*, vol. 7, no. 6, pp. 1–14, 2015.

[22] C. Clos, "A study of non-blocking switching networks," *Bell System Technical Journal*, vol. 32, no. 2, pp. 406–424, 1953.

[23] I. Corporation. (2017) Intel news fact sheet. [Online]. Available: https://newsroom.intel.com/newsroom/wp-content/uploads/sites/11/2017/09/10-nm-icf-fact-sheet.pdf

[24] E. Timurdogan, C. M. Sorace-Agaskar, J. Sun, E. S. Hosseini, A. Biberman, and M. R. Watts, "An ultralow power athermal silicon modulator," *Nature communications*, vol. 5, p. 4008, 2014.

[25] Y. Salamin, P. Ma, B. Baeuerle, A. Emboras, Y. Fedoryshyn, W. Heni, B. Cheng, A. Josten, and J. Leuthold, "100 ghz plasmonic photodetector," *ACS photonics*, vol. 5, no. 8, pp. 3291–3297, 2018.

[26] H. Tsang and Y. Liu, "Nonlinear optical properties of silicon waveguides," *Semiconductor Science and Technology*, vol. 23, no. 6, p. 064007, 2008.

Energy Efficiency of Microring Resonator (MRR)-Based Binary Decision Diagram (BDD) Circuits

Ozan Yakar, Yuqi Nie, Kazumi Wada, Anuradha Agarwal and İlke Ercan, *Junior Member, IEEE*

***Abstract*—The saturation of rapid progress in transistor technology has brought us to a point where the computing systems face fundamental physical limitations. Emerging technologies propose various alternatives and photonic circuits are among promising candidates due to their high operation speed, energy efficient passive components, low crosstalk and appropriateness for parallel computation. In this work, we design a microring resonator (MRR) based Binary Decision Diagram (BDD) NAND logic gate and study its characteristics inline with a MRR-based BDD half adder circuit proposed by Wada et. al. [1]. We analyze energy efficiency limitations of BDD architectures using reversible and irreversible circuit structures. The circuits we focus on in this work are composed of silicon MRR-based switching nodes where the coupling gap and ring size play a key role in the performance of the circuits. We study the physical structure of the circuits as well as dynamics of the information processing, and calculate the fundamental lower bounds on the energy dissipation as a result of computation. We also perform extensive analyses on Lumerical MODE simulations to optimize the energy efficiency based on various factors including waveguide properties, ring radius and gap size. The results we obtain allow us to assess limitations imposed by the physical nature of MRR-based photonic circuits in computation, and compare theory against simulation and hence significantly contribute to the strategic development of this technology as a part of future computers.**

I. Introduction

The exponential improvement in the performance of the transistor technologies came to a halt due to limitations imposed on scaling of electronic device size. The silicon electronics and printed metallic tracks inherently limit the maximum bandwidth achievable by the transistor-based circuits. Moreover, increasing density of the designs leads to a rise in heat generation, therefore causes inefficient operation of the transistors. Hence, the trend towards smaller devices has shifted towards parallelism as the multicore architectures with optical interconnects were proposed in the recent years [2], [3].

The desire to compensate the saturation in the trend towards smaller devices (increasing capacity) at thousand fold pace in every 10 years without an increasing cost led many alternative proposals for the transistor technologies by using carbon nanotubes [4]–[6], graphene like 2D materials [7], [8], spintronic materials [9], [10], novel technologies using quantum computing [11], neuromorphic computing [12], [13] and photonics [1], [14]–[19]. However, the embodiment of these models face a wide range of obstacles cutting across variety of levels spanning device, circuit, architecture and fabrication; and energy efficiency is arguably the most crucial among these challenges.

Optics is widely recognized as the optimal way of transmitting information due to its energy efficient passive components, swiftness, low crosstalk and adaptability for parallel processing. However, optical systems are promising alternatives to current computing paradigms with its energy efficient, low heat generative components and its intrinsic swiftness allows rapid and low energy computation compared to electronics [20], especially with the emergence of the integrated photonic circuits [21] computation through photonic systems is envisaged. All-optical switching may provide a detour to the opto-electronic conversions. Moreover, Larger et. al. [22] and Paquot et. al. [23] proposed various pre- and post-processors allowing us to reduce the computation time and complexity of vector matrix multipliers from $O(N^2)$ to $O(N)$ [13]. These improvements suggest that photonics can provide solutions to limitations of current computing technologies and opto-electronic circuits are among promising alteranatives to transistor technologies.

In this paper, we study MRR-based BDD logic circuits and their efficiency limitation using the reversible and irreversible computing schemes that are operated using thermo-optic and electro-optic effect. We design an MRR-based BDD NAND logic gate and analyze its operation to obtain energy efficiency limitations. We calculate the fundamental lower bounds on the energy dissipation as a result of computation and compare our theoretical findings with extensive Lumerical MODE simulations. We focus on optimizing the energy efficiency based on various factors including waveguide properties, ring radius and gap size. Our findings provide insights into limitations imposed by the physical nature of MRR-based photonic circuits in computation and significantly contribute to the strategic development of this technology as a part of future computers. This paper is organized as follows; we first introduce the fundamental principle and microring based realization of the BDD and present the microring resonator realization of BDD NAND gate in sections II and III. In section IV, we calculate the logic threshold for BDD NAND we designed, formulating the conditions under which accurate computation can be performed. Then, we utilize the computational tools to strengthen our findings by simulating the microring resonator model in

This research is funded by the MIT International Science and Technology Initiatives (MISTI): MIT-Boğaziçi University Seed Fund.

Ozan Yakar, and İlke Ercan are with the Department of Electrical and Electronics Engineering, Boğaziçi University, İstanbul, Turkey.

Yuqi Nie is with School of Physics, Peking University, Beijing, People's Republic of China.

Kazumi Wada is with Department of Materials Science and Engineering, Massachusetts Institute of Technology, USA

Anuradha Agarwal is with Materials Research Laboratory, Massachusetts Institute of Technology, USA

Manuscript received on September 6, 2019

978-1-7281-5221-9/19/$31.00 ©2019 IEEE

Lumerical MODE Solutions to compare the analytic results of our formalism to that of high precision computational tools in the section V. We also perform energy efficiency calculations using different tuning nonlinear effects in section V. Finally, we conclude by closing remarks and comments on our future work in section VI.

II. Microring Resonator (MRR)-Based BDD Logic Circuits

In order to design a universal logic gate, the fundamental criteria are fan-out and bistability, which provides logic level restoration and infinite cascadibility [24]. The passive elements do not provide optical amplification, therefore the design of universal logic gate using the conventional methods is not plausible. However, the use of Y-splitters can allow fan-out with loss. In the use of BDD architecture, the attenuation of control signal does not provide a great loss in the computation of information. Hence, we have a partial fan-out property. Moreover, BDD architecture allows the systems to be designed in a more scalable way compared to the conventional CMOS architectures, which are growing exponentially. Analogously, recombiners allow the superposition of electromagnetic field on that account provide fan-in property, which cannot be realized in electronics [20].

The use of fan-in property creates an irreversibility in the operation since it is impossible to extract the superposed electromagnetic fields after superposition. Hence, microring resonator BDD logic circuits can allow reversible operation through making the system work without using recombiners, therefore allow the reconstruction of inputs by inspecting the outputs.

Despite the lack of bistability in most photonic devices, use of chalcogenide phase-change materials shows there can be bistable photonic logic and memory elements in refs. [14] –[17]. Albeit the lack of bistability in the BDD half-adder using microring resonators in ref. [1], the resonance condition puts a restriction on volatility, therefore allows microring resonator to eliminate this problem for some applications (Video Processing). Footprint is a huge drawback compared to electronic computational paradigms. This can be overcome through the design of 3D structures.

There are several candidates that offer switching, i.e. photonic crystals and Mach-Zehnder interferometers (MZIs). Although photonic crystals are able to offer bistable switching and small footprint [25], microring resonators have lower scattering and absorption rates, and have higher switching due to high field enhancement, which make microring resonators more energy efficient for logic applications. Moreover, the large footprint and true volatility of MZIs make them unsuitable for both switching and memory applications. Microring resonator based devices have a strong potential for ultralow switching energies compared to the state-of-the-art electro-optic modulators with orders of magnitude increase in the energy efficiency.

High index difference between silicon and silicon-oxide (or air) creates a high confinement of light in the silicon waveguides, therefore the silicon microring resonators have smaller footprints compared to the resonators using other mateirals. Moreover, the advanced production techniques for silicon electronics and compatibility with the CMOS technology leads the silicon to be widely used in the design of integrated photonic devices [26]–[31]. Hence, the silicon-based photonic technologies are high in demand as they are constructed using widely known structures in well-established facilities. This emerging trend raises questions around the physical constraints and performance limitations of these optical systems. Although there has been both experimental [32]–[36] and theoretical [37]–[40] analyses of single microring resonators conducted, there has been very few attempts made to study the physical and operational details of photonic logic circuits. In this paper, we address this issue motivated by BDD HA circuit designed in ref. [1] as an illustrative example laying out the physical design and relations between the resonators and the constraints it enforces to perform a logic operation.

III. MRR-Based BDD NAND

In this section, we review the microring-resonator-based Half Adder circuit proposed by Wada et. al [1] and present our design of NAND gate using BDD architecture. We begin by outlining the operation and characteristics of BDD-based logic circuits.

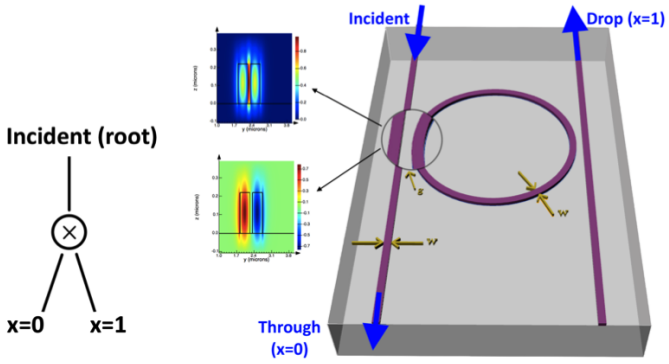


Fig. 1. Symbol of the BDD switching node (left), schematic diagram of a switch using microring resonator on SOI wafer (right). Variable x of the control signal coming from the root (incident) is tested in the nodes, the signal is sent to different leaves depending on the value of x_i, i.e. 0 (Through) or 1 (Drop).

The BDD logic is a computation technique using compressed representation to perform logic operations allowing a higher scaling compared to traditional logic systems using Karnaugh Maps and algebraic manipulations of logic systems [1], [41], [42]. The BDD architecture first proposed by Akers [41], and improved by Bryant [42]. There are many physical realizations of this abstract computational model including Single Electron Tunneling (SET) Transistor Technology [43], quantum computing [44], mirrors [18], and optical microring resonators [1]. In this paper, we focus on physical realization of BDD architecture using microring resonator logic circuits.

The microring resonators have been introduced in ref. [45] and its theory has been developed by [37]–[40]. Here, in this paper, we analyze the physical responses of multi-microring resonator logic devices both analytically and numerically.

Let $x_1, ..., x_n$ be a list of independent binary variables representing the inputs of the of a Boolean function. The signal is processed through serial switching nodes. This mechanism enforces one to have a control over each layer of computation with a unique parameter, x_i. A single BDD node is shown in fig. 1 (left). Variable x_i of the control signal coming from the root is tested in the nodes, the signal is sent to different leaves depending on the value of x_i, i.e. 0 or 1. Then the signal is tested in the next layer, x_{i+1} Therefore, a set of nodes constitute a path connecting the input and output. Microring resonator realization of a BDD switching node is illustrated in fig. 1 (right). Microring resonator is the most common traveling wave-integrated optical cavity resonator, using the total internal reflection (TIR) and frustrated TIR of continuous light along the edges of the waveguides and optical medium with - in general- a $SO(2)$ symmetry, such as microdisk, microsphere, or microring. In general, the waveguides used in the resonator are designed to have very narrow width so that only a single mode, i.e. fundamental mode, is supported [46]. There are two straight waveguides called the bus waveguides in the microring resonator switching node in fig. 1 (right). The control signal is sent through the incident port, and using frustrated TIR, the light splits into two modes, symmetric and antisymmetric shown in fig. 1 (right). Every mode is a superposition of these two modes. Hence, the control signal arriving in the ring interferes with itself therefore creating interference conditions ($2\pi r = m\lambda/n_{eff}$) where $m \in \mathbb{Z}$ and arriving into the second bus, exiting through drop port. Hence, this allows one to make a selection of wavelengths arriving in the drop and through port. Therefore through tuning the n_{eff} allows switching to be done. Thus, the input, x_i, can be realized as the tuning parameter of the ring.

The truth table of NAND operation is shown in table I and its BDD logic diagram is shown in fig. 2 (a). We designed a BDD NAND circuit using microring resonators fig. 2 (b) and (c). In the microring realization of BDD NAND, the control signal is sent through the port Incident and tested in the resonator tuned by the variable a_0 and then resonator tuned by the variable b_0. Variables a_0 and b_0 can be various things such as temperature, voltage or light intensity. The energy efficiency analyses of various tuning effects and physical analyses of BDD NAND will be done in the following sections.

IV. Physical Analyses of Microring-Resonator-Based BDD Circuits

In this section, we formulate the minimum shift required in the frequency (wavelength) domain to conduct an accurate logic operation. Then, we make an energy efficiency analyses for various index tuning effects, i.e. thermo-optic effect (TOE) and electro-optic effect (EOE) and we comment on the $\chi^{(2)}$ effect.

The transmission responses of a single microring resonator with two busses are calculated in [37]. Drop port transmission ($\mathbb{K}$) and through port transmission ($\mathbb{T}$) found in [37] is shown below

$$\mathbb{K} = \frac{\delta\omega^2 k_0}{\Delta\omega^2 + \delta\omega^2} \quad \textit{and} \quad \mathbb{T} = 1 - \frac{\delta\omega^2 t_0}{\Delta\omega^2 + \delta\omega^2} \tag{1}$$

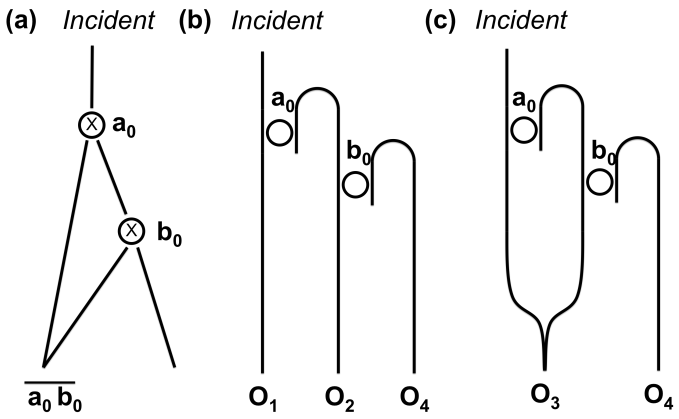


Fig. 2. BDD NAND logic diagram (a), microring-resonator-based realization of reversible BDD NAND logic (b), and microring-resonator-based realization of irreversible BDD NAND logic (c). The inputs a_0 and b_0 are the tuning parameters of the rings. O_1 and O_2 denote the outputs of reversible NAND and O_3 denotes the output of irreversible NAND. O_4 denotes the outputs of the right waveguides of (b) and (c), which corresponds to the AND operation.

where $\delta\omega$ is half-width half-maximum of the transmission, $\Delta\omega = \omega - \omega_0$, ω_0 denotes the resonant frequency, t and k denote the amplitudes. t and k is bound with the relation [37]

$$\sqrt{1 - t_0} + \sqrt{k_0} = 1 \tag{2}$$

Because microring resonators are passive optical elements, t and k can take values in the interval $[0, 1]$. At $\Delta\omega = 0$, the eqs. (1) become,

$$\mathbb{K}_0 = k_0 \quad \textit{and} \quad \mathbb{T}_0 = 1 - t_0 \tag{3}$$

If we shift the resonant frequency by $\tilde{\omega}$, at ω_0 the eqs. (1) become,

$$\mathbb{K}_1 = \frac{k_0}{s^2 + 1} \quad \textit{and} \quad \mathbb{T}_1 = 1 - \frac{t_0}{s^2 + 1} \tag{4}$$

where s is $\tilde{\omega}/\delta\omega$. The microring resonator can be used as a BDD node and switching is done through, shifting the resonances. Thus, the transmission responses of the ring designed to be eqs. (3) for logic-0 and eqs. (4) for logic-1.

A. Analysis of BDD NAND

In this subsection, we calculate the minimum shift required in the frequency (wavelength) domain to conduct an accurate logic operation for irreversible and reversible NAND gates depending on the use of a recombiner. In our analyses, we assume the resonators are placed far apart so that they do not couple with each other.

In the BDD NAND in fig. 2 (b) and (c), the field that is resonant to the ring a_0 goes to the upper guide and its phase evolves as it goes through the waveguide. Then, it goes through the second ring b_0 and its phase evolves again. The field that is not resonant to the ring a_0 goes through the bottom waveguide and its phase evolves while it is going through the waveguide. These two fields can be represented as B^0 and B^{10} respectively. In the reversible NAND in fig. 2 (b), these two fields are not superposed and one can obtain the inputs by seperately looking at the power transmissions in the two

different paths. In the irreversible NAND in fig. 2 (c), these two fields are superposed before arriving to the port N. B^0 and B^{10} can be represented as

$$B^0 = T_a e^{-i\Phi_1} \quad and \quad B^{10} = T_b K_a e^{-i\Phi_2} \tag{5}$$

where B^1 and B^{10} denote transfer function of the EM field at the output port and Φ_1 and Φ_2 denote the phase delay introduced along the waveguides.

In fig. 2 (b), the EM fields coming from two different paths are not superposed before arriving to the output since there is no recombiner. So, we only add the intensities at the output port. The transmission of reversible NAND becomes

$$\mathbb{N}_R = \mathbb{T}_a + \mathbb{T}_b\mathbb{K}_a \tag{6}$$

where $\mathbb{N}_R$ denotes power transmission of reversible NAND, which is the sum of transmissions through two optical paths.

In the analysis of irreversible NAND in fig. 2 (c), we assume that the merging is adiabatic. The EM fields coming from two different paths need to be superposed before arriving to the output. So, the total EM field becomes

$$N = \frac{B^0 + B^{10}}{\sqrt{2}} \tag{7}$$

where, N denotes transfer function of the EM field at the output port. The transmission becomes,

$$\mathbb{N}_I = \frac{1}{2}(\mathbb{T}_a + \mathbb{T}_b\mathbb{K}_a + 2\sqrt{\mathbb{T}_a\mathbb{T}_b\mathbb{K}_a}\cos(\Delta\Phi + \Delta p)) \tag{8}$$

There is an interference raised from the optical pathlength differences at the recombiner, this will cause variations in the transmission responses and will set a threshold for the logic operation. This problem can be overcome either by spending more energy to separate two logic states or tuning the pathlength difference in a way that the logic is symmetric. Although the latter can be done in theory and simulations, variations in manufacturing and local charge and temperature distributions will alter the transmission responses and lead the interference.

a_0	b_0	$\overline{a_0 b_0}$	O_1	O_2	O_3	O_4
0	0	1	$\mathbb{T}_0$	$\mathbb{K}_0\mathbb{T}_0$	$\frac{1}{2}(\sqrt{\mathbb{T}_0} \pm \sqrt{\mathbb{K}_0\mathbb{T}_0})^2$	$\mathbb{K}_0\mathbb{K}_0$
0	1	1	$\mathbb{T}_0$	$\mathbb{K}_0\mathbb{T}_1$	$\frac{1}{2}(\sqrt{\mathbb{T}_0} \pm \sqrt{\mathbb{K}_0\mathbb{T}_1})^2$	$\mathbb{K}_0\mathbb{K}_1$
1	0	1	$\mathbb{T}_1$	$\mathbb{K}_1\mathbb{T}_0$	$\frac{1}{2}(\sqrt{\mathbb{T}_1} \pm \sqrt{\mathbb{K}_1\mathbb{T}_0})^2$	$\mathbb{K}_1\mathbb{K}_0$
1	1	0	$\mathbb{T}_1$	$\mathbb{K}_1\mathbb{T}_1$	$\frac{1}{2}(\sqrt{\mathbb{T}_1} \pm \sqrt{\mathbb{K}_1\mathbb{T}_1})^2$	$\mathbb{K}_1\mathbb{K}_1$

TABLE I
LOGIC TABLE OF NAND GATE AND TRANSMISSION RESPONSES ACCORDING TO THE INPUTS a_0 AND b_0 OF REVERSIBLE (O_1 AND O_2) NAND AND ENVELOPE CURVES OF THE IRREVERSIBLE (O_3) NAND. O_1 AND O_2 DENOTES THE LEFT AND CENTER WAVEGUIDES IN FIG. 2 (B). O_3 DENOTES THE BOUNDARIES (ENVELOPE CURVES) OF THE IRREVERSIBLE OPERATION SHOWN IN EQ. (8). O_4 DENOTES THE RIGHT WAVEGUIDE, WHICH CORRESPONDS TO THE AND OPERATION.

In table I, the truth table and the corresponding interval of the physical output are shown for reversible and irreversible NAND and AND gates. The physical outputs of the reversible NAND are represented as O_1 and O_2 and envelope curves of the irreversible NAND represented as O_3. O_1 and O_2 represents the left and center waveguides in fig. 2 (b) and O_3 represents the minimum and maximum values of each input derived in eq. (8). O_4 denotes the right waveguide, which corresponds to the AND operation.

Issue	Reversible	Irreversible
Interference	No.	Imposes a higher logic threshold.
Cascadibility	Not energy efficient, abrupt increase in footprint.	Comparatively more energy efficient and compact.
Information	Can be retrieved from the outputs.	Inputs can not be retrieved.

TABLE II
COMPARISON OF REVERSIBLE AND IRREVERSIBLE BDD NAND GATES USING MICRORING RESONATOR SWITCHING NODES BY MEANS OF INTERFERENCE, CASCADIBILITY AND INFORMATION LOSS.

In table II, the comparison of the reversible and irreversible BDD NAND gates using microring resonator switching nodes are shown. In the reversible computation, we don't observe interference and we can retrieve the inputs by looking at both outputs at O_1 and O_2 ports in fig. 2, but when we complexity increases, we need to use more switches compared to the irreversible computing that increases the energy consumption and footprint abruptly. In the irreversible computing, interference sets a logic threshold for an accurate computation, as well as the information cannot be retrieved from the output port O_3.

BDD HA is symmetric each sides go through a same kind and number of operations, however, in NAND some part of the incoming beam goes through two resonators and the remaining part goes through one resonator. Although 5 NAND gates are needed to make an HA in the CMOS architecture, there needs to be two switching nodes to make NAND and 3 for HA.

V. IMPLICATIONS FOR LOGIC APPLICATIONS

In this section, we present both theoretical results and simulations with varFDTD algorithm of Lumerical for irreversible BDD NAND. Then we derive the effect of various design parameters on the threshold for an accurate logic operation and using these results, we find the minimum energy required to make the logic operation using different nonlinear tuning effects.

The crosstalk in the computation is not completely avoidable. Even if, a completely symmetric structure is designed, there may be some variations while the circuits are manufactured. Ring resonators are very sensitive to these variations. Thus, the response of the circuits may change significantly for a small change in the dimensions of the components. Although we cannot overcome these variations in production and tuning in without consuming too much energy, we may still continue to do the logic operation with a carefully chosen threshold.

Logic mapping makes one to be able to envisage the possible outputs and their corresponding inputs. While the device is operated the physical response of the designated inputs must give well defined outputs and the output sets must be disjoint. When these conditions are satisfied, it is appropriate to define regions, i.e. envelope curves, to quantize the outputs. Therefore, one can define a threshold for the decision making.

In the irreversible NAND structure in fig. 2 (c), we need to group inputs (a_0b_0) as $\{00, 10, 10\}$ and $\{11\}$ while inspecting the N port (see Table I). We need to design the physical circuit in a way that the two groups of outputs, i.e. 0 and 1 state, are well-defined and disjoint.

When the boundaries of two sets physical outputs, i.e., transmission intensities, overlap, the logic state will be unknown. If the boundary curves of two outputs intersect, there is a probability that the actual transmission curves of the manufactured system is crossing each other because there can be some fluctuations caused by the environment, flaws in the production or tuning mechanism. To assure to have a stable system, there needs to be a separation of the regions that are bounded by the envelope curves.

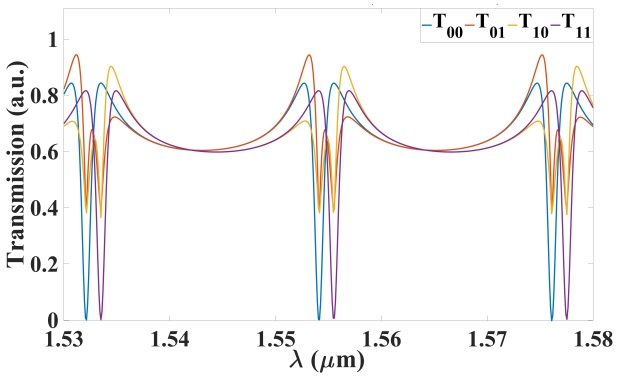


Fig. 3. The theoretical transmission response of the Irreversible BDD NAND structure with $|K_1| = 0.8$, $|K_2| = 0.75$ and the distance between two resonances as $1.4nm$ and $\delta\lambda = 0.8nm$.

In fig. 3, the theoretical transmission responses of all four inputs $a_0b_0 = 11$ are shown. If we do the logic operation above $1.55\mu m$, a threshold can be set below 0.4, so that we can group two inputs.

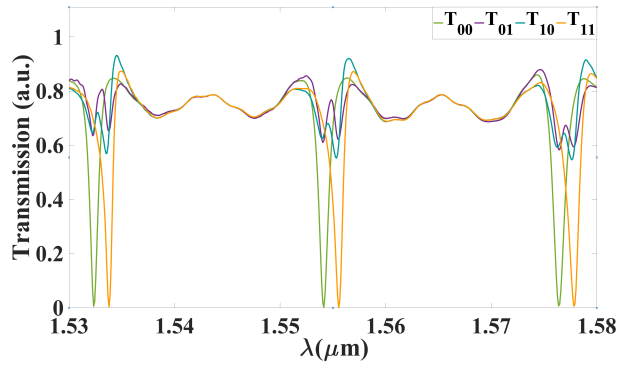


Fig. 4. The Lumerical varFDTD simulation results of the transmission responses of the logical inputs of the Irreversible BDD NAND structure with $r = 5\mu m$, $g = 0.1\mu m$, $w = 0.4\mu m$, $h = 0.22\mu m$, and the temperature change for switching is $\Delta T = 15K$. The simulations are conducted using mesh size $10pm$, mesh accuracy 5, simulation time $22ps$.

In fig. 4, the transmission values for the irreversible NAND obtained from Lumerical Mode are shown for all a_0b_0 are shown. In our design, the radius, gap, waveguide width are set to be $5\mu m$, $0.1\mu m$, $0.4\mu m$ respectively. $220nm$ thick SOI rings are used in order to be compatible with the CMOS technology. The mesh size, mesh accuracy and simulation time are $10pm$, 5 and $22ps$.

The system is operated at $\omega_0 + s\delta\omega$ although it is going to give the same result for operation at $\omega_0 - s\delta\omega$. The central frequency ω_0 is chosen as 0 and frequency is represented as $\omega/\delta\omega$ for the sake of simplicity.

In fig. 6, the envelope of logic-1 intersects with the one of logic-0 and there is an undefined region for logic operation. This prevents the logic operation.

In fig. 7, $0.828\delta\omega$ shift of the resonances will create a threshold for logic operation at ω_0. So, if the shift of two peaks are more than $0.828\delta\omega$, the outputs are well-defined and disjoint, therefore an accurate logic operation is conducted. As the shift between the resonant peaks $s\delta\omega$ is increased more and more, the transmission amplitudes of two logic states will be more and more separated. This increase will make the device more noise-tolerant. This is illustrated in figs. 8. Moreover, the fluctuations caused by the phase differences $\Delta\Phi$ will lose its importance. Hence, the effect of the phase modulation will decrease with an increase of the switching energy due to the decrease in the crosstalk as seen in eq. 8. That means the phase sensitivity decreases.

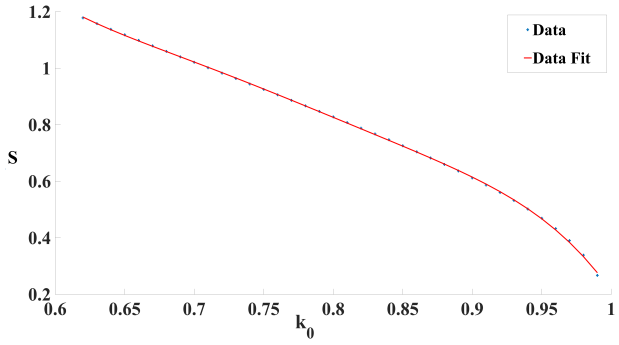


Fig. 5. The effect of (coupling constant) k_0 on the minimum shift per $\delta\omega$ is shown above. The 5^{th} order polynomial fit with $R^2 = 0.99988$ is found. In these calculations, the change in coupling constant due to tuning effects is ignored. The material losses are also ignored since they have no effect on logic in Silicon due to isotropy of the Si-waveguide.

In fig. (5), the minimum shift per half-width half maximum ($\delta\omega$) to conduct an accurate logic operation (s) versus k_0 is shown. In these calculations, the change in coupling constant with tuning effects is ignored. The material losses are also ignored since they have no effect on logic in Silicon due to isotropy of the Si-waveguide. We can fit a 5^{th}-order polynomial to model this behavior.

$$s = -410.4456k_0^5 + 1581.5722k_0^4 - 2431.0294k_0^3 + 1862.5338k_0^2 - 712.9708k_0^1 + 110.5514 \quad (9)$$

where linear regression is 0.99988 and root mean square error is 0.0027. Here, one can see the minimum shift required for logic operation decreases with increasing k_0. Also, as $\delta\omega$ increases the threshold increases linearly. The dependence of energy will be calculated in the following section.

To decrease the minimum shift required $\tilde{\omega} = s\delta\omega$, quality factor can be increased in expense for increasing increasing footprint and cavity lifetime, therefore the operation time.

The figure-of-merit (FoM) here is $I/E.A$, where I is the information capacity, E is the bit energy, A is the footprint. In our design, we decided $Q < 2000$ in order to have a faster operation speed than electronics.

According to the eq. (9), the minimum energy for switching is spent in the critical coupling $\tilde{\omega} = 0.26\delta\omega$. In the next section, we are going to calculate the minimum energy for switching using different nonlinear effects.

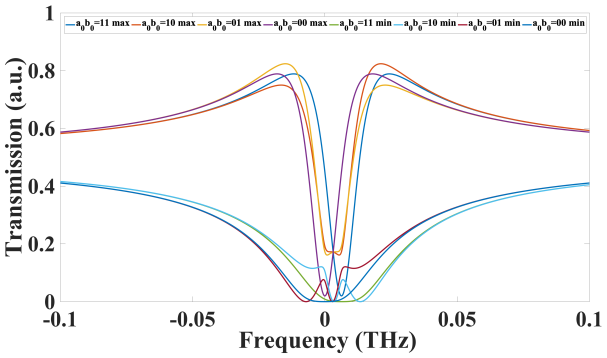


Fig. 6. The theoretical transmission response of the Irreversible NAND Structure with $k_0 = 0.8$ and the distance between two resonances as $0.5\delta\omega$.

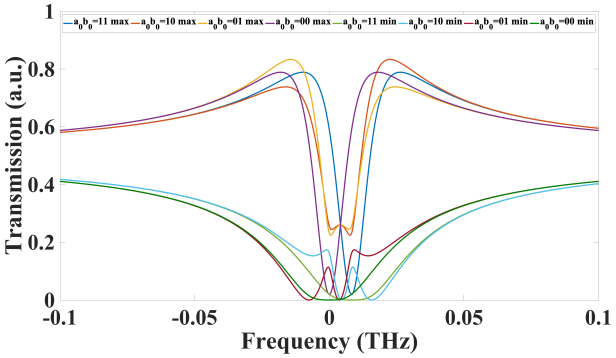


Fig. 7. The theoretical transmission response of the Irreversible NAND Structure with $k_0 = 0.8$, and the distance between two resonances as $0.828\delta\omega$.

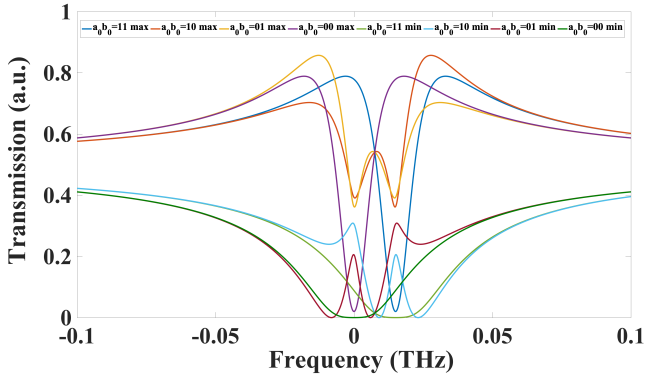


Fig. 8. The theoretical transmission response of the Irreversible NAND Structure with k_0=0.8, and the distance between two resonances as $1.5\delta\omega$. This is above the limiting case for logic operation.

A. Energy Analyses

In this subsection, we make an energy efficiency analyses for various index tuning effects, i.e. thermo-optic effect (TOE) and electro-optic effect (EOE) and we comment on the $\chi^{(2)}$ effect.

B. Using Thermo-optic Effect

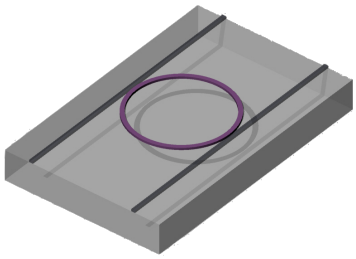


Fig. 9. Ring resonator structure divided into 3 parts: 1. Ring (purple), 2. bus waveguides (dark grey), 3. BOX layer (grey)

In order to calculate the energy requirement for tuning with TOE, the heat required will be $Q = mc\Delta T$, where m is the mass, c is the specific heat of the material and ΔT is the difference between initial and final temperatures $(T_f - T_i)$. And the mass will be calculated by $m = \rho V$, where ρ is the density of the material. In our calculations, we can divide the heat dissipation into 3 parts as in fig. 9, i.e., ring, bus waveguides and the BOX.

The volume of the ring is $V_{ring} = \pi(r_{out}^2 - r_{in}^2)h$. In the ring resonator we designed $r_{out} = r$ and $r_{in} = r - w$. So the volume of the single ring can be represented as $V_{ring} = \pi wh(2r - w)$. We can plug the numbers that we used in our design, i.e. $r = 5\mu m$, $w = 0.4\mu m$, $h = 0.22\mu m$. So $V_{ring} = 2.654\mu m^3$. Density of Silicon crystal is $\rho_{Si} = 2.328g.cm^{-3}$ and specific heat of Silicon is $c_{Si} = 0.71J.g^{-1}.K^{-1}$. So, the heat needed to change the ring from 0 state to 1 state is

$$Q_{ring} = \rho_{Si}c_{Si}V_{ring}\Delta T = 4.3868\times10^{-12}[J.K^{-1}]\Delta T \quad (10)$$

In the simulations, one can see that the temperature difference is in the order of $10K$ for one $\delta\omega$ shift. So,

$$Q_{ring} \approx 50pJ \quad (11)$$

In our design the waveguides are $25\mu m$ long.[1] So, $V_{bus} = 2lwh = 4.4\mu m^3$. And

$$Q_{bus} = \rho_{Si}c_{Si}V_{bus}\Delta T = 7.2727\times10^{-12}[J.K^{-1}]\Delta T \approx 80pJ \quad (12)$$

The energy required for heating the BOX layer is

$$Q_{BOX} = \rho_{SiO_2}c_{SiO_2}V_{BOX}\Delta T = 2.3214 \times 10^{-9}[J.K^{-1}]\Delta T \approx 30nJ \quad (13)$$

[1]The volume of the BOX layer is $V_{BOX} = w_x t_x l = 16\mu m \times 3\mu m \times 25\mu m = 1200\mu m^3$. Density of Silicon crystal is $\rho_{SiO_2} = 2.65g.cm^{-3}$ and specific heat of Silicon is $c_{SiO_2} = 0.68 - 0.73J.g^{-1}.K^{-1}$.

There is 3 orders of magnitude difference in the minimum energy required for local heating –i.e. heating the ring alone– versus heating the region including the bus and BOX which can be considered as partially uniform heating."

C. Using Electro-optic Effect

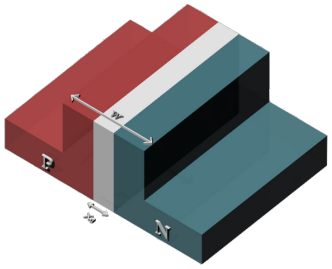


Fig. 10. pn-junction waveguide of width w and depletion width of x_d

For Silicon, the refractive index change with carriers are stated in ref. [47]

$$\Delta n = -8.8 \times 10^{-22} \Delta N - 8.5 \times 10^{-18} \Delta P^{0.8} \tag{14}$$

Say p and n-doped regions are doped equally, i.e. $P = N$. So, the depletion region will be symmetric. Therefore the average index change becomes,

$$< \Delta n > \simeq \frac{\Delta n_N \left(\frac{w-x_d}{2}\right) + \Delta n_P \left(\frac{w-x_d}{2}\right) + 0.x_d}{w} \tag{15}$$

Simplifying the above equation,

$$\begin{aligned} < \Delta n > &= (\Delta n_N + \Delta n_P) \left(\frac{w - x_d}{2w}\right) \\ &= \left(\frac{\Delta n_N + \Delta n_P}{2}\right) \left(1 - \frac{x_d}{w}\right) \end{aligned} \tag{16}$$

The width of the depletion region is equal to

$$x_d = \left[\frac{\epsilon(V_0 - V)}{q} \left(\frac{1}{N} + \frac{1}{P}\right)\right]^{1/2} \tag{17}$$

Plugging eq. (17) into (16),

$$\begin{aligned} < \Delta n > = &\left(\frac{\Delta n_N + \Delta n_P}{2}\right) ... \\ &\left(1 - \frac{1}{w}\sqrt{\frac{2\epsilon(V_0 - V)}{q} \left(\frac{1}{N} + \frac{1}{P}\right)}\right) \end{aligned} \tag{18}$$

In literature, the doping, on average, is around $10^{18} cm^{-3}$. So, we can assume, the doping of p and n-regions are $10^{18} cm^{-3}$. Intrinsic carrier concentration of silicon at room temperature is $n_i = 1.5 \times 10^{10} cm^{-3}$. So, $P_n = \frac{n_i^2}{N} = 2.25 \times 10^2 cm^{-3}$ and $N_p = \frac{n_i^2}{P} = 2.25 \times 10^2 cm^{-3}$. Because there is 16 orders of magnitude difference between h^+ and e^- concentrations on each side, we can neglect the minority carrier concentrations. Plugging the numbers for p and n sides separately gives us $\Delta n_N = -8.8 \times 10^{-4}$ and $\Delta n_P = -2.34 \times 10^{-3}$. Electronic charge equals to $q = 1.6 \times 10^{-19} C$, dielectric constant of silicon is $\epsilon = \epsilon_r \epsilon_0 = 11.7 \times 8.85 \times 10^{-14} F/cm$. Inserting these numbers into the eq. (18),

$$\begin{aligned} < \Delta n > = &- \left(\frac{8.8 \times 10^{-4} + 2.34 \times 10^{-3}}{2}\right) ... \\ &\left(1 - \frac{1}{w}\sqrt{\frac{2 \times 11.7 \times 8.85 \times 10^{-14}}{1.6 \times 10^{-19}} 2 \times 10^{-18} \sqrt{V_0 - V}}\right) \end{aligned} \tag{19}$$

V_0 is equal to $V_0 = \frac{kT}{q} ln(\frac{NP}{n_i^2}) = 0.92V$

$$< \Delta n > = -0.0015575(1 - 0.1272\sqrt{0.92 - V}) \tag{20}$$

If we reverse bias the pn junction, current becomes $I = I_0 \left(e^{V/V_T} - 1\right) \approx -I_0$. Because the power is directly proportional to the current, changing the index with reverse bias is more efficient than doing it with forward bias. The index change we need can be found using dn/dT value and the temperature change we assumed in the previous section. So, the index change we need is around $\Delta n = 1.87 \times 10^{-4} \times \Delta T \approx 2 \times 10^{-3}$. So the voltage we need is around 3-4 volts. Hence the power we need is around $10^{-8} W = 10 nW$. So, if we do the switching in $1ms$, the energy spent will be $10pJ$, and if we do the switching in $1ns$ the switching energy will be $10aJ$.

VI. Conclusions and Future Work

In this paper, we perform a physical analysis of the BDD NAND with microring resonator switching nodes, and show that the magnitude of frequency shift in switching and the optical pathlength are two crucial factors in the design of microring resonator logic circuits because of the interference phenomenon. The high sensitivity of microring resonators sets a threshold for logic operation. When the waveguide lengths are not optimized, the minimum distance of the two resonances at the O_3 port peaks should be $s\delta\omega$ away from each other, where s is found from eq. (9), and where $\delta\lambda$ is the half-width half-maximum of the transmission. The energy efficiencies of the thermo-optic effect and the electro-optic effect for silicon are thus obtained under ideal operating conditions.

The $\chi^{(2)}$ effect of silicon is obtained by applying a strained layer of Si_3N_4 in ref. [48]. Jacobsen et al. found the $\chi^{(2)}_{zzx}$ to be $15pmV^{-1}$ and with the group index enhancement, they have reached $\chi^{(2)}_{enh} = 830pmV^{-1}$ [48]. Adding a capacitor avoids the application of a dc current for tuning, such that the switching time is not limited by charge storage effects of pn junctions, possibly leading to lower energies and increased speed for computation. As a part of our future work, $\chi^{(2)}$ effect of silicon for microring resonators, and energy efficiencies of photonic crystal cavities will both be analyzed.

Acknowledgements

The authors would like to thank MIT International Science and Technology Initiatives (MISTI) for funding this collaboration via MIT-Boğaziçi University Seed Fund. Ozan Yakar would also like to thank Ms. Blanca Ameiro Mateos for her generous help with technical drawing of the BDD circuit schematic.

REFERENCES

[1] S. Lin, Y. Ishikawa, and K. Wada, "Demonstration of optical computing logics based on binary decision diagram," *Optics Express*, vol. 20, no. 2, pp. 1378–1384, 2012.

[2] L. Hardesty, "Selling chip makers on optical computing," Nov 2009. [Online]. Available: http://news.mit.edu/2009/optical-computing

[3] D. Miller, "Device requirements for optical interconnects to cmos silicon chips," in *Integrated Photonics Research, Silicon and Nanophotonics and Photonics in Switching*. Optical Society of America, 2010, p. PMB3. [Online]. Available: http://www.osapublishing.org/abstract.cfm?URI=PS-2010-PMB3

[4] A. Bachtold, P. Hadley, T. Nakanishi, and C. Dekker, "Logic circuits with carbon nanotube transistors," *Science*, vol. 294, no. 5545, pp. 1317–1320, 2001.

[5] S. J. Tans, A. R. Verschueren, and C. Dekker, "Room-temperature transistor based on a single carbon nanotube," *Nature*, vol. 393, no. 6680, p. 49, 1998.

[6] M. M. S. Aly, M. Gao, G. Hills, C.-S. Lee, G. Pitner, M. M. Shulaker, T. F. Wu, M. Asheghi, J. Bokor, F. Franchetti *et al.*, "Energy-efficient abundant-data computing: The n3xt 1,000 x," *Computer*, vol. 48, no. 12, pp. 24–33, 2015.

[7] M. Chhowalla, D. Jena, and H. Zhang, "Two-dimensional semiconductors for transistors," *Nature Reviews Materials*, vol. 1, no. 11, p. 16052, 2016.

[8] R. Sordan, F. Traversi, and V. Russo, "Logic gates with a single graphene transistor," *Applied Physics Letters*, vol. 94, no. 7, p. 51, 2009.

[9] D. Pesin and A. H. MacDonald, "Spintronics and pseudospintronics in graphene and topological insulators," *Nature materials*, vol. 11, no. 5, p. 409, 2012.

[10] S. Gardelis, C. Smith, C. Barnes, E. Linfield, and D. Ritchie, "Spin-valve effects in a semiconductor field-effect transistor: A spintronic device," *Physical Review B*, vol. 60, no. 11, p. 7764, 1999.

[11] T. D. Ladd, F. Jelezko, R. Laflamme, Y. Nakamura, C. Monroe, and J. L. OBrien, "Quantum computers," *Nature*, vol. 464, no. 7285, p. 45, 2010.

[12] B. J. Shastri, A. N. Tait, T. Ferreira de Lima, M. A. Nahmias, H.-T. Peng, and P. R. Prucnal, "Principles of neuromorphic photonics," *Unconventional Computing: A Volume in the Encyclopedia of Complexity and Systems Science, Second Edition*, pp. 83–118, 2018.

[13] D. Woods and T. J. Naughton, "Optical computing: Photonic neural networks," *Nature Physics*, vol. 8, no. 4, p. 257, 2012.

[14] C. Ríos, M. Stegmaier, P. Hosseini, D. Wang, T. Scherer, C. D. Wright, H. Bhaskaran, and W. H. Pernice, "Integrated all-photonic non-volatile multi-level memory," *Nature Photonics*, vol. 9, no. 11, p. 725, 2015.

[15] C. Rios, M. Stegmaier, Z. Cheng, N. Youngblood, C. D. Wright, W. H. Pernice, and H. Bhaskaran, "Controlled switching of phase-change materials by evanescent-field coupling in integrated photonics," *Optical Materials Express*, vol. 8, no. 9, pp. 2455–2470, 2018.

[16] M. Stegmaier, C. Ríos, H. Bhaskaran, C. D. Wright, and W. H. Pernice, "Nonvolatile all-optical $1\times$ 2 switch for chipscale photonic networks," *Advanced Optical Materials*, vol. 5, no. 1, p. 1600346, 2017.

[17] Z. Cheng, C. Ríos, N. Youngblood, C. D. Wright, W. H. Pernice, and H. Bhaskaran, "Device-level photonic memories and logic applications using phase-change materials," *Advanced Materials*, vol. 30, no. 32, p. 1802435, 2018.

[18] T. Chattopadhyay, "Optical logic gates using binary decision diagram with mirrors," *Optics & Laser Technology*, vol. 54, pp. 159–169, 2013.

[19] A. Fushimi and T. Tanabe, "All-optical logic gate operating with single wavelength," *Optics express*, vol. 22, no. 4, pp. 4466–4479, 2014.

[20] H. J. Caulfield and S. Dolev, "Why future supercomputing requires optics," *Nature Photonics*, vol. 4, no. 5, p. 261, 2010.

[21] AIM-Photonics-Academy, "Integrated photonic systems roadmap 2017," March 2018.

[22] L. Larger, M. C. Soriano, D. Brunner, L. Appeltant, J. M. Gutiérrez, L. Pesquera, C. R. Mirasso, and I. Fischer, "Photonic information processing beyond turing: an optoelectronic implementation of reservoir computing," *Optics express*, vol. 20, no. 3, pp. 3241–3249, 2012.

[23] Y. Paquot, F. Duport, A. Smerieri, J. Dambre, B. Schrauwen, M. Haelterman, and S. Massar, "Optoelectronic reservoir computing," *Scientific reports*, vol. 2, p. 287, 2012.

[24] D. A. Miller, "Are optical transistors the logical next step?" *Nature Photonics*, vol. 4, no. 1, p. 3, 2010.

[25] M. Notomi, A. Shinya, S. Mitsugi, G. Kira, E. Kuramochi, and T. Tanabe, "Optical bistable switching action of si high-q photonic-crystal nanocavities," *Optics Express*, vol. 13, no. 7, pp. 2678–2687, 2005.

[26] W. Bogaerts, R. Baets, P. Dumon, V. Wiaux, S. Beckx, D. Taillaert, B. Luyssaert, J. Van Campenhout, P. Bienstman, and D. Van Thourhout, "Nanophotonic waveguides in silicon-on-insulator fabricated with cmos technology," *Journal of Lightwave Technology*, vol. 23, no. 1, p. 401, 2005.

[27] W. M. Green, M. J. Rooks, L. Sekaric, and Y. A. Vlasov, "Ultra-compact, low rf power, 10 gb/s silicon mach-zehnder modulator," *Optics express*, vol. 15, no. 25, pp. 17 106–17 113, 2007.

[28] J. Fedeli, E. Augendre, J. Hartmann, L. Vivien, P. Grosse, V. Mazzocchi, W. Bogaerts, D. Van Thourhout, and F. Schrank, "Photonics and electronics integration in the helios project," in *Group IV Photonics (GFP), 2010 7th IEEE International Conference on*. IEEE, 2010, pp. 356–358.

[29] C. Gunn *et al.*, "Cmos photonics for high-speed interconnects," *IEEE micro*, vol. 26, no. 2, pp. 58–66, 2006.

[30] C. Gunn, "Silicon photonics: Poised to invade local area networks-by integrating optical functions into a microprocessor fabrication process, engineers can achieve the performance of optical links within the," *Photonics Spectra*, vol. 40, no. 3, pp. 62–69, 2006.

[31] W. Bogaerts, P. De Heyn, T. Van Vaerenbergh, K. De Vos, S. Kumar Selvaraja, T. Claes, P. Dumon, P. Bienstman, D. Van Thourhout, and R. Baets, "Silicon microring resonators," *Laser & Photonics Reviews*, vol. 6, no. 1, pp. 47–73, 2012.

[32] S. T. Chu, B. E. Little, W. Pan, T. Kaneko, S. Sato, and Y. Kokubun, "An eight-channel add-drop filter using vertically coupled microring resonators over a cross grid," *IEEE Photonics Technology Letters*, vol. 11, no. 6, pp. 691–693, 1999.

[33] J. Hryniewicz, P. Absil, B. Little, R. Wilson, and P.-T. Ho, "Higher order filter response in coupled microring resonators," *IEEE Photonics Technology Letters*, vol. 12, no. 3, pp. 320–322, 2000.

[34] B. E. Little, J. Foresi, G. Steinmeyer, E. Thoen, S. Chu, H. Haus, E. P. Ippen, L. Kimerling, and W. Greene, "Ultra-compact si-sio 2 microring resonator optical channel dropping filters," *IEEE Photonics Technology Letters*, vol. 10, no. 4, pp. 549–551, 1998.

[35] R. Wakabayashi, M. Fujiwara, K.-i. Yoshino, Y. Nambu, M. Sasaki, and T. Aoki, "Time-bin entangled photon pair generation from si micro-ring resonator," *Optics express*, vol. 23, no. 2, pp. 1103–1113, 2015.

[36] H. Li, B. Dong, Z. Zhang, H. F. Zhang, and C. Sun, "A transparent broadband ultrasonic detector based on an optical micro-ring resonator for photoacoustic microscopy," *Scientific reports*, vol. 4, p. 4496, 2014.

[37] B. E. Little, S. T. Chu, H. A. Haus, J. Foresi, and J.-P. Laine, "Microring resonator channel dropping filters," *Journal of lightwave technology*, vol. 15, no. 6, pp. 998–1005, 1997.

[38] M. Hammer, K. R. Hiremath, and R. Stoffer, "Analytical approaches to the description of optical microresonator devices," in *AIP conference proceedings*, vol. 709, no. 1. AIP, 2004, pp. 48–71.

[39] J. E. Heebner, V. Wong, A. Schweinsberg, R. W. Boyd, and D. J. Jackson, "Optical transmission characteristics of fiber ring resonators," *IEEE journal of quantum electronics*, vol. 40, no. 6, pp. 726–730, 2004.

[40] P. M. Alsing, E. E. Hach III, C. C. Tison, and A. M. Smith, "Quantum-optical description of losses in ring resonators based on field-operator transformations," *Physical Review A*, vol. 95, no. 5, p. 053828, 2017.

[41] S. B. Akers, "Binary decision diagrams," *IEEE Transactions on computers*, no. 6, pp. 509–516, 1978.

[42] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *Computers, IEEE Transactions on*, vol. 100, no. 8, pp. 677–691, 1986.

[43] N. Asahi, M. Akazawa, and Y. Amemiya, "Single-electron logic device based on the binary decision diagram," *IEEE Transactions on Electron Devices*, vol. 44, no. 7, pp. 1109–1116, 1997.

[44] N. Yoshikawa, F. Matsuzaki, N. Nakajima, and K. Yoda, "Design and component test of a 1-bit rsfq microprocessor," *Physica C: Superconductivity*, vol. 378, pp. 1454–1460, 2002.

[45] E. Marcatili, "Bends in optical dielectric guides," *Bell System Technical Journal*, vol. 48, no. 7, pp. 2103–2132, 1969.

[46] I. Chremmos, O. Schwelb, and N. Uzunoglu, *Photonic microresonator research and applications*. Springer, 2010, vol. 156.

[47] G. T. Reed and C. J. Png, "Silicon optical modulators," *Materials Today*, vol. 8, no. 1, pp. 40–50, 2005.

[48] R. S. Jacobsen, K. N. Andersen, P. I. Borel, J. Fage-Pedersen, L. H. Frandsen, O. Hansen, M. Kristensen, A. V. Lavrinenko, G. Moulin, H. Ou *et al.*, "Strained silicon as a new electro-optic material," *Nature*, vol. 441, no. 7090, p. 199, 2006.

An n-bit adder realized via coherent optical parallel computing

1st Bogdan Reznychenko
Probayes
Grenoble, France
bodgan.reznychenko@probayes.com

2nd Emmanuel Mazer
Probayes
Grenoble, France
emazer@probayes.com

3rd Maurizio Coden
University of Padova
Padova, Italy
maurizio.coden@unipd.it

4th Elisabetta Collini
University of Padova
Padova, Italy
elisabetta.collini@unipd.it

5th Carlo Nazareno DiBenedetto
Institute for Physical and Chemical Processes
Bari, Italy
carlonazareno@libero.it

6nd Ariela Donval
Elbit Systems
Jerusalem, Israel
ariela.donval@elbitsystems.com

7th Barbara Fresch
University of Padova
Padova, Italy
barbara.fresch@unipd.it

8th Hugo Gattuso
University of Liège
Liège, Belgium
hgattuso@uliege.be

9th Noam Gross
Elbit Systems
Jerusalem, Israel
noam.gross@elbitsystems.com

10th Yossi Paltiel
Hebrew University of Jerusalem
Jerusalem, Israel
paltiel@mail.huji.ac.il

11th Francoise Remacle
University of Liège
Liège, Belgium
FRemacle@uliege.be

12th Marinella Striccoli
Institute for Physical and Chemical Processes
Bari, Italy
m.striccoli@ba.ipcf.cnr.it

***Abstract*—The quantum properties of nanosystems present a new opportunity to enhance the power of classical computers, both for the parallelism of the computation and the speed of the optical operations. In this paper we present the COPAC project aiming at development of a ground-breaking nonlinear coherent spectroscopy combining optical addressing and spatially macroscopically resolved optical readout. The discrete structure of transitions between quantum levels provides a basis for implementation of logic functions even at room temperature. Exploiting the superposition of quantum states gives rise to the possibility of parallel computation by encoding different input values into transition frequencies. As an example of parallel single instruction multiple data calculation by a device developed during the COPAC project, we present a n-bit adder, showing that due to the properties of the system, the delay of this fundamental circuit can be reduced.**

***Index Terms*—quantum optical computing, parallel logic, non von Neumann architectures**

I. Introduction

There has been much effort recently to create classical logic units exploiting quantum properties of matter [1]. We present here the Coherent Optical parallel computing (COPAC) project, that is a transformative novel area in computing both because of the technology, coherent information transfer by ultrafast laser addressing of engineered quantum dots arrays and because of the specialized parallel processing of large amounts of information.

Funding: EC FET open H2020 project COPAC (#766563).

Since 2001, the group at ULiege[1] and the HUJI[2] groups have developed and demonstrated that complex logic operations can be implemented at the molecular- and nanoscale by exploiting and controlling the dynamics of the response of the internal degrees of freedom of confined systems to various inputs: optical, chemical and electrical [2]–[4].

Partners involved in COPAC already collaborated in demonstrating logic processing by coherent optical addressing by a sequence of three laser pulses on dyes tethered on DNA in a 2D photon echo set-up [5], [6].

As the COPAC project aims at designing a real machine, the IPCF[3] is in charge of the synthesis of semiconductor quantum dots in quantum confinement regime (as CdSe) with electronic transitions in the visible range (from 530 to 650 nm, according to QD size) to achieve nanostructure with control on the size, shape and crystallinity [7]. These QDs are used by UNIPD[4] to provide 2D photon echo measures which could in turn be interpreted thanks to theoretical models developed in collaboration with ULiege and HUJI [4]. This experiments are conducted on solutions and films, provided by HUJI [8], [9]. Finally Elbit and Probayes are respectively in charge of the packaging of the machine and its programming.

[1]The University of Liège, Belgium
[2]The Hebrew University of Jerusalem, Israel
[3]Institute for chemical and physical processes, Italy
[4]The University of Padua, Italy

978-1-7281-5221-9/19/$31.00 ©2019 IEEE

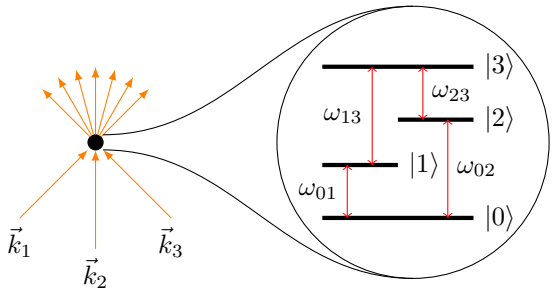


Fig. 1. Scheme of the experiment and the energy levels

The goal of this paper is to provide an example of classical computations speedup due to the intrinsic parallelism and optical addressing of the device we are building. An interesting example of a circuit to implement is a n-bit adder. In classical logic it is challenging due to long delay times, arising from the necessity to evaluate sums of each pair of bits consecutively.

In Sect. II of the present paper we briefly describe the idea behind the device able to evaluate binary n-input functions in parallel (the COPAC device in the following). In Sect. III we give the description of implementation of logical functions with a COPAC device. In Sect. IV we present the realization of a parallel n-bit adder with a COPAC device.

II. Physical description

In the present section we provide a description of the method of the calculation with molecular logical units, developed in [4], [5], [10].

A. Notation

The states $|x\rangle$ are vectors in Hilbert space. Ket vectors $|x\rangle$ are column vectors, bra vectors $\langle x|$ are row vectors. They relate as $\langle x| = (|x\rangle^*)^T := (|x\rangle)^\dagger$ where † operation is Hermitian conjugation. A scalar product of two vectors is written as $\langle x|y\rangle$.

The expectation value of a Hermitian operator O (i.e. $O^\dagger = O$) is denoted as $\langle x|O|x\rangle$. It's a quantity that is measured in an experiment.

B. System description

The physical system under study was described in detail in [4]. The energy structure is presented in Fig. 1, where the eigenstates $|i\rangle, i = 0, 1, 2, 3$, are represented as black horizontal lines, and the allowed transitions with their respective frequencies are depicted as red vertical lines. For simplicity we assume that the transition between $|1\rangle$ and $|2\rangle$ is forbidden.

The ensemble of systems is excited by three laser pulses. Each of the pulses has a different propagation direction $\vec{k}_i, i = 1, 2, 3$. The pulses are assumed to either correspond to a single transition frequency or to be spectrally broad, so that they can equally interact with all the transitions of the system. We also assume that it is possible to have consequent pulses with different spectral properties, which is not implemented in our setup but is generally possible with some adjustments [11]–[13].

Due to the large number of systems in the ensemble, there is no need to repeat the experiment multiple times and to average over the results, as is commonly done in quantum mechanical experiments. The measurement over an ensemble already presents an averaged result [4], [10]. For the same reason of having a large ensemble we can discard errors caused by dephasing and temperature-induced relaxation/excitation: they represent a contribution that is relatively small (around 1%) and constant.

For simplicity we assume that there is no dissipation. The state of the system in this case is represented by a state vector

$$|\phi(t)\rangle = \sum_{j=0}^{3} C_j(t)\,|j\rangle$$

where the $C_j(t)$ coefficients are probability amplitudes, i.e. the probability to find the system in a state $|j\rangle$ at time t is equal to $|C_j(t)|^2$.

The dynamics of the system is described by the Schrödinger equation:

$$\frac{d}{dt}\,|\phi(t)\rangle = -\frac{i}{\hbar}H(t)\,|\phi(t)\rangle \tag{1}$$

where $H(t)$ is the Hamiltonian of the system and $\hbar$ is Planck's constant.

The Hamiltonian $H(t)$ consists of two parts:

$$H(t) = H_0 + V(t) \tag{2}$$

the behavior of the system in absence of the excitation is described by H_0 and the interaction with the light – by $V(t)$. The latter writes $V(t) = \mu E(t)$, where μ is the transition dipole operator and the electric field $E(t)$ is the sum over the three pulses:

$$E(t) = \sum_{j=1}^{3}\left(E_j(t)e^{-i\omega_j t + i\vec{k}_j\vec{r}} + \text{c.c.}\right) \tag{3}$$

The shapes of the pulses are defined by $E_j(t)$. We assume the impulsive limit: the durations of the pulses are much shorter then characteristic times of the system.

To simplify further calculations we use a representation, known as the interaction picture. The operators and the states write respectively:

$$O^I(t) = \exp\left(\frac{i}{\hbar}H_0\,(t-t_0)\right)O(t)\exp\left(-\frac{i}{\hbar}H_0\,(t-t_0)\right) \tag{4}$$

$$|\psi^I(t)\rangle = \exp\left(\frac{i}{\hbar}H_0\,(t-t_0)\right)|\psi(t)\rangle \tag{5}$$

The Schrödinger equation is simplified to

$$\frac{d}{dt}\,|\phi^I(t)\rangle = -\frac{i}{\hbar}V(t)\,|\phi^I(t)\rangle \tag{6}$$

By integration of both right- and left-hand sides, it can be rewritten in a form of an integral equation

$$\left|\phi^I(t)\right\rangle = \left|\phi^I(t_0)\right\rangle - \frac{i}{\hbar}\int_{t_0}^{t} dt_1\, V(t_1)\left|\phi^I(t_1)\right\rangle \qquad (7)$$

To solve it, we follow [14], iteratively substituting $\left|\phi^I(t_1)\right\rangle$ in the right-hand side by the whole right-hand side $\left(\left|\phi^I(t_0)\right\rangle - \frac{i}{\hbar}\int_{t_0}^{t_{j+1}} dt_j\, V(t_j)\right)$. The right-hand side and thereby the solution of (7) becomes a series of nth order perturbations:

$$\left|\phi^I(t)\right\rangle = \sum_{n=0}^{\infty}\left|\phi^{(n)}(t)\right\rangle \qquad (8)$$

$$\left|\phi^{(n)}(t)\right\rangle = \left(-\frac{i}{\hbar}\right)^n \prod_{j=1}^{n}\int_{t_0}^{t_{j+1}} dt_j V(t_j)\left|\phi^I(t_0)\right\rangle \qquad (9)$$

where $t_{n+1} = t$ and $\left|\phi^{(0)}(t)\right\rangle = \left|\phi^I(t_0)\right\rangle$.

Note that the nth order perturbation includes n interactions with the electromagnetic field of the pulses.

C. Third-order polarization

The detector of the experimental setup measures the optical response $P(t)$, defined as

$$P(t) = \langle\phi(t)|\,\mu\,|\phi(t)\rangle = \left\langle\phi^I(t)\right|\mu^I(t)\left|\phi^I(t)\right\rangle \qquad (10)$$

We substitute $\left|\phi^I(t)\right\rangle$ with (8). To distinguish different perturbation orders in the polarization, we can represent it as $P(t) = \sum_n P^{(n)}(t)$, where

$$P^{(n)}(t) = \sum_{m=0}^{n}\left\langle\phi^{(n-m)}(t)\right|\mu^I(t)\left|\phi^{(m)}(t)\right\rangle \qquad (11)$$

The perturbations to the state $\left|\phi^{(m)}(t)\right\rangle$ include m integrals over $V(t)$, and $V(t) \sim \sum_{j=1}^{3}\left(E_j(t)e^{-i\omega_j t + i\vec{k}_j\vec{r}} + \text{c.c.}\right)$. Thus, each of the terms in (11) consists of subterms, that are proportional to $\exp\{i\sum_j l_j\vec{k}_j\vec{r}\}$ where $l_j \in [-m, m]$. These subterms are contributions to the intensity of the emission in corresponding directions.

The first non-zero perturbation with directions of emitted light different from the incident light directions $\{k_1, k_2, k_3\}$, is $P^{(3)}(t)$. We'll consider only this perturbation as the strongest one.

III. Application to Logic

To consider the implementation of binary logic let us explore possible evolution paths of the system. Different evolution paths correspond to different propagation directions of the scattered light. We assume that we can detect only in one direction. It allows us to disregard most of the evolution trajectories, as they correspond to other directions.

Those that we are interested in, are presented in Fig. 2, where the nodes represent the states that contribute to the $P^{(3)}$: the left(right) number corresponds to the excitation of a ket(bra) state. The excitation of a ket(bra) state by ith pulse corresponds of the plus(minus) sign before k_i in the emission direction. The relaxation due to the stimulated emission corresponds to the opposite sign. The direction we detect in is the rephasing direction (R) $-k_1 + k_2 + k_3$.

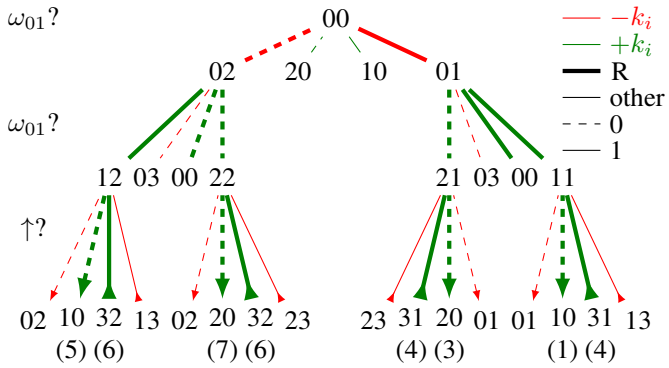


Fig. 2. Diagram of the evolution of the system under the three pulses. The legend is in the upper right corner: green (red) lines correspond to $(-)k_i$, and the lines that correspond to rephasing direction $-k_1 + k_2 + k_3$ are marked as thick. The questions to the left give rise to three binary variables, and the paths that correspond to answer 0(1) are shown with dashed(solid) lines. The numbers in brackets below final states indicate the corresponding positions on the R spectrum Table I.

A. First pulse

The initial state is $|\phi(t_0)\rangle = |0\rangle$. The first excitation corresponds to a term in (11) that includes either $V(t_1)\,|0\rangle$ or $\langle 0|\,V(t_1)$. The operator $V(t_1)$ acting on $|0\rangle$ excites either $|1\rangle$ or $|2\rangle$ (and same for the bra vectors). Thus, there are four possible options for a transition: 01, 10, 02 and 20.

Excitation of 01 and 02 corresponds to $-k_1$, while 10 and 20 contribute $+k_1$ to the total wave-vector. As we are interested only in the direction, that includes $-k_1$, we will not continue the branches of 20 and 10.

Let us define excitation frequency Ω_{exc} as the frequency of the transition induced by the first pulse. If 01 is excited, $\Omega_{exc} = \omega_{01}$, as can be seen from the scheme of the energy levels in Fig. 1, and similarly $\Omega_{exc} = \omega_{02}$ for the excitation of 02 by the first pulse.

B. Second and third pulses

The situation is similar for the second pulse, with two exceptions. First, now we are interested in the transitions corresponding to $+k_2$ (they are shown with thick lines). Second, the system is now not in the ground state, but in a superposition of the excited states $|1\rangle$ and $|2\rangle$, and is evolving under the free Hamiltonian H_0.

The interaction between the pulse and the system depends on the state of the system. Thus by tuning the delay between the pulses, one is able to choose the state in which the system will be at the time of the second pulse, and hence to define the probabilities of different transitions. This is the way to implement a particular logic function.

The trajectories involving 00 state after the second pulse (the ground state bleach, GSB) are not considered in the logic implementation, as their contribution is a constant with respect to the time between the 2nd and the 3rd pulses [5].

TABLE I
POSSIBLE COMBINATIONS OF THE EMISSION FREQUENCY Ω_{em} AND THE EXCITATION FREQUENCY Ω_{exc}.

Ω_{exc} \ Ω_{em}	ω_{01}	ω_{23}	ω_{02}	ω_{13}
ω_{02}	5	6	7	8
ω_{01}	1	2	3	4

The third pulse induces transitions at multiple frequencies. A binary variable corresponding to this transition is linked with a question, whether the interaction with the field induced an absorption of a photon and excitation of a higher state (shown with arrows pointing up), or it was a stimulated emission that caused relaxation of the system (shown with downwards arrows).

The frequency of the final emission is a frequency of the transition between the bra and ket states in the final node (i.e., the system in a node ij after the third pulse will emit at ω_{ij}). These four values of the emission frequency Ω_{em} together with the two values of the excitation frequency Ω_{exc} present 8 possible combinations. Let us enumerate them as shown in Table I.

C. Introducing logical variables

The 8 possible values of the two frequencies give rise to the three logical variables.

The first transition can happen at one of the two frequencies: ω_{01} and ω_{02}. Therefore the value of the first logical variable corresponds to the question "is the frequency Ω_{exc} equal to ω_{01}"?

Transitions induced by the second pulse happen at all the four frequencies of the system, however each of the paths within R contributions presents a choice between ω_{01} and another frequency. Thus the value of the second logical variable corresponds to the same question "was the transition frequency equal to ω_{01}?"

During the third pulse in each of the paths there is either an absorption or an emission of a photon by a system. The value of the third logical variable corresponds to the question "was there an absorption?"

The transitions corresponding to logical 0 (1) are shown in the diagram Fig. 2 with dashed (solid) lines.

If the three bits corresponding to "ω_{01}?", "ω_{01}?" and "$\uparrow$?" are labeled and ordered as abc, they can be represented with a decimal value $x = 2^0c+2^1b+2^2a$. The logical values (columns 2 to 4), the decimal value x corresponding to them (column 1) and the corresponding points on the spectrum Table I for R direction (5th column) are presented in the Table II. Here we also consider the non-rephasing (NR) direction $k_1 - k_2 + k_3$ (6th column).

A concatenation of Tables I and II give the Table III that shows the correspondence between the decimal value of the input and the excitation and emission frequencies.

The observable $P^{(3)}(t)$, measured at R direction, includes the contributions of different trajectories. The spectrum obtained from a single experiment allows to obtain an

TABLE II
THE LOGICAL VALUES AND THE POINTS ON THE SPECTRUM (TABLE I) OF REPHASING CONTRIBUTIONS R AND NON-REPHASING CONTRIBUTIONS NR.

x	ω_{01}?	ω_{01}?	$\uparrow$?	R spectrum point	NR spectrum point
0	0	0	0	7	7
1	0	0	1	6	6
2	0	1	0	5	7
3	0	1	1	6	8
4	1	0	0	3	2
5	1	0	1	4	2
6	1	1	0	1	1
7	1	1	1	4	4

TABLE III
CORRESPONDENCE BETWEEN THE COMBINATIONS OF THE EMISSION FREQUENCY Ω_{em} AND THE EXCITATION FREQUENCY Ω_{exc} ON ONE HAND, AND THE INPUTS x OF THE LOGICAL FUNCTIONS.

Ω_{exc} \ Ω_{em}	ω_{01}	ω_{23}	ω_{02}	ω_{13}
ω_{02}	2	1, 3	0	-
ω_{01}	6	-	4	5, 7

evaluation of a logical function for multiple values of its inputs simultaneously. By tuning the time between the second and the third pulses it is possible to choose, what paths will be possible, and thus to choose values of 1 or 0 for different points in the spectrum, thereby encoding different functions.

Note that not all the functions are possible to encode with the detection in R direction, but only those satisfying $f(100) = f(110)$ and $f(101) = f(111)$. There are two possibilities to realize functions not satisfying these conditions. The first one is to consider several directions: the last column of the Table II shows the relation between the logical variables and the spectrum for the NR direction: the values of the inputs corresponding to the points on the spectra are different. Another way to evaluate more functions is to use the permutations of variables, or to permute the variables with their complements. In the following we restrict ourselves to the latter option.

The value of a function $f(x)$ is equal to 1 (0) in the case when the detected intensity for the pair of frequencies $\Omega_{exc}, \Omega_{em}$ is higher (lower) then some threshold.

IV. ADDER IMPLEMENTATION

A. Logic operations

This section describes operation of addition of (sets of) two numbers a and b with the COPAC device. Their sum is denoted as s:

$$s = a + b \tag{12}$$

Any number a can be represented in binary form as

$$a = a_n a_{n-1} ... a_1 a_0.$$

For example, $3_{dec} = 011_{bin}$. In the following we shall omit the subscripts dec and bin where the representation is clear from the context. Note that if a and b are n-bit numbers, their sum s contains $n + 1$ bits.

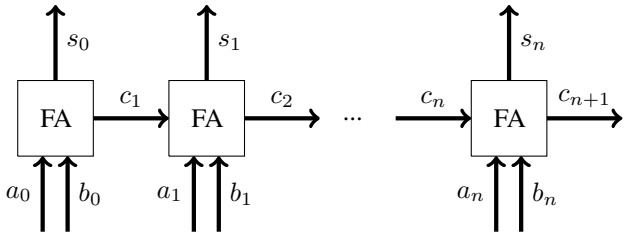


Fig. 3. n-bit adder

TABLE IV
THE TRUTHTABLE FOR THE FA. THE DECIMAL VALUES OF THE INPUT $a_ib_ic_i$ AND OUTPUT $c_{i+1}s_i$ ARE GIVEN BY x AND y RESPECTIVELY.

x	a_i	b_i	c_i	c_{i+1}	s_i	y
0	0	0	0	0	0	0
1	0	0	1	0	1	1
2	0	1	0	0	1	1
3	0	1	1	1	0	2
4	1	0	0	0	1	1
5	1	0	1	1	0	2
6	1	1	0	1	0	2
7	1	1	1	1	1	3

The logical operations used in the following are: OR ($a_i + b_i$), AND (a_ib_i), EXOR ($a_i \oplus b_i$). They are commutative and associative, AND is distributive over OR and EXOR.

B. Logical n-bit adder

An n-bit adder is a logical circuit that takes two n-bit numbers a and b as inputs and returns $(n+1)$-bit number that represents their sum s. Alternatively: it takes $2n$ single bits as inputs and maps them on $(n+1)$ single-bit output. It can be seen as a sequence of full adders (FA), as in Fig. 3.

FA takes tree bits as inputs: two from the addends (a_i and b_i) and one as a carry out c_i from the addition of lower order bits, and returns two outputs: sum s_i and carry out c_{i+1}, that are calculated according to

$$s_i = a_i \oplus b_i \oplus c_i \tag{13}$$
$$c_{i+1} = a_ib_i + (a_i \oplus b_i)\, c_i \tag{14}$$
$$c_0 = 0 \tag{15}$$
$$s_{n+1} = c_{n+1} \tag{16}$$

Eq. (15) represents the initial condition arising from the fact that there is no carry to take into the first adder. Eq. (16) shows that the $(n+1)$th bit of the result s of the addition of two n-bit numbers a and b is a carry out of the last FA.

The FA has two outputs, s_i and c_{i+1}, that correspond to different logic functions (13) and (14) respectively. We assume that they can be realized by tuning the time between the second and the third pulses to the values of T_s and T_c respectively.

The truth table for the FA is shown as Table IV, representing (13) and (14). The x column presents the decimal value of binary number $x = (a_ib_ic_i)_{dec}$, and the last column gives the decimal value of the binary result of FA calculation $y = (c_{i+1}s_i)_{dec}$.

C. Single bit addition

Building on these premises, let us first present the set of operations required to implement a single FA, that is equivalent to an addition of three single bits. It is done by the following algorithm

1) Find the pair of frequencies $\Omega_{exc}, \Omega_{em}$ corresponding to $a_ib_ic_i$
2) Calculate s_i:
   a) Excite the system with a first pulse at frequency Ω_{exc}
   b) Excite the system with a spectrally broad second pulse
   c) Wait for time T_s between the 2^{nd} and the 3^{rd} pulses
   d) Excite the system with a spectrally broad third pulse
   e) Collect the output at Ω_{em}
   f) Save it as s_i
3) Calculate c_{i+1}:
   a) Repeat 2a.-2b.
   b) Wait for time T_c between the 2^{nd} and the 3^{rd} pulses
   c) Repeat 2d.-2e.
   d) Save the output as c_{i+1}

Let us explore several examples. First, let's assume there is no carry out of the previous addition $(c_i = 0)$.

The input of the simplest addition 0+0 ($a_i = 0$, $b_i = 0, x = 0$), according to the Table III, corresponds to the excitation at $\Omega_{exc} = \omega_{02}$ and collection at $\Omega_{em} = \omega_{02}$. As, the value of both functions s_i and c_{i+1} for $x = 0$ is zero, the detected signal will be below the threshold for both waiting times T_s and T_c, thus implementing $s_i(0) = 0, c_{i+1}(0) = 0$.

The addition 1+0 ($a_i = 1$, $b_i = 0, x = 4$) corresponds to the excitation at $\Omega_{exc} = \omega_{01}$ and collection at $\Omega_{em} = \omega_{02}$. When the function s_i (c_{i+1}) is implemented, the detected signal will be above (below) the threshold. Together the two output bits $c_{i+1}s_i = 01_{bin} = 1_{dec}$.

The addition 1+1 ($a_i = 1$, $b_i = 1, x = 6$) corresponds to the excitation at $\Omega_{exc} = \omega_{01}$ and collection at $\Omega_{em} = \omega_{01}$. When the function s_i (c_{i+1}) is implemented, the detected signal will be below (above) the threshold. Together the two output bits $c_{i+1}s_i = 10_{bin} = 2_{dec}$.

If $c_i = 1$, addition 1+1 ($a_i = 1$, $b_i = 1$) now corresponds to x=7 and to the excitation at $\Omega_{exc} = \omega_{01}$ and collection at $\Omega_{em} = \omega_{13}$. For both outputs the detected intensity will be above threshold, thus $a_ib_ic_i = 111_{bin} \Rightarrow c_{i+1}s_i = 11_{bin} = 3_{dec}$.

However, the value $a_ib_ic_i = 101_{bin} = 5_{dec}$ corresponds to the same set of frequencies as $x = 7$. As $c_{i+1}(5) = c_{i+1}(7)$, this is not a problem for the carry out calculation. But for the implementation of the function s_i this will pose a challenge: $s_i(5) = 0$, while $s_i(7) = 1 > s_i(5)$, thus the signal from $x = 7$ makes impossible to measure the absence of signal that should occur for $x = 5$. This can be solved by a permutations of the bits of the input, when such a situation occurs, for the

bits where the functions have the same values. The problem arises for $c_{i+1}(1), s_i(3), s_i(5)$, thus the possible permutations to be made are: $001 \to 010$, $011 \to 110$, $101 \to 110$.

D. Parallel single bit addition

To have several parallel calculations, we can exploit quantum superposition. This can be done in twofold way:

- by detection at several frequencies
- by excitation at several frequencies

However, in order to be able to distinguish between outputs from different inputs, only such values can be evaluated at parallel, that correspond to different emission frequencies.

The algorithm of evaluation in this case should be modified:

1) Find the pairs of frequencies $(\Omega_{exc}, \Omega^j_{em}), j \in [1, k], k \leq 3$ corresponding to $(a_i b_i c_i)^j$
2) Calculate s_i:
   a) Excite the system with a first pulse at frequency Ω_{exc}[5]
   b) Excite the system with a spectrally broad second pulse
   c) Wait for time T_s between the 2^{nd} and the 3^{rd} pulses
   d) Excite the system with a spectrally broad third pulse
   e) Collect the output at $\Omega^1_{em}, \Omega^2_{em}, ..., \Omega^k_{em}$
   f) Save it as $s^1_i, s^2_i, ..., s^k_i$
3) Calculate c_{i+1}:
   a) Repeat 2a.-2b.
   b) Wait for time T_c between the 2^{nd} and the 3^{rd} pulses
   c) Repeat 2d.-2e.
   d) Save the output as $c^1_{i+1}, c^2_{i+1}, ..., c^k_{i+1}$

For example, one can compute 1+0+0, 1+1+0 and 1+1+1 at the same time. The excitation frequency is ω_{02}, and the results of detection at ω_{02}, ω_{01}, ω_{23} are written in $s_i(100), s_i(110), s_i(111)$, and the same for c_{i+1}.

Thus, a parallel computation of up to three sums of three two-bit variables can be performed by two calls to the COPAC device. When more than three sums need to be computed, they must be divided into groups of sums that are possible to evaluate at a single call.

The time duration of a call is limited by following factors:

- Wigner time delay (time to absorb a photon) that depends on the system under study, and waiting times $T_{s,c}$: in our setup they are of the order of $10^{-15} - 10^{-12}$ s.
- Photodetectors time resolution. For the modern detectors they can reach as low as 10^{-12}s, but typical values are about 10^{-9}s.
- Laser repetition rates, that can reach GHz, with typical values about MHz.
- Time to move the delay lines between setups with T_s and T_c.

Thus, potentially, calculations with the COPAC machine can outperform the state-of-art technologies.

[5]One can also excite at both excitation frequencies ω_{02} and ω_{01} (or with a broad pulse), but with prudence, avoiding the situations when the signal for different Ω_{exc} and same Ω_{em} overshadows the absence of signal that we are looking for.

E. Parallel n-bit addition

The optimal way to implement a n-bit adder would be to compute first all the c_{i+1} values and store them, and then reuse them to compute the s_i. A new computed value of s_i would be stored in place of c_i. The computation of multiple s_i at one call is optimal because it allows broader use of parallel computation.

As an example, let us consider three sums: 6+6, 2+1, 2+5. In binary: 110+110, 010+001, 010+101. One can notice, that the first two bits of the two last sums coincide, which allows to reduce number of calculations.

First we consequently evaluate c_i according to the algorithm above. For the first bit $\Omega_{exc} = \omega_{02}$, $[\Omega^1_{em}, \Omega^2_{em}, \Omega^3_{em}] = [\omega_{02}, \omega_{01}, \omega_{01}]$, and the results are $[c^1_1, c^2_1, c^3_1] = [0, 0, 0]$. For the second bit $\Omega_{exc} = \omega_{01}$, $[\Omega^1_{em}, \Omega^2_{em}, \Omega^3_{em}] = [\omega_{01}, \omega_{02}, \omega_{02}]$, and the results are $[c^1_2, c^2_2, c^3_2] = [1, 0, 0]$. For the third bit we need two calls: $\Omega_{exc} = \omega_{01}$, $\Omega^1_{em} = \omega_{13}$, $c^1_3 = 1$ and $\Omega_{exc} = \omega_{02}$, $[\Omega^2_{em}, \Omega^3_{em}] = [\omega_{02}, \omega_{01}]$, $[c^2_3, c^3_3] = [0, 0]$.

Then we have 9 entries to evaluate s_i on. First bit: 000, 010, 010; second bit: 110, 100, 100; third bit: 111, 000, 010. They can be evaluated in two calls:

1) $\Omega_{exc} = \omega_{02}$, $[\Omega^1_{em}, \Omega^2_{em}] = [\omega_{02}, \omega_{01}]$, and the results are $s^1_0, s^2_2, s^2_0 = 0$ (evaluated at ω_{02}), $s^3_0, s^3_2 = 1$ (evaluated at ω_{01}).
2) $\Omega_{exc} = \omega_{01}$, $[\Omega^1_{em}, \Omega^2_{em}, \Omega^3_{em}] = [\omega_{01}, \omega_{02}, \omega_{13}]$, and the results are $s^1_1 = 0$,$s^2_1, s^3_1 = 1$, $s^1_2 = 1$ at the respective frequencies.

Thus, if the delay of a single gate is D, the total delay in this example is $6D$ for three 3-but additions, that in a classical ripple-carry adder would take $21D'$, where the single gate time may differ.

F. Scaling

In the general case, for evaluation of an arbitrary function f one needs to know the corresponding time between T_f between the pulses. A compiler was written for this device, whose task is mainly to provide T_f and the set of freuqencies $(\Omega_{exc}, \Omega^j_{em})$ that correspond to the function to evaluate and its inputs respectively, according to the rules, described above.

The result of a calculation is stored in memory. To be reused in another calculation, this result must be passed to the compiler to be expressed as a set of frequencies.

V. Summary

In this paper we presented the principles behind a device developed under COPAC project and able to evaluate a logical function for multiple values of inputs in parallel. The algorithm to implement a n-bit adder with a COPAC machine was suggested, showing that the machine provides the speedup due to the possibility to compute several additions in parallel.

In the paper we focused on a quantum system with 4 eigenstates, connected as in Fig. 1. However, different configurations can be engineered, providing more input variables and more possibilities for parallel computation.

ACKNOWLEDGMENTS

The authors would like to thank Raphael Levine and Dimitar Dimitrov for helpful discussions.

REFERENCES

[1] Susan Stepney, Steen Rasmussen, and Martyn Amos. *Computational Matter*. Springer, 2018.

[2] F Remacle and RD Levine. Towards a molecular logic machine. *The Journal of Chemical Physics*, 114(23):10239–10246, 2001.

[3] Françoise Remacle, Rainer Weinkauf, and Raphael D Levine. Molecule-based photonically switched half and full adder. *The Journal of Physical Chemistry A*, 110(1):177–184, 2006.

[4] Barbara Fresch, Dawit Hiluf, Elisabetta Collini, RD Levine, and Françoise Remacle. Molecular decision trees realized by ultrafast electronic spectroscopy. *Proceedings of the National Academy of Sciences*, 110(43):17183–17188, 2013.

[5] Barbara Fresch, Marco Cipolloni, Tian-Min Yan, Elisabetta Collini, RD Levine, and Françoise Remacle. Parallel and multivalued logic by the two-dimensional photon-echo response of a rhodamine–dna complex. *The journal of physical chemistry letters*, 6(9):1714–1718, 2015.

[6] Ron Orbach, Fuan Wang, Oleg Lioubashevski, RD Levine, Francoise Remacle, and Itamar Willner. A full-adder based on reconfigurable dna-hairpin inputs and dnazyme computing modules. *Chemical Science*, 5(9):3381–3387, 2014.

[7] Elisabetta Fanizza, Carmine Urso, Vita Pinto, Antonio Cardone, Roberta Ragni, Nicoletta Depalo, M Lucia Curri, Angela Agostiano, Gianluca M Farinola, and Marinella Striccoli. Single white light emitting hybrid nanoarchitectures based on functionalized quantum dots. *Journal of Materials Chemistry C*, 2(27):5286–5291, 2014.

[8] Oren Ben Dor, Noam Morali, Shira Yochelis, Lech Tomasz Baczewski, and Yossi Paltiel. Local light-induced magnetization using nanodots and chiral molecules. *Nano letters*, 14(11):6042–6049, 2014.

[9] Eyal Cohen, Michael Gruber, Elisabet Romero, Shira Yochelis, Rienk van Grondelle, and Yossi Paltiel. Properties of self-assembled hybrid organic molecule/quantum dot multilayered structures. *The Journal of Physical Chemistry C*, 118(44):25725–25730, 2014.

[10] Tian-Min Yan, Barbara Fresch, RD Levine, and Francoise Remacle. Information processing in parallel through directionally resolved molecular polarization components in coherent multidimensional spectroscopy. *The Journal of chemical physics*, 143(6):064106, 2015.

[11] Thomas AA Oliver, Nicholas HC Lewis, and Graham R Fleming. Correlating the motion of electrons and nuclei with two-dimensional electronic–vibrational spectroscopy. *Proceedings of the National Academy of Sciences*, 111(28):10061–10066, 2014.

[12] Patrick F Tekavec, Jeffrey A Myers, Kristin LM Lewis, and Jennifer P Ogilvie. Two-dimensional electronic spectroscopy with a continuum probe. *Optics letters*, 34(9):1390–1392, 2009.

[13] Nicholas M Kearns, Randy D Mehlenbacher, Andrew C Jones, and Martin T Zanni. Broadband 2d electronic spectrometer using white light and pulse shaping: noise and signal evaluation at 1 and 100 khz. *Optics express*, 25(7):7869–7883, 2017.

[14] Shaul Mukamel. *Principles of nonlinear optical spectroscopy*, volume 29. Oxford university press New York, 1995.

Author Index

P

Q

R

S

T

V

W

Y