APPLICATIONS OF EMERGING MEMORY IN MODERN COMPUTER SYSTEMS: STORAGE AND ACCELERATION

by

Xiaoxiao Liu

Bachelor of Electrical Engineering, Beihang University, 2005

Master of Software Engineering, Beihang University, 2009

Master of Electrical Engineering, Florida International University, 2009

Submitted to the Graduate Faculty of

Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2017

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF ENGINEERING

This dissertation was presented

by

Xiaoxiao Liu

It was defended on

May 31, 2017

and approved by

Yiran Chen, Ph.D., Adjunct Associate Professor, Department of Electrical and Computer Engineering

Hai Li, Ph.D., Adjunct Associate Professor, Department of Electrical and Computer Engineering

Rami Melhem, Ph.D., Professor, Department of Computer Science

William Stanchina, Ph.D., Professor, Department of Electrical and Computer Engineering

Murat Akcakaya, Ph.D., Assistant Professor, Department of Electrical and Computer Engineering

Dissertation Director:

Yiran Chen, Ph.D., Associate Professor, Departmental of Electrical and Computer Engineering

APPLICATIONS OF EMERGING MEMORY IN MODERN COMPUTER SYSTEMS: STORAGE AND ACCELERATION

Xiaoxiao Liu, PhD

University of Pittsburgh, 2017

In recent year, heterogeneous architecture emerges as a promising technology to conquer the constraints in homogeneous multi-core architecture, such as supply voltage scaling, off-chip communication bandwidth, and application parallelism. Various forms of accelerators, e.g., GPU and ASIC, have been extensively studied for their tradeoffs between computation efficiency and adaptively. But with the increasing demand of the capacity and the technology scaling, accelerators also face limitations on cost-efficiency due to the use of traditional memory technologies and architecture design.

Emerging memory has become a promising memory technology to inspire some new designs by replacing traditional memory technologies in modern computer system. In this dissertation, I will first summarize my research on the application of Spin-transfer torque random access memory (STT-RAM) in GPU memory hierarchy, which offers simple cell structure and non-volatility to enable much smaller cell area than SRAM and almost zero standby power. Then I will introduce my research about memristor implementation as the computation component in the neuromorphic computing accelerator, which has the similarity between the programmable

resistance state of memristors and the variable synaptic strengths of biological synapses to simplify the realization of neural network model. At last, a dedicated interconnection network design for multicore neuromorphic computing system will be presented to reduce the prominent average latency and power consumption brought by NoC in a large size neuromorphic computing system.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

PREFACE

This dissertation is submitted in partial fulfillment of the requirements for Xiaoxiao Liu's degree of Doctor of Philosophy in Electrical and Computer Engineering. It contains the work done from January 2013 to May 2017. My advisor is Yiran Chen, University of Pittsburgh, 2013 – present.

The work is to the best of my knowledge original, except where acknowledgement and reference are made to the previous work. There is no similar dissertation that has been submitted for any other degree at any other university.

Part of the work has been published in the conference:

1. DAC2015: **Xiaoxiao Liu**, Mengjie Mao, Beiye Liu, Hai Li, Yiran Chen, Boxun Li, Yu Wang, Hao Jiang, Mark Barnell, Qing Wu, Jianhua Yang, "RENO: A high-efficient reconfigurable neuromorphic computing accelerator design," Design Automation Conference (DAC), pp. 1-6, 2015.
2. HPEC2014: **Xiaoxiao Liu**, Mengjie Mao, Hai Li, Yiran Chen, Hao Jiang, J.J. Yang, Qing Wu, M. BarnellX. Li, T. Huang, Q. Wu, M. Barnell, H. Li, Y. Chen, "A heterogeneous computing system with memristor-based neuromorphic accelerators," High Performance Extreme Computing Conference (HPEC), pp. 1-6, 2014.

3. ASPDAC2015: **Xiaoxiao Liu**, Mengjie Mao, Xiuyuan Bi, Hai Li, Yiran Chen, "An efficient STT-RAM-based register file in GPU architectures," Asia and South Pacific Design Automation Conference (ASPDAC), pp. 490-495, 2015.
4. ASPDAC2014: **Xiaoxiao Liu**, Yong Li, Yaojun Zhang, Alex K Jones, Yiran Chen, "STD-TLB: A STT-RAM-based dynamically-configurable translation lookaside buffer for GPU architectures," Asia and South Pacific Design Automation Conference (ASPDAC), pp. 355-360, 2014.
5. ICCAD2017: **Xiaoxiao Liu**, Wei Wen, Hai Li, Yiran Chen, "Neu-NoC: A High-efficient Interconnection Network for Accelerated Neuromorphic Systems," International Conference on Computer-Aided Design (ICCAD), 2017, Submitted.

Part of the work has been published in journal publications:

1. **Xiaoxiao Liu,** Mengjie Mao, Beiye Liu, Bosun Li, Yu Wang, Hao Jiang, Mark Barnell, Qing Wu, Jianhua Yang, Hai Li, Yiran Chen, "Harmonica: A Framework of Heterogeneous Computing Systems With Memristor-Based Neuromorphic Computing Accelerators," IEEE Transactions on Circuits and Systems I, vol. 63, Issue 5, pp. 617-628, 2016.

ACKNOWLEDGEMENTS

I would like to acknowledge the support of my advisors, Dr. Yiran Chen and Dr. Hai Li, who made this work possible. Dr. Chen led me into the world of research, and Dr. Li has always been a role model to me as a female scientist. During the past four years, they gave me numerous insightful guidance, and their supports helped me overcome all the difficulties during my PhD study. I also would like to thank the committee members, Dr. Rami Melhem, Dr. William E Stanchina, and Dr. Murat Akcakaya, for their help ameliorating for this dissertation.

Besides, I'd like to express my gratitude to the members from Evolutional Intelligent (EI) lab at Swanson School of Engineering for their consistent supports during my research. Finally, I'd like to thank my family for their great encouragement. I dedicate this PhD thesis to my husband, Ling Zhang, who always believes me and stands by me, and my son, Mo, who bring the greatest joy to my life every day.

1.0 INTRODUCTION

Homogeneous multi-core architecture was proposed in microprocessor designs to fully utilize silicon area and overcome the obstacles of frequency up-scaling caused by the rapidly increased wire delay and circuit power consumption [1][2]. However, the constraints on supply voltage scaling, off-chip communication bandwidth, and application parallelism severely hinder the pace of increasing the number of CPU cores on a single chip, and consequently, limit the overall computational capacity [3]. In recent years, heterogeneous architecture emerges as a promising technology to conquer the above challenges [4]. Various forms of off-chip accelerators, e.g., ASIC, FPGA, and GPU, have been extensively studied [5][6][7] for their tradeoffs between computation efficiency and adaptivity.

Graphic processing unit (GPU) has become the most widely utilized off-chip accelerator in general purpose computing acceleration for high throughput and power efficiency. Thousands of parallel threads are processed simultaneously so that the long access latency of off-chip memory can be effectively hidden. The extremely high parallelism requires large on-chip memories in GPU, like translation look-aside buffer (TLB) which has more than 20x capacity of TLB in CPU to retain enough physical page addresses on chip to handle a large volume of parallel processing threads[20][21], and register file (RF) with a few megabytes capacity to hold the data for thousands active threads used to maintain the extremely high parallelism in GPU[11]. On-chip memories are commonly implemented with SRAM cells to satisfy the

capacity and performance needs. However, the capacity and power consumption of GPU on-chip memories are often constrained by the large cell area and high leakage current of SRAM and has become one of the major bottlenecks of GPU performance and energy efficiency [10].

Besides off-chip accelerators, many practices [12][13][14][15] were also conducted to integrate general-purpose CPU cores with processing elements that are designed to accelerate the execution of some special codes (called target codes), e.g., the codes producing approximated results. Many target codes of approximate computing (e.g., approximated calculation, rendering methodology and statistical representation) have been identified in a large variety of applications such as pattern recognition, computer vision, data mining, signal processing etc. [16][50]. Artificial neural network (ANN) can be also considered as one kind of approximate computing with high adaptivity to many high-performance applications [50]. The inherent resilience to soft and hard errors in computation makes ANN a promising solution to conquer the aggravated system reliability issue under the highly-scaled technology nodes [52]. Software-based ANN realizations, however, are often associated with extremely high hardware cost required by emulating the complex connection in the neural network. Tradition CMOS-based implementations of ANN also suffer from the inefficiency in power/area due to the large cell area, high leakage current and volatility [16][68].

Emerging memory technologies have been studied as the promising next generation nanoscale technologies to conquer the scaling issue and high leakage current in traditional technologies, like CMOS. They usually do not rely on charging and discharging their electronic storage devices to store data and, hence are not impacted by the storage device size shrinks and loss of power. Spin-transfer torque random access memory (STT-RAM) is a popular new memory technology [25][26] featuring high integration density, nanosecond read access time,

and low leakage power consumption. However, the well-known STT-RAM drawbacks of long write latency and high write energy impose constraints on STT-RAM's application in last-level cache and main memory of CPUs [27][28][29] and GPUs [30]. In this work, we demonstrate that TLB in GPUs is an ideal application of STT-RAM due to relatively large memory footprints of GPU applications and infrequent updates of the stored contents. Hence, the large volume of address translation accesses could greatly benefit from the increased TLB capacity by using STTRAM and achieve an improved hit rate. Additionally, the low write pressure typically observed by the TLB naturally mitigates the negative impact of STT-RAM in write performance and energy. Furthermore, by leveraging differential sensing technique [18], we are able to improve the read performance of STT-RAM-based TLB whenever read access latency becomes critical. A novel TLB architecture – STD-TLB: STT-RAM-based dynamically-configurable translation lookaside buffer, is proposed to dynamically balance the access performance and capacity of the TLB upon run-time TLB access requirements.

Moreover, the recent invention of multilevel cell (MLC) design [20] doubles the storage density of STT-RAM. The use of MLC STT-RAM (MLC-STT) in last level cache of CPUs has been widely investigated for energy efficiency enhancement [21][22]. We extends the application of MLC-STT to RF in GPUs in this work. Beyond the near zero leakage power, the potential of implementing larger capacity will effectively relax the limitation on active thread number in register-hungry GPGPU applications and hence, improve system performance. However, the hard and the soft bits in a MLC-STT cell demonstrate very different access time requirements. Thus, we propose a remapping strategy to relocate data based on its access frequency and the register usage requirement of the application. Particularly, the frequently-accessed data is always mapped to the fast rows built with the soft bits while the slow rows

composed of the hard bits are used only when a larger capacity is needed. A runtime warp rescheduling scheme is also developed to reorder the issuing of the ready warps to minimize the access stall induced by the long write operations of MLC-STT.

Besides the off-chip accelerators, the rediscovery of memristor [63] motivates an exciting approach of implementing neuromorphic systems, which denotes the VLSI realization of ANN computation. Compared to the design of traditional CMOS-based digital and analog neuromorphic accelerators [16][68], the similarity between the programmable resistance state of memristors and the variable synaptic strengths of biological synapses dramatically simplifies the structure of neural network circuits [63]. In this work, we propose RENO, a novel highly-efficient reconfigurable neuromorphic computing accelerator with on-chip memristor-based crossbar (MBC) arrays as perceptron network, aiming at the acceleration of ANNs computations. Unlike many neuromorphic systems that perform the computations on pure digital ALUs or analog approximate computing units with AD/DA interface, our design adopts a hybrid method in data representation: the computation within the MBC arrays and the signal communications among the MBC arrays are conducted in analog form, while the control information remains as digital signals. To suppress the accuracy degradation incurred by the memristor resistance shifting during execution, an inline calibration scheme is also developed with negligible performance overhead.

Even with the efficiency coming from the emerging memory applications, in a large scale neuromorphic acceleration system, interconnection network on chip (NoC) has become to consume a major part of energy consumption and delay. Because of the data-intensity and direction-intensity of neuromorphic data traffic between neurons in different layers, the traditional designs of NoC in general purpose multi-core system do not work well in

neuromorphic acceleration system. For instance, the data communication may count for more than 30% computation cost in a deep learning accelerator and grows up rapidly with the increase of the system scale [89]. Meanwhile, about 26% chip area and more than 10% total energy consumption comes from the NoC [89]. The non-uniformity of the transmission data size over the connections, the redundancy of the data transmission between the neural network layers, and the local congestions all significantly degrade the efficacy of the NoC in a neuromorphic acceleration system. Ring topology has been recently used in the NOC design of some multicore systems for its simplicity, i.e., Intel Nehalem [90]. But its implementation in a large system is doubtful by taking into account the increased hop count and long wire delay. In this work, we propose a hybrid ring-mesh NoC architecture (namely, Neu-NoC) that can adapt to the unique communication patterns in neuromorphic acceleration systems to achieve better performance, less energy, and smaller area.

2.0 EMERGING MEMORY APPLICATION FOR STORAGE

2.1 PRELIMINARY

2.1.1 STT-RAM basic

Data is stored in an STT-RAM cell as the resistance of a magnetic tunneling junction (MTJ) device. A MTJ consists of two ferromagnetic layers which sandwich a MgO layer. The magnetization direction of one ferromagnetic layer (fixed layer) is pinned while that of the other ferromagnetic layer (free layer) can be flipped by passing a spin-polarized current. When the magnetization direction of the two layers are anti-parallel, the MTJ resistance is at high state (R_{high}); otherwise, the MTJ resistance is at low state (R_{low}), as shown in Fig. 1(a)[17], respectively. Figure 1(a) depicts the popular "1T1J" STT-RAM cell structure where the MTJ switching current is supplied by a connected NMOS transistor. The STT-RAM cell area is mainly determined by the NMOS transistor size.

2.1.2 STT-RAM with differential sensing

A differential sensing STT-RAM architecture is recently proposed [18] to offer better read performance, same write latency and lower error rate than the 1T1J cell. Differential sensing technique can improve the readability of STT-RAM cells by trading off the memory capacity as

follows: In addition to writing the data to one STT-RAM cell, the inversion of the data is written into an adjacent cell. Rather than comparing the cell states to a reference, the stored data is read out by comparing the resistance of these two complimentary cells. The sense margin is increased from $\frac{1}{2}(R_{high} - R_{low})$ to$(R_{high} - R_{low})$, considerably reducing the corresponding read latency with the improved read reliability. We refer to the cell structure of differential sensing as "2T2J", which is indeed composed of two 1T1J cells. Compared to a 1T1J cell, the dynamic read energy of a 2T2J cell remains constant, while the write dynamic energy of a 2T2J cell essentially doubles as writing the data and its complement into neighboring cells.

2.1.3 Multi-level cell STT-RAM

To further enhance the density, MLC-STT technology was proposed by virtually integrating two MTJs in a cell [19] [20]. Figure 1(a) illustrates a popular serial structure which is composed of a small MTJ and a large MTJ. The combinations of the resistance states of the two MTJs construct four different resistance levels, representing a 2-bit data. The bits determined by the resistance states of the large and small MTJs are normally denoted as "hard bit" and "soft bit", respectively.

As the two MTJs are connected in serial and driven by the same NMOS access transistor, the same current passes through both MTJs during read and write operations. When applying a large switching current to program the large MTJ, the small MTJ encounters an even larger switching current density and hence will turn to the same resistance state as the large MTJ. Therefore, an additional programming step with a small write current is required to program the soft-bit to the target value, which only flips the resistance state of the small MTJ. Such a 2-step write operation of MLC-STT cells is depicted in Figure 1(b). Similarly, a 2-step read operation is

required to detect the soft and the hard bits in sequence, as shown in Figure 1(c). Notably, programming or detecting only the soft bit can be completed in one step.

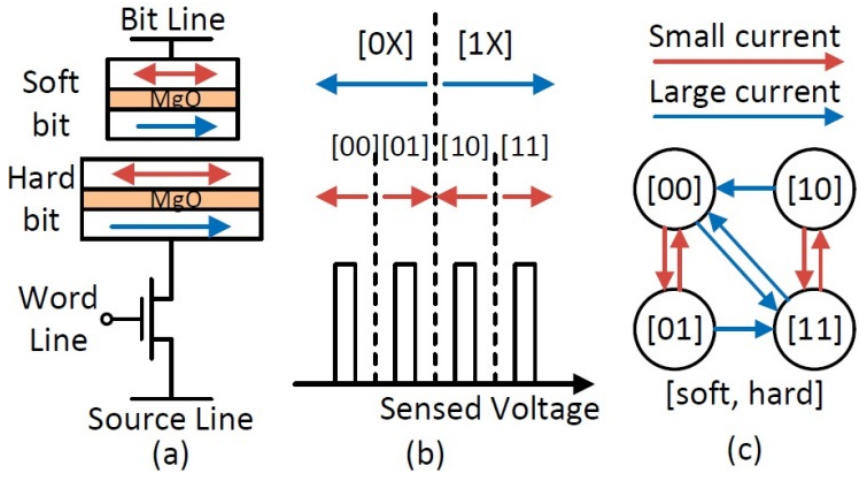

Figure 1. STT-RAM cell structure and operation diagram. (a) multi-level cell (MLC) using serial stacking structure. (b,c) 2-step read and write operation of MLC-STT.

When implementing memory with MLC-STT cells, the hard and soft bits can be separately grouped into different data sets [21]. For instance, a row of MLC-STT cells in a physical array can be regarded as two logic rows that respectively consist of the hard bits and soft bits of these MLC-STT cells. As such, reading or writing the soft-bit row requires only one step. Accessing a hard-bit row, in contrast, need a two-step operation, resulting in much longer access latency and higher energy consumption. Such performance difference between hard-bit and soft-bit rows motivates us to propose a remapping strategy in the designs of MLC-STT-base memory hierarchy.

2.2 STD-TLB: A STT-RAM-BASED DYNAMICALLY-CONFIGURABLE TRANSLATION LOOKASIDE BUFFER FOR GPU ARCHITECTURES

2.2.1 Background and motivation

In 2009, Nvidia announced the first GPU to support unified address space – *Fermi* [23]. Later in 2010, AMD also announced *Graphic Core Next (GCN)* architecture with a virtual memory system to offer x86 addressing with unified address space for both CPU and GPU [24]. In these two architectures, GPU memories are mapped into a continuous 64-bit address space. General instructions can access local scratchpad memory, global memory and system memory addresses defined within the unified address space.

Figure 2 illustrates the overview of the virtual memory system in GPU architectures. The translation from virtual address to physical address is conducted by the memory management unit (MMU) through a multi-level TLB hierarchy. Upon an L1 TLB miss, the corresponding page table entry (PTE) will be loaded from the L2 TLB, which is usually significantly larger than the L1 TLB and stores more cached PTEs.

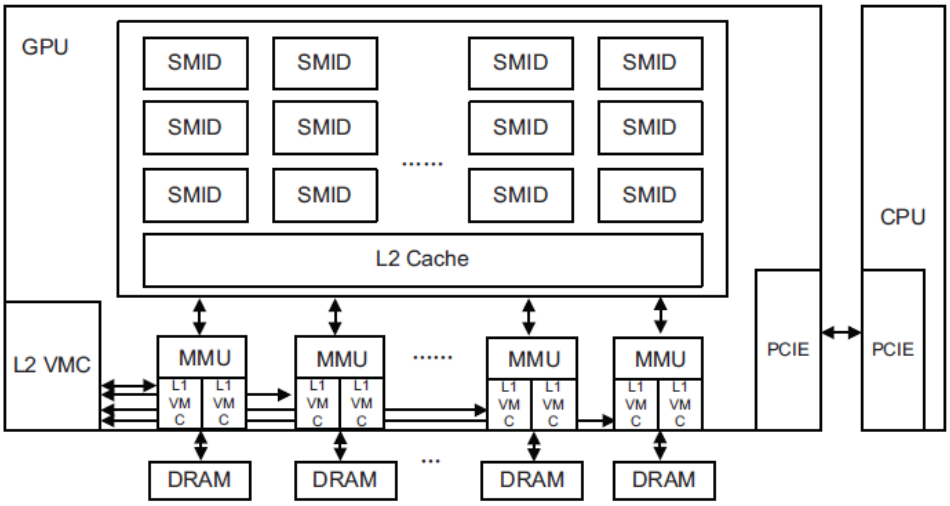

Figure 2. Virtual memory with multi-level TLBs in GPUs.

One important observation that motivates our work is that TLB entries are not modified as frequently as cache blocks. This difference is caused by the function of TLBs: A TLB primarily caches virtual to physical address mappings, which change only upon extremely infrequent OS events, e.g., page re-mapping, page swaps, context switches, and privilege changes. From an application's perspective, the TLB entry is updated only when an old mapping needs to be evicted and a new entry is filled into the TLB. The infrequent writing to the TLB entries makes TLBs particularly well suited to be built with STT-RAM, which usually has a long write latency but relatively short read latency. To verify this observation, we examine various GPU benchmarks and found that out of 13 benchmarks, only one benchmark (SAD) has similarly large write ratio (total number of writes divided by total number of reads) in both the data cache and TLBs. For all other benchmarks, the TLB write ratio is significantly lower than the cache write ratio, as illustrated in Figure 3 For example, in TLB, more than 40% of the data cache accesses are writes while less than 10% of the TLB accesses are writes. Based on this observation, a TLB implemented with STT-RAM is less harmed by the influence of the

expensive writes but can still take advantage of the improved read speed from differential sensing and larger same-area replacement capacity to achieve better performance and lower energy consumption.

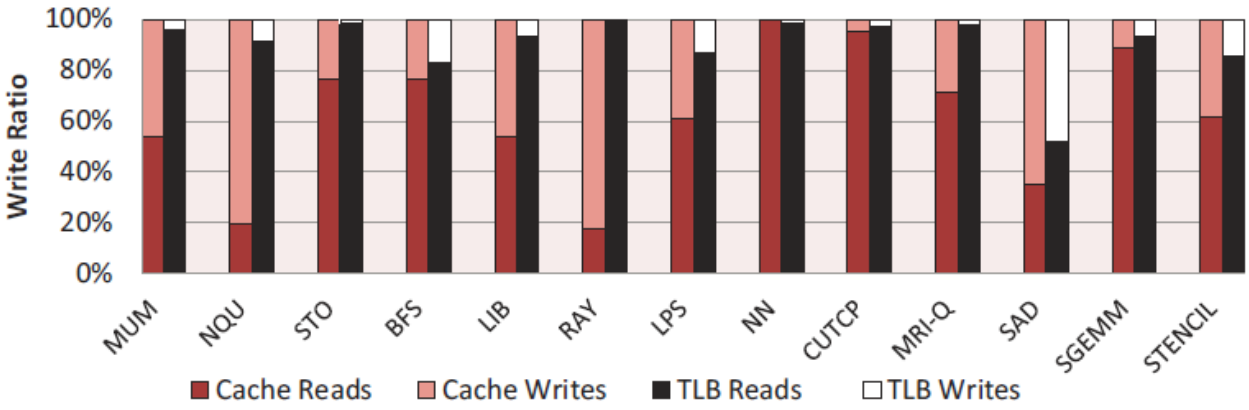

Figure 3. Comparison of write ratio for data cache and TLB.

2.2.2 TLB design with STT-RAM

2.2.2.1 Standard STT-RAM TLB design

Due to the structural simplicity, STT-RAM has much higher integration density than SRAM. Table I compares the design parameters of two SRAM-based and STT-RAM-based TLB configurations based on ITRS 2011 [25] and NVSIM [33] simulation results. In each configuration, the three designs with different memory technologies (i.e., SRAM, 1T1J and 2T2J) share similar area while the capacity of the standard STT-RAM TLB (1T1J) is 4× as large as that of SRAM TLB. The capacity of an entirely differential sensing design (2T2J) is between

that of the SRAM and standard STT-RAM since it trades half of its capacity for faster read access.

Table 1. Comparison of SRAM and STT-RAM based TLB

TLB Config.	TLB Config. 1			TLB Config. 2		
Technology	SRAM	STT-RAM		SRAM	STT-RAM	
	(1K)	1T1J(4K)	2T2J(2K)	(16K)	1T1J(64K)	2T2J(32K)
Memory Area(mm^2)	0.037	0.043	0.043	0.434	0.505	0.505
Sensing Time(ns)	0.49	1.13	0.71	0.49	1.13	0.71
Total Read time(ns)	1.78	1.81	1.39	7.24	6.78	6.36
Total Write time(ns)	2.48	11.37	11.37	6.89	14.02	14.02
Read energy (nJ)	0.12	0.06	0.06	0.13	0.09	0.09
Write energy (nJ)	0.13	0.39	0.79	0.15	0.49	0.98
Leakage power (mW)	62.49	21.86	21.86	522.67	157.48	157.48

2.2.2.2 STT-RAM-based dynamically-configurable TLB

A standard STT-RAM TLB configured for 1T1J access provides the best capacity and energy advantages among all TLB designs. The differential sensing design (2T2J) improves read performance, but increases miss rate due to the reduced capacity. As part of this work, we propose a STT-RAM-based dynamically-configurable TLB (STD-TLB) design that leverages the differential sensing technique to improve the read performance of selected memory blocks in a TLB while retaining the capacity advantage available in standard STT-RAM TLB design as much as possible.

Figure 4 shows the memory structure of our STD-TLB design: In mode 0 (high-capacity mode), the stored data is read out by comparing the resistance of the memory cell to a reference cell; in mode 1 (high-performance mode), the resistances of the complimentary memory cells are compared to each other. Since the sense margin in mode 1 is 2× that in mode 0, the read access time is reduced. However, two memory cells are now occupied to store a single entry in mode 1.

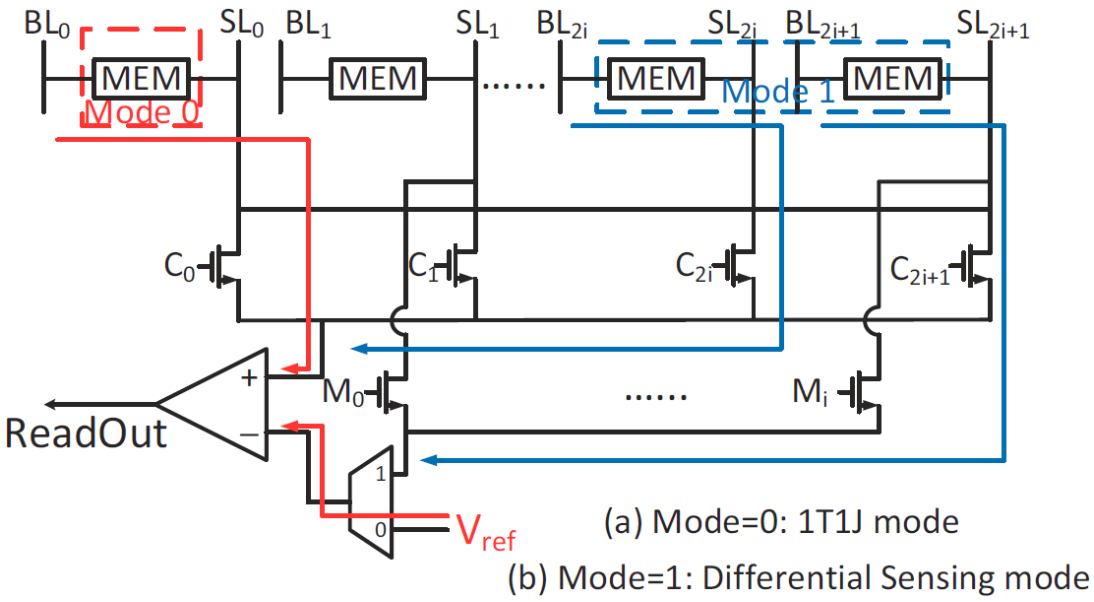

Figure 4. Reconfigurable differential sensing circuit.

We note that the read accesses to TLB have very unbalanced patterns. We study different TLB entries and group them into “hot” and “cold” based on their frequency of accesses. As an example shown in Figure 5, the distributions of the read accesses across different TLB entries is highly skewed in benchmarks STO, NN, CUTCP and MRI-Q. Other benchmarks experience less but still noticeable uneven distributions. Generally, there are far more accesses occurring on the

hot TLB entries (more than 75% on average). Hence, it is desirable to switch the selected cache blocks of STD-TLB to high-performance mode only when the corresponding memory addresses are hot. As illustrated in Figure 5, these active memory blocks experience frequent read accesses but only occupy a relatively small portion of the total memory blocks. Therefore, mode 1 can be effectively applied to a small number of TLB blocks during the operation, resulting in a marginal degradation on the total capacity and hit rate.

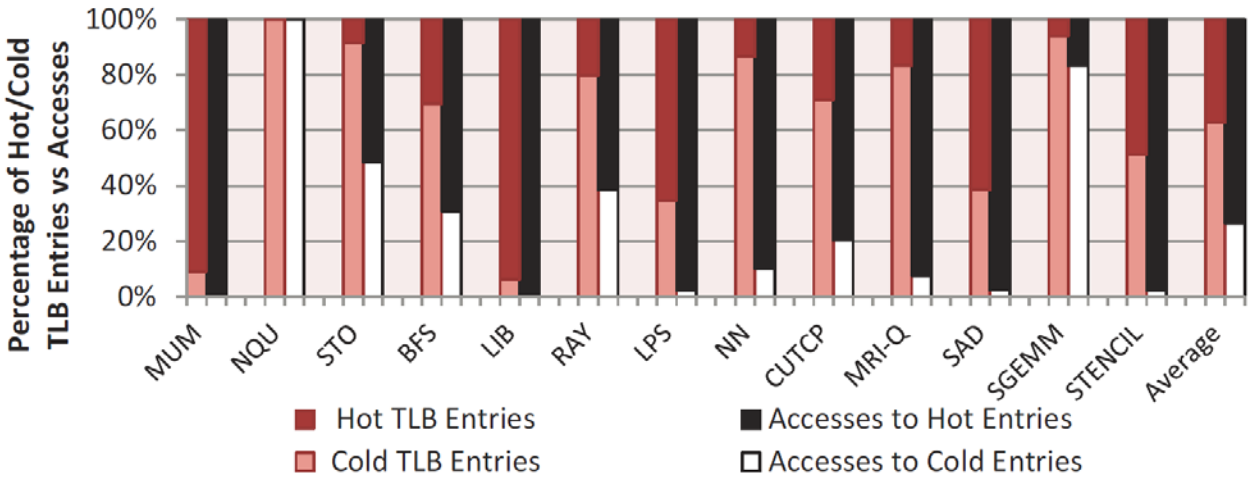

Figure 5. Unbalanced TLB accesses in GPU.

2.2.2.3 Organization of STD-TLB

Figure 6 shows the organization of the proposed STD-TLB design. Each two adjacent cache blocks form a complimentary block pair when any of them switches to high-performance mode. Each cache block is augmented with a 1-bit mode flag (S) and a n-bit read counter (C) and can function independently in high-capacity mode (S = 0). When the paired cache blocks are in

high-performance mode, both blocks' mode flags will be set to 1. The read counter is used to record the number of read accesses to the cache block.

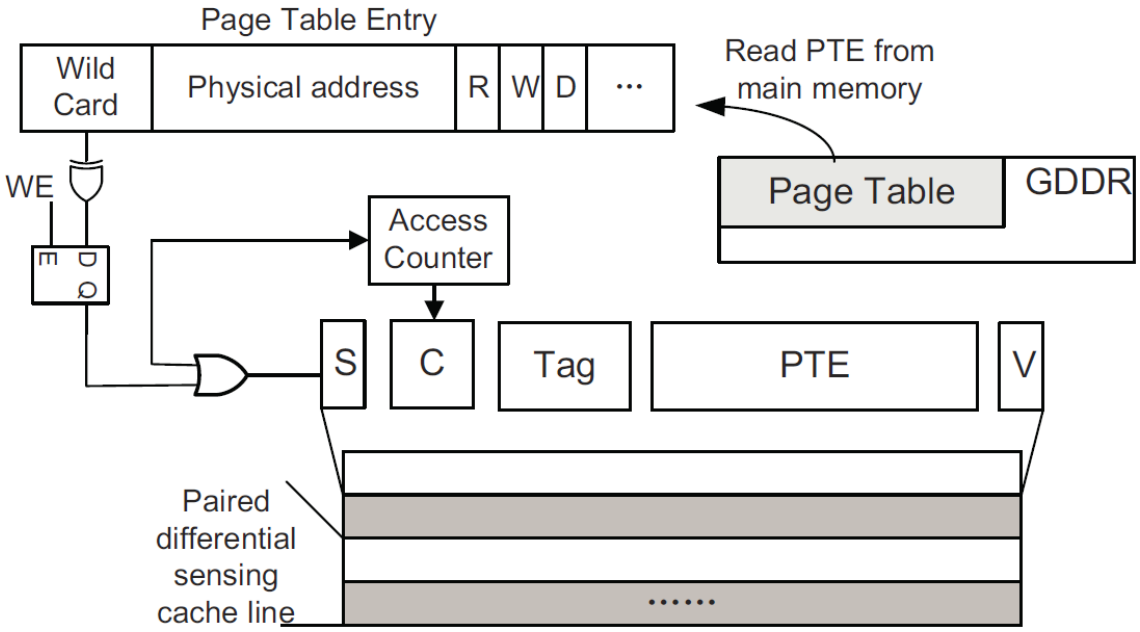

Figure 6. Organization of STD-TLB design.

2.2.2.4 TLB working mode management

Through execution analysis we discovered that the page size of a data accessed by a GPU is directly linked to its access patterns. A large page size corresponding to a wide accessible data range from one PTE typically results in more accesses over time. Based on our observation on 13 GPU benchmarks, the PTEs of large pages always have more accesses than that of small pages. Therefore, our TLB working mode management is set heuristically as follows: when a new PTE is loaded into the TLB, we first check the wildcard bits of the PTE to retrieve the page size

information. If the page size is bigger than 4KByte (i.e., wildcard > 0), the PTE will be written into the paired cache blocks in high-performance mode; otherwise, the PTE will be saved in the cache block in high-capacity mode.

Although this heuristic scheme works reasonably well, there are still some frequently accessed small pages that could not be covered by the scheme. To discover these pages, we utilize the augmented counter in each cache block to record how heavily the PTE is accessed. When a PTE is loaded into the TLB, the corresponding read counter is cleared to 0. The value of the counter increments upon each read access. The mode of the cache block is then dynamically reconfigured based on how intensively the page has been accessed: if the counter exceeds a threshold, the empty corresponding entry will be elevated to high-performance mode. In contrast, the entry set to high-performance mode (based on its page size) can be "demoted" to high-capacity mode during evictions: A high performance TLB entry which is planned to be retired by the replacement policy (e.g., least-recently-used) will switch to high-capacity mode first rather than being evicted from the TLB immediately. Only the entries in high-capacity mode can be evicted from the TLB directly.

2.2.3 Evaluation

We created a STT-RAM device model based on an industrial prototype. We assume the MTJ switching time is 10ns [34] and the sense margins of the STT-RAM cell in high-capacity and high-performance modes are *40mV* and *80mV*, respectively. A modified CACTI [31] tool is used to derive the peripheral circuitry latencies of various TLB designs at *45nm* technology [35]. The timing parameters of sense amplifiers are derived from SPICE simulations. All TLB design parameters are summarized in Table 1.

2.2.3.1 System Configuration and Workloads

We verify our architectural design through functional and timing simulations on GPGPUsim [36], a cycle accurate GPU performance simulator. We modified the baseline GPU architecture in GPGPU-sim based on Nvidia Quadro FX5800 [37] by adding virtual memory mapping support. In particular, we implemented a two-level TLB hierarchy including first-level private TLBs with small capacity and short translation latency, and a second-level TLB with much larger capacity and long translation latency, which is shared across all memory banks. We assume a dual-sized paging system with both small page (4K bytes) and large page (128K bytes). A large page is usually used for the applications with large memory footprint, e.g., enormous texture or color data.

Table 2. System configuration

GPU Configuration	Memory Clock 1250MHz, Memory bandwidth 102.4GB/s, Fillrate Pixel 20.736GP/s		
	Technology	Level-1 TLB	Level-2 TLB
TLB Configuration	SRAM baseline mode	4-way, 128 entries, 3-cycle latency	16-way, 2048 entries, 10-cycle latency
	STT-RAM (1T1J mode)	4-way, 512 entries, 3-cycle latency	16-way, 8192 entries, 9-cycle latency
	STT-RAM (2T2J mode)	4-way, 256 entries, 2-cycle latency	16-way, 4096 entries, 8-cycle latency
Memory/Paging Parameters	4Kb small page and 128Kb large page, 300-cycle page table access latency		

We use SRAM TLB as the baseline implementation. Our proposed STT-RAM TLB implementations assume the same area replacement associated with an increased capacity. The detailed parameters and system configuration are summarized in Table 2. Because the cache blocks in STD-TLB can independently operate in high-capacity or high-performance mode, the maximum and minimum capacities of the STDTLB are listed as 1T1J and 2T2J in the Table, respectively. In practice the effective capacity will be somewhere in between. In our evaluations,

we utilize a wide range of GPU workloads selected from the Parboil [38] and ISPASS2009 benchmark suite. Our intent is to primarily focus on the workloads which have high memory access intensity and thus could potentially benefit from STT-RAM-based TLBs. However, we also include some less access intensive workloads to test the potential impact of STT-RAM-based TLBs in different scenarios. The characteristics of the adopted workloads are listed in Table 3.

Table 3. Characteristics of GPU benchmarks (instruction number (in))

	Abbrev.	Benchmark	IN
ISPASS2009	MUM	MUMmerGPU, low cost ultra-fast sequence alignment	85M
	NQU	N-Queens Solver, finding an arrangement of queens	1M
	STO	StoreGPU, accelerating numbers of hashing	125M
	BFS	Breadth First Search, a graph operation	17M
	LIB	LIBOR, calculating the evolution of 80 forward rates	998M
	RAY	Ray Tracing in 3D computer graphic	63M
	LPS	3D Laplace discretisation Solver	73M
	NN	a Neural Network Recognition of Handwritten Digits	84M
	CUTCP	Cutoff-limited Coulombic Potentiall	3G
Parboil	MRI-O	Mri non-cartesian Q matrix calculation	1.8G
	SAD	Sum of Absolute Differences	300M
	SGEMM	SGEMM dense matrix operation	9M
	STENCIL	solving partial differential equations	2G

2.2.3.2 Experimental Results

To demonstrate the influences of using STT-RAM to implement TLB in GPUs, we study the mode configuration accuracy, TLB miss rate, translation runtime performance, energy

consumption, energy-delay product etc., and compare them with the SRAM baseline. We also evaluate the performance and energy improvement of STD-TLB over the standard STT-RAM TLB. Finally, we evaluate the overall system performance improvement and summarize the general rules of applying STD-TLB.

1) Mode Configuration Accuracy: We verify if the TLB entry of a small page is configured in an appropriate mode by using a counter to track its read accesses and comparing the value of the counter with a threshold. Ideally, the TLB entry corresponding to a small page shall be set to high capacity mode and has a low counter value, i.e., smaller than a threshold. Otherwise, the TLB performance may be adversely impacted because of its poor read speed. Heuristically, we set the threshold of defining a "hot" small page to 900. The configuration accuracy of the TLB entry of a large page is also measured by the probability of being demoted to high capacity mode. Figure 7 shows that our page-size-only mode selection mechanism captures averagely 83% of "cold" small pages and 90% of "hot" large pages. NN is an outlier and exhibits heavy reads for both small and large pages. On average, our page-size-only mode selection mechanism introduces a mode prediction accuracy from 40%~100% for both small and large pages. To improve the address translation efficiency, the promote/demote scheme is applied to STD-TLB atop the page-size-only mode selection mechanism, as presented in 2.2.2.4.

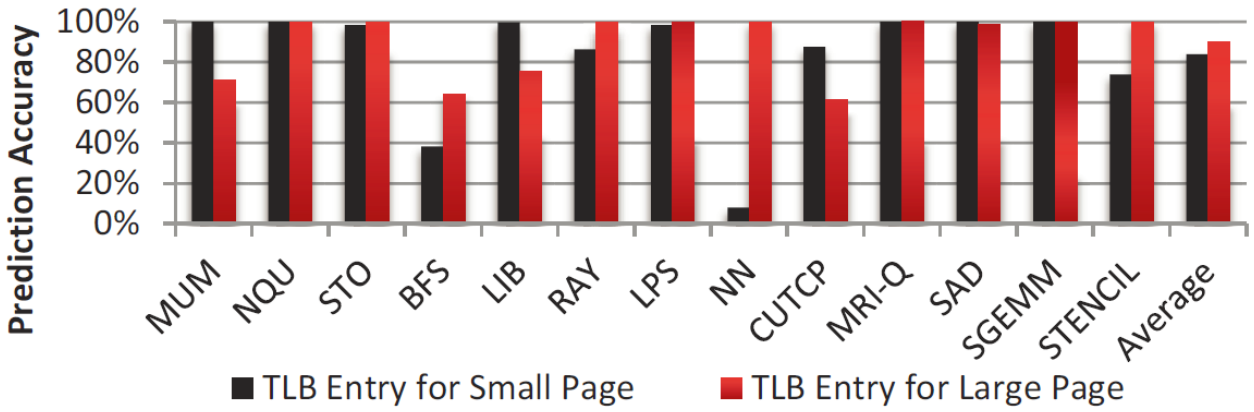

Figure 7. Percentage of correct mode configuration for TLB entries.

2) TLB Miss Rate: Figure 8 depicts the normalized overall miss rate of the TLB hierarchy for the tested benchmarks. Due to the larger capacity, standard STT-RAM TLB and STD-TLB exhibit significantly lower miss rates than SRAM TLB in 10 out of the 13 benchmarks. In NQU and SGEMM, all three designs perform closely because the kernels of these benchmarks exercise small working sets that can be effectively accommodated by a small TLB. For MUM, BFS, LIB and SAD, standard STT-RAM TLB drastically reduces the miss rate. STD-TLB also considerably reduces the miss rate of BFS, LIB, and SAD. But the reduction is less significant than standard STT-RAM TLB because of the smaller runtime effective capacity. Under these scenarios, the applications typically utilize more large pages with intensively accessed TLB entries. On average, standard STT-RAM TLB reduces the miss rate by 38% while STD-TLB reduces the miss rate by 30%, compared to the SRAM baseline.

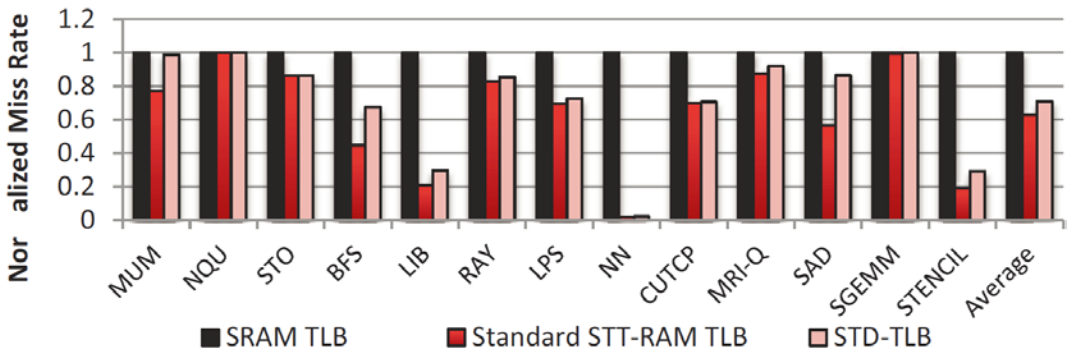

Figure 8. TLB miss rate in form of off chip misses compared to all accesses.

3) Performance: The address translation runtime performance improvements [39] of different designs are presented in Figure 9 Standard STT-RAM TLB achieves a significant speedup because of the prominent miss rate reduction. Although STD-TLB has a higher miss rate, STD-TLB further improves the runtime performance by performing a faster address translation for most accesses. We can see that other than two benchmarks – NQU and SGEMM, which have relatively small data sets, all other benchmarks experience remarkable runtime performance improvements under STD-TLB. On average, standard STT-RAM TLB improves the translation performance by 32% over SRAM baseline while STD-TLB further improves it by 55% and 15% compared to SRAM TLB and standard STT-RAM TLB, respectively.

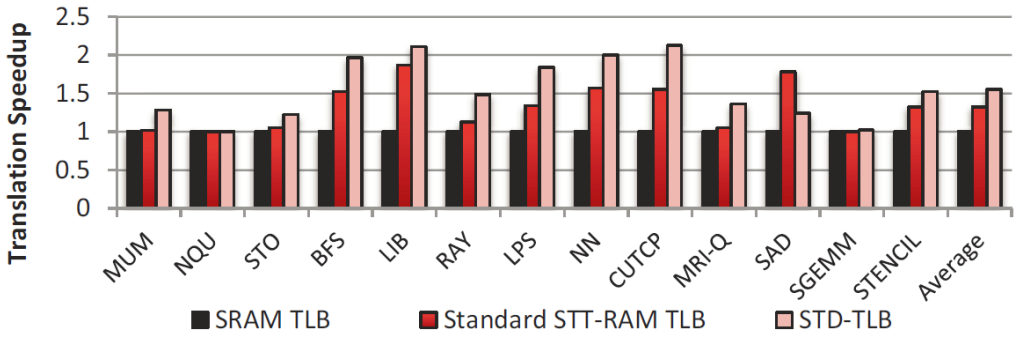

Figure 9. TLB translation performance improvement.

4) Energy: The energy comparison of all designs is shown in Figure 10. Both standard STT-RAM TLB and STD-TLB achieve over 70% leakage energy reduction compared to SRAM TLB. STD-TLB has slightly higher dynamic energy due to the higher write energy consumption in high-performance mode. For benchmarks that exhibit heavy write loads, e.g., NN and SAD, STD-TLB consumes considerably more dynamic energy than standard STT-RAM TLB. However, dynamic energy increases are negligible for other benchmarks. On average, standard STT-RAM TLB and STD-TLB achieve 57% and 49% energy savings over SRAM TLB, respectively.

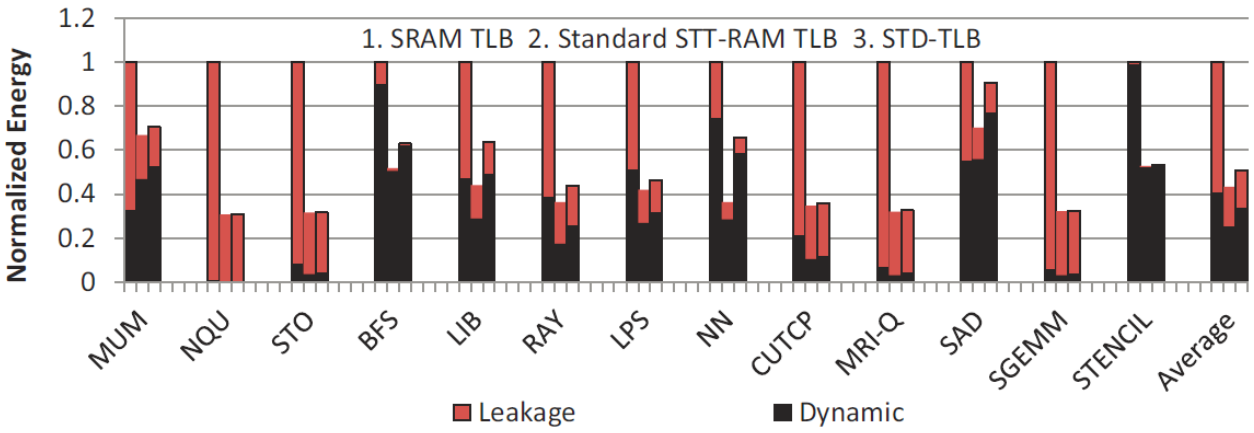

Figure 10. Dynamic and leakage power consumptions for different TLBs.

5) Energy Delay Product: The overall benefit by both translation performance and energy is often denoted by energy delay product (EDP). As illustrated in Figure 11, standard STTRAM TLB achieves over 75% EDP reduction w.r.t. SRAM TLB while STD-TLB further boosts it to nearly 80%. The significant improvement is mainly from: 1) low leakage power induced by STT-RAM technology; 2) low miss rate as a result of the increased TLB capacity; and 3)

reduced translation latency. STD-TLB further achieves 13% EDP improvement over STT-RAM TLB regardless its nominally increased energy consumption. STD-TLB performs worse than standard STT-RAM TLB only in SAD because of the increased dynamic energy and translation latency induced by the heavy write loads and high miss rate.

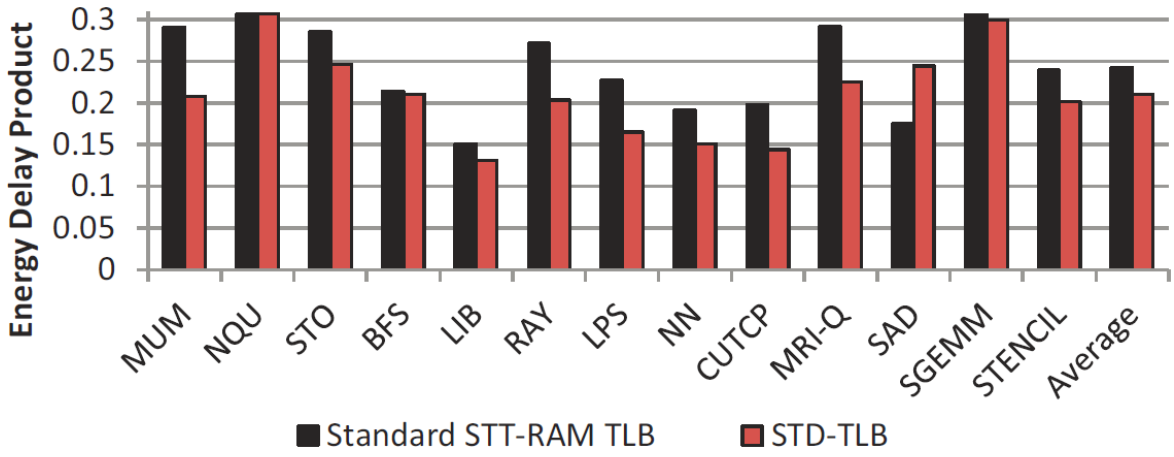

Figure 11. Energy delay product improvements (normalized to SRAM TLB).

6) System Performance: As shown in Figure 12, standard STT-RAM TLB achieves very moderate performance improvement in many benchmarks. It is because modern GPU designs intent to hide memory latency, which is the main optimization goal of our proposed techniques. However, significant performance improvements are still achieved in some benchmarks like BFS (9.2%) and LIB (10.8%). After employing STD-TLB, significant GPU system performance improvement is achieved in MUM and STENCIL, say, 15% and 6%, respectively. On average, STD-TLB achieves 4% and 2% system performance speedup compared to SRAM TLB and standard STT-RAM TLB, respectively.

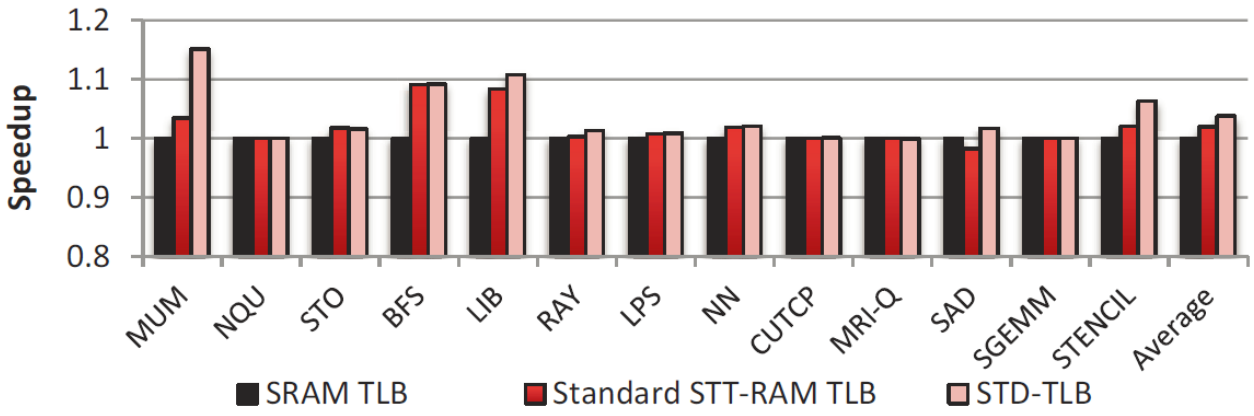

Figure 12. Overall system performance speedup comparison of different TLBs.

2.3 MLC STT-RAM BASED REGISTER FILE

2.3.1 Modern GPU register file

Threads are executed in single instruction multiple thread (SIMT) fashion in GPUs to maximize computation parallelism and throughput. Without loss of generality, in this work, we adopt the terminologies of Nvidia and use GTX480 [41] in Fermi family as the baseline model.

A GPU is composed of 16 streaming multiprocessors (SMs), each of which includes one GPU processing pipeline. GPU kernel is instantiated with a grid of parallel thread blocks. The maximum number of the threads that can be concurrently executed in one thread block is 1024 with maximum 63 32-bit registers assigned to each thread. 32 threads in one block are grouped as a warp to be issued. Before launching a thread block, the CUDA compiler checks the number of available registers: if it is smaller than the total register number requested by the thread block, the launching will be suspended.

As illustrated in Figure 13(a), a GPU processing pipeline includes Instruction Unit, Register Address Unit, Register File (RF), and Execution Unit [43]. Instruction Unit decodes instructions, and schedules the execution based on additional information like priority and data dependency. The instructions selected in a round-robin fashion are sent to Register Address Unit at the end of every clock cycle. Accordingly, Register Address Unit accesses the registers in the RF.

The RF in a SM contains 32,768 32-bit registers to hold the instruction operands. Highly banked architecture is usually utilized to realize multi-port access in the RF implementation. In Fermi architecture, one SM integrates 16 banks, each of which contains 64 entries of 1,024 single-port SRAM cells (i.e., 32 registers). As shown in Figure 13(b), an operand collector in RF is used to collect and dispatch the active warps to available banks. When all the operands have been acquired, instructions and operands are output to Execution Unit.

Very recently, Emerging memories is discussed to be used to implement a much denser RF and other types of GPU on-chip memory for power and performance improvement [44][45][46][47]. Our proposed MLC-STT-based RF not only further enlarges the potential of RF capacity, also successfully overcomes the long write operation and the unbalanced accesses to different MLC bits, and demonstrates both power and performance improvement as presented in our evaluation.

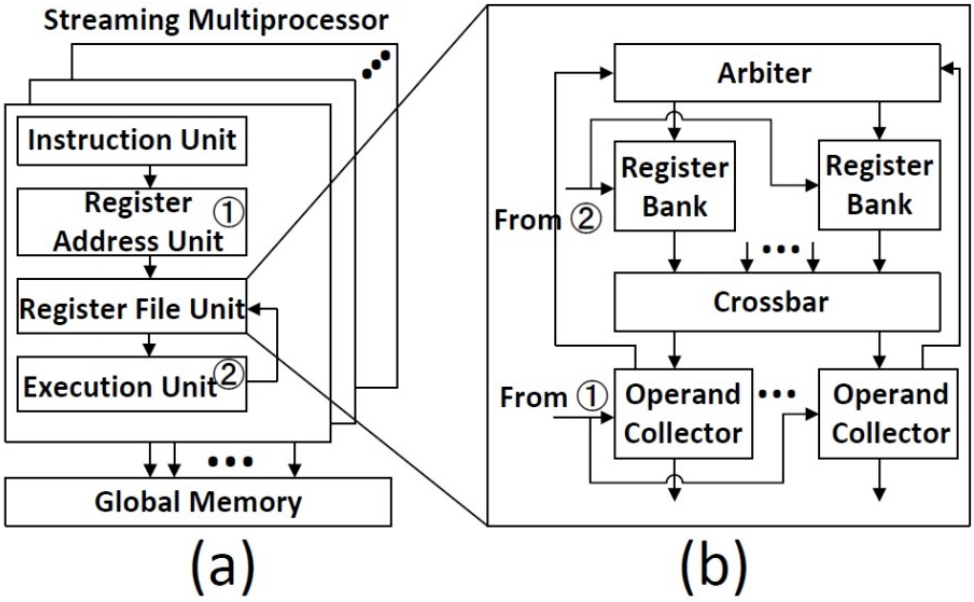

Figure 13. GPU architecture and the register file design in GPU.

2.3.2 MLC STT-RAM based register file

Figure 14 illustrates the proposed MLC-STT based register file (MLC-RF) in Fermi architecture. Each RF entry contains 1,024 MLC-STT cells, corresponding to 2,048 data bits, which can be further divided into two logic entries, a fast row with 1,024 soft-bits and a slow row with 1,024 hard-bits. A row decoder is also implemented to support the accesses to the logic rows by employing the lowest address bit to control the access to the soft or the hard bits of each physical entry.

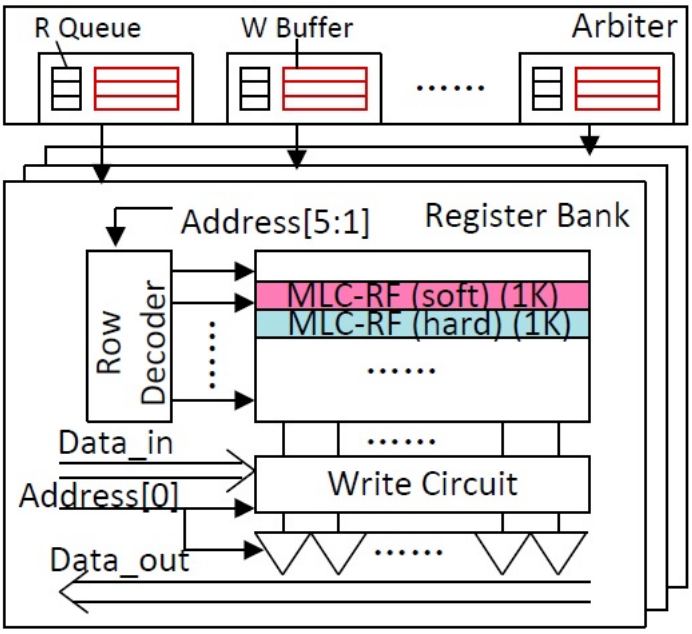

Figure 14. **MLC STT-RAM-based register file.**

Table 4 summarizes the design parameters of 64Kb register banks built with SRAM and MLC-STT at 32nm technology node. The results are simulated by CACTI [31] and HSpice with PTM [32] under 32nm technology. In MLC-STT design, we assume the area of the big MTJ (hard bit) is twice as that of the small MTJ (soft bit), which achieves the best robustness in read and write operations [20]. The timing information of sense amplifier is derived from SPICE simulations. The adopted RF configuration and area information is obtained through a modified NVsim [33]. The area of MLC-STT design is only about 13.8% of that of SRAM design.

Write Buffers are adopted in Arbiters to alleviate the impact of slow access time of MLC-STT design. Each register bank requires one Write Buffer. The Write Buffer temporally holds the writeback data until the former write is completed and releases write port. The needed number of write buffer entries is determined by the cycles and frequency of MLC-STT write

access. Based on the evaluation of the selected benchmarks, a 4-entry Write Buffer is sufficient to remove the overflow since writebacks occur infrequently and the RF is highly banked.

Table 4. Comparison of SRAM and STT-RAM based register banks

Technology	SRAM	MLC-STT
Memory Area (mm^2)	0.350	0.0486
Total Read time (ns)	0.710	S: 1.25, H: 1.63
Total Write time (ns)	0.708	S: 7.18, H: 14.86
Read energy (nJ)	0.044	S: 0.018, H: 0.023
Write energy (nJ)	0.044	S: 0.087, H: 0.14
Leakage power per SM (mW)	100.416	0.8125

In the original arbiter design in Fermi architecture, a read queue is assigned to each register bank to arrange all the read requests from different Operand Collectors in a row. The Read Queue structure remains unchanged in our proposed MLCRF, holding read requests when awaiting the slow MLC write operations to finish. The introduction of Write Buffer incurs 15% increase in MLC-RF bank area. Nonetheless, the overall area of a MLC-RF bank is still less than 30% of the one of SRAM implementation, mainly benefiting from the high data storage density of MLC-STT.

2.3.2.1 MLC-aware Remapping Strategy

Write Buffers enhance the response time of RF but cannot help on reducing the long read and write latencies of MLCSTT banks. As aforementioned in 2.1.3, MLC-STT contains soft- and

hard-bit rows which have very distinctive read and write performance. It can be beneficial if the GPU maximizes the usage of soft-bit rows for sake of system performance and energy consumption. In other words, the data (especially the one being accessed frequently) shall be mapped to the soft-bit rows before considering the hard-bit rows. In the implementation, run-time information such as how many registers to be accessed and how many times they will be accessed can be obtained from the compiler. During compilation, we first perform profiling on the access frequency of all the registers involved during the execution of a kernel. A bitmap vector is then generated to record whether a register will be placed in soft-bit row or hard-bit row. Once the kernel is offloaded to SMs, such a bitmap vector is copied to a special register and guide the register mapping.

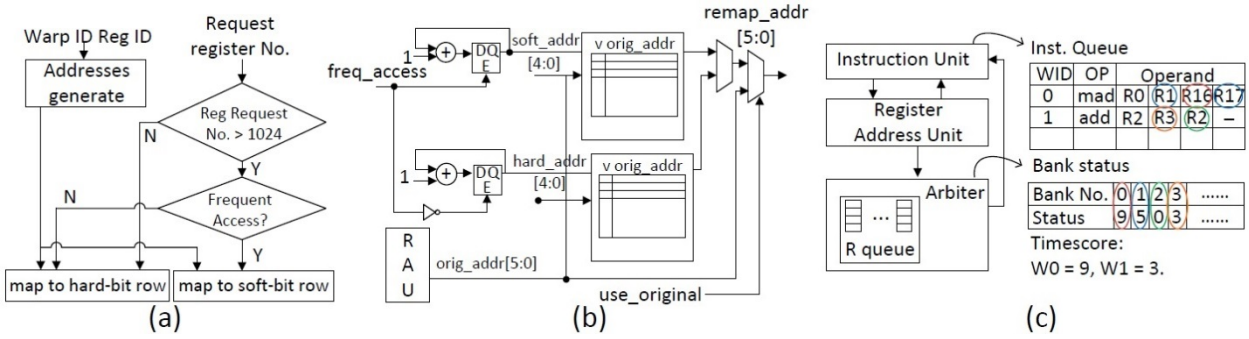

Figure 15 (a) The register bank address remapping algorithm. (b) The hardware implementation diagram of remapping. (c) Warp rescheduling.

Figure 15(a) depicted the corresponding register file remapping algorithm, which requires the support of two mapping tables and one register address unit (RAU), as shown in Figure 15(b). Remapping is activated only when the register demand exceeds the half capacity of the MLC-STT register bank. Otherwise, hard-bit rows will not be utilized. When more than half

capacity of the MLC-STT register bank is requested, the frequent-accessed registers will be mapped to the soft-bit rows first while the rest will go to the hard-bit rows. The relationship between the original address and the mapped physical address is retained in a remapping table. Each remapping table entry corresponds to one row in the register bank, including 1 valid bit and 6-bit mapped address if RF is configured as Fermi architecture. Hence, the size of the remapping table is small and the incurred area overhead is negligible.

2.3.2.2 Warp Rescheduling

The long write latency of STT cells (even in the soft-bit rows of MLC-STT) may severely degrade the system performance by blocking other accesses to the same register bank. Figure 15(c) describes such an example, where two warps – W0 and W1, are waiting in the instruction buffer in a serial Round Robin manner. Their register requests are mapped to different banks and they do not have data dependency with the current warps being executed. When the access to the register bank thatW0 requests is held by a writeback from the previous warp, the operand fetching of W0 has to wait. Since all the available entries in Scoreboard, Operand Collector, and Read Queue have been held by W0, the processing of W1 is stalled at the issue stage even the register bank to be accessed by W1 is free. In such a case, we may swap the issue sequence of W0 and W1 to avoid the stalling of W1. Based on the above observation, we propose a warp rescheduling scheme to minimize the waiting time of the issued warps for register bank access. The rescheduling tends to rearrange the issue order of the ready warps by prioritizing the accesses to the free register banks. The effectiveness of rescheduling is mainly determined by the availabilities of the free register banks and the warps ready to be issued, which are generally large in GPU execution. A parameter, $timescore$, for each warp is defined to guide warp rescheduling as:

$$timescore = Max[busy\_cycle[bank\_id_i] \& valid[bank\_id_i]] \quad (1)$$

Here $busy\_cycle$ indicates the remaining cycles to complete the current write operation of the register bank, and valid denotes whether the warp has a register request from this bank. $bank\_id$ labels the register bank and is determined by the identifier of the register (reg_id) by:

$$\boldsymbol{bank\_id} = reg\_id \,\%\, bank\_number \quad (2)$$

Here $bank\_number$ is the total number of register banks in one SM, which is 16 in Fermi architecture. As shown in Figure 15(c), a bank status table is added in Instruction Unit to assist acquiring the $busy\_cycle$ of each register bank. $busy\_cycle$ is updated by RF whenever new write requests arrive and automatically counts down every cycle.

During scheduling, $timescore$ will be checked whenever warps are going to scoreboard for data dependency check. As shown in Eq. (1), the maximum $busy\_cycle$ will be chosen as the $timescore$ of a warp. The warp will be managed to issue only when its $timescore$ smaller than a threshold. If a warp failed to pass the $timescore$ checking, it will remain in Instruction Buffer and wait for next round check. The scheduler will continue to check the next available warp until one available warp is issued successfully. Note that data dependency checking and $timescore$ checking can be performed simultaneously. Hence, no timing overhead on scheduling step is introduced. We run extensive experiments and found that 5 is the optimal value of the $timescore$ threshold which maximizes the system performance and energy efficiency of our design.

2.3.3 Evaluation

2.3.3.1 Evaluation Setup

We implement all the designs on GPGPU-sim [36], a cycle accurate GPU performance simulator, for system functionality verification and performance evaluation. The baseline GPU is configured as Nvidia GTX480 [41]. Table 2 summarizes the parameters of our system configuration. Loose round-robin (LRR) scheduler, which is widely adopted in GPU warp scheduling, is selected as the basic scheduler. The SM operating frequency is set to 700MHz. The parameters of the RF are generated from a modified NVsim [33] at 32nm technology node. The STT device parameters from [32][42] are adopted for cell area estimation. The detailed configurations of register banks can refer to Table 5.

Table 5. GPGPU-sim configuration

Parameter	Value
Number of SMs	16
Core/Shader/DRAM Frequency	700/1400/924MHz
Register File/SM	128KB
Max Warps/SM	48
Max Threads/SM	1536
Max Thread Blocks/SM	8
Max Registers/Thread	63
Max Threads/Thread Block	1024
L1/Shared Memory	16KB/48KB
Warp Scheduler	Round Robin

Ten GPU workloads from [36][40] are selected in our performance evaluations. Table 6 shows that the register file usages of the selected workloads vary from 27.2% to 92.9%, which offers a good coverage on all representative cases.

Table 6. Characteristics and register file usage statistics of 10 selected GPU benchmarks

Application	Abbr.	Regs/Thread	Max Warps	Reg Usage
Matrix Q Computation	MRI-Q	12	48	84.3%
Hotspot	HOT	34	24	79.7%
BlackScholes	BLK	25	32	78.1%
Neural Network	NN	21	8	27.2%
Lattice-Boltzmann Method	LBM	36	27	92.9%
Breadth First Search	BFS	12	48	56.2%
Path Finder	PF	13	48	90.9%
Streamcluster	SC	12	48	56.3%
Discrete Cosine Transform	DCT	32	16	50.0%
Back Propagation	BP	11	48	81.5%

2.3.3.2 System Performance

Figure 16 summarizes the GPU performance with different register file configurations over the selected benchmarks. All the results are normalized to that of the baseline with SRAM register file (“SRAM”). As the area of MLC-STT implementation only occupies 13.85% of SRAM, it explores the possibility of a larger register file to support more threads operating at the same time. We build a same-sized (2MB), a 2x (4MB) and a 4x (8MB) RFs to find the proper configuration. It shows that simply replacing SRAM with same-sized MLC-STT without any optimizations (“MLC”) result in an average 11.4% performance degradation. In particular,

significant system performance degradation (>14%) was observed in the applications that have intensive register file accesses, e.g., MRI-Q and BLK, due to the long read and write access latencies of MLC-STT. As increasing RF size, some benchmarks with high register file usage archive better performance due to more threads operating at the same time. HOT, LBM, and BP even outperform the SRAM baseline. But the performance stops improving when the capacity of RF is larger than 4MB since the number of concurrently executed threads is limited by other resources like shared memory. Hence, we use 4MB RF configuration in the following experiments.

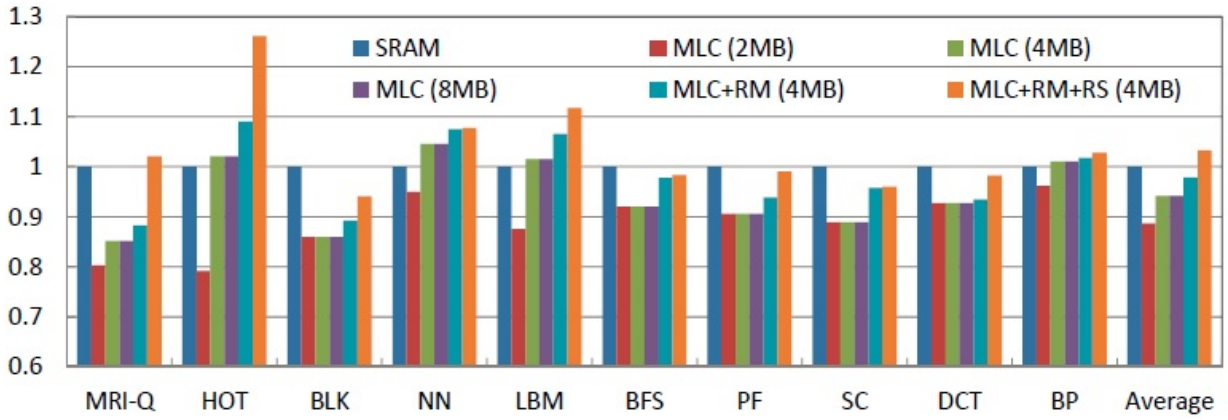

Figure 16. System performance comparison under different register file configurations. All the results are normalized to that of SRAM baseline design.

Interestingly, NN also shows performance improvement even its RF usage is only 27.2% across all kernels. Our detailed analysis found that the RF usage of NN is concentrated on only one kernel. Hence, the increased RF capacity greatly raises the number of the threads that can be issued on that kernel, resulting in considerable performance improvement.

When remapping strategy is applied ("MLC+RM"), the twostep read and write operations of MLC-STT RF are minimized, introducing on average 3.59% enhancement in system performance compared to "MLC". Remapping is particularly effective in the benchmarks sensitive to RF access latency: In SC and HOT, for example, the performance improvements w.r.t. "MLC" are as high as 7.5% and 7.01%, respectively.

After applying the rescheduling scheme ("MLC+RM+RS"), the GPU performance substantially improves 7.35% compared to "MLC+RM". Among five register-hungry benchmarks (MRI-Q, HOT, NN, LBM, and BP), MLC+RM+RS even outperforms SRAM baseline quite significantly, say, on average 10.4% speedup! Across all benchmarks, MLC+RM+RS still outperform SRAM baseline by 3.28%.

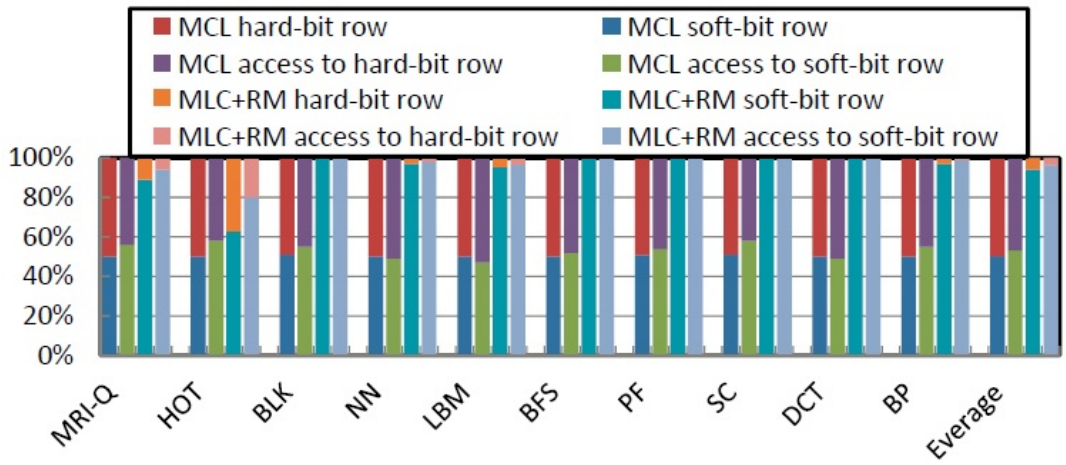

Figure 17. The statistics of soft-/hard-bit row accesses with/without remapping.

2.3.3.3 Effectiveness of Remapping and Rescheduling

To further evaluate the effectiveness of the proposed remapping strategy, we compare the ratio of the accesses and the usage of the hard-bit and soft-bit rows in each benchmark before and after the remapping strategy is applied. Figure 17 shows the statistical results. Without remapping, on average 53.3% of accesses fall on soft-bit rows, accounting for 50.2% of register bank entries in use , while the RF accesses are evenly distributed to soft-bit and hard-bit rows. The situation changes dramatically after applying the remapping strategy: the accesses to soft-bit rows are greatly promoted to 96.6% as averagely 94.1% of overall soft-bit rows are occupied.

The effectiveness of the proposed rescheduling scheme can be represented by $timescore$ of each warp after it is issued. As shown in Figure 18, originally majority (>80%) of issued warps need to wait for more than one cycle before successfully retrieving the operands from the RF. After applying the rescheduling, more than 50% warps can immediately access the RF while the maximum $busy\_cycle$ of all the warps reduces from more than 12 cycles down to 5 cycles.

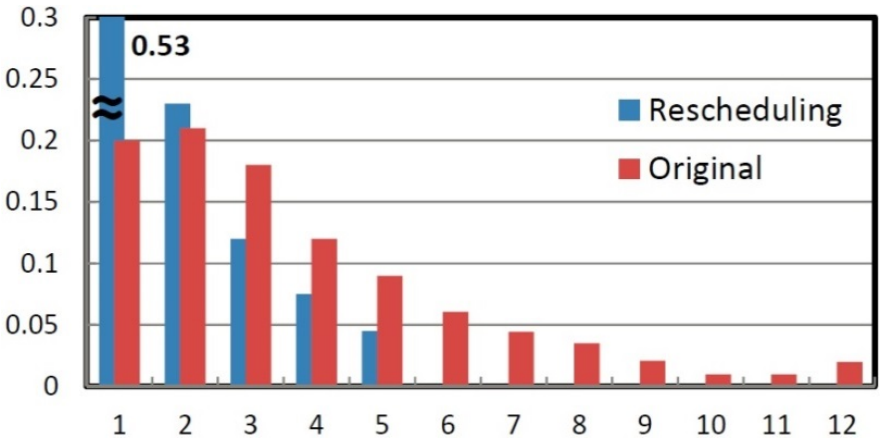

Figure 18. Rescheduling influence on timescore of issued warps.

2.3.3.4 Energy Consumption and Energy Efficiency

Figure 19 shows the energy consumption of the RF with different configurations normalized to SRAM baseline, including both dynamic and leakage energies. The leakage energy consumption of the RF dramatically reduces when MLC-STT is directly applied thanks to the non-volatility of STT-RAM, but dynamic energy consumption is increased due to the complex 2-step write and read operations. On average, MLC increases the total energy consumption by 16.2%.

In general, the dynamic energy of MLC-STT RF dramatically reduces when remapping strategy is applied: On average, MLC+RM consumes 9.48% less energy than SRAM. Note that rescheduling scheme does not reduce the number of write operations. Hence, it does not affect energy dissipation visibly. All the simulations above have taken into account the energy consumed on the additional control circuits.

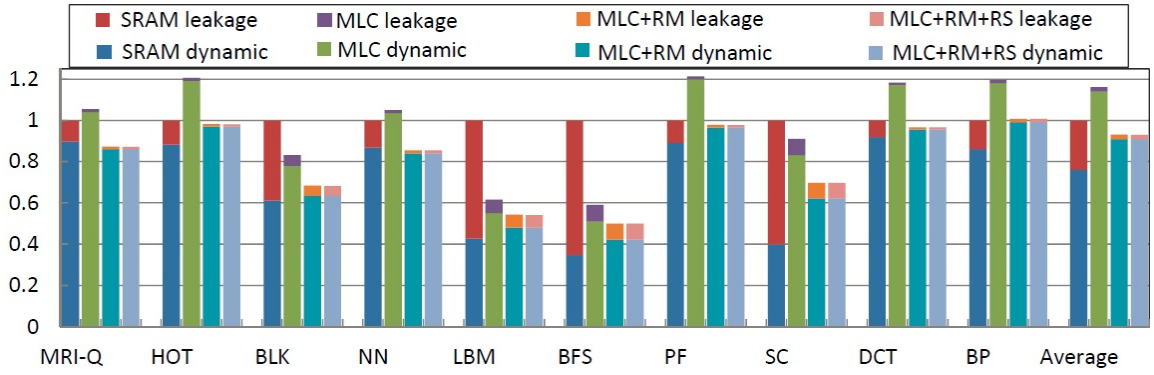

Figure 19. Register Energy consumption under different register file configurations. All the results are normalized to that of SRAM baseline design.

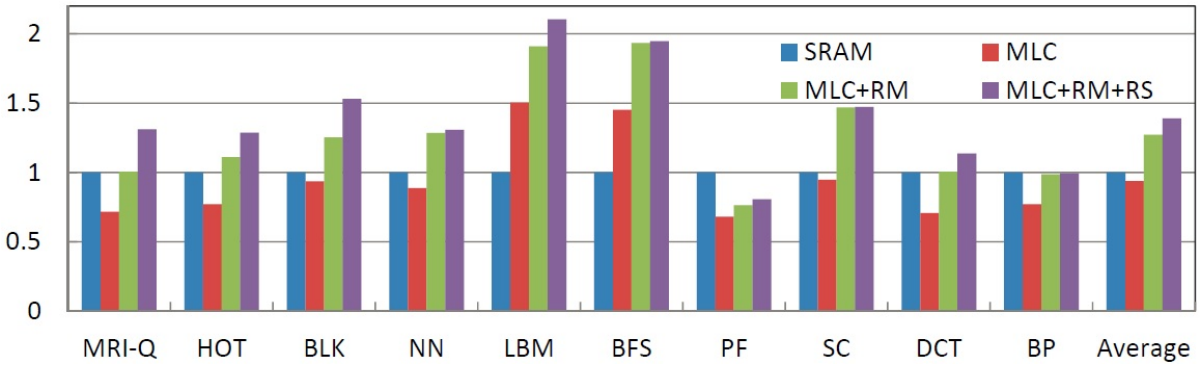

Figure 20. **Normalized energy efficiency.**

In this work, we use the ratio of the normalized performance over energy from [48] to measure the energy efficiency of our proposed MLC-STT RF design. Figure 20 shows the normalized energy efficiency of various RF configurations with and without optimizations. Our design, i.e., MLC-STT RF with remapping strategy and rescheduling scheme ("MLC+RM+RS"), achieves the best energy efficiency in 7 out of 10 benchmarks. On average, it is 38.9% more efficient than SRAM baseline.

3.0 EMERGING MEMORY APPLICATION FOR ACCELERATION

3.1 PRELIMENARY

3.1.1 Artificial neural network

Artificial neural network (ANN) can be also considered as one kind of approximate computing with high adaptivity to many high-performance applications [51]. The inherent resilience to soft and hard errors in computation makes ANN a promising solution to conquer the aggravated system reliability issue under the highly-scaled technology nodes [52].

MLP belongs to feedforward ANNs that have been widely utilized in approximate computing [49][50]. It maps a set of input data to outputs through multiple layers of nodes in a directed graph, in which every layer is fully connected to the next layer. Figure 21(a) shows an example of 3-layer MLP implementation. The input nodes collect and convey the input bits to the following layer through the weighted connections. A weighted connection (or synapse) is associated with a preset weight to modulate the carried signal. Except for the input nodes, each node represents a neuron with a nonlinear activation function, e.g., a sigmoid function $f(x) = \frac{1}{1+e^{x}}$ on the sum of all the signals it receives.

AAM is normally used as recurrent neural networks, performing pattern recognition and completion [53]. Figure 21(b) shows a Hopfield network acting as an AAM. Each pair of

neurons in the network are bridged through a weighted connection. An input vector distorted by noises or other randomness will go through the network iteratively and converge to the closest version of the vector pattern. In general, the non-iterative MLP implementation executes faster than the iterative AAM implementation while the latter is much more dependable due to its inherent fault tolerance characteristic.

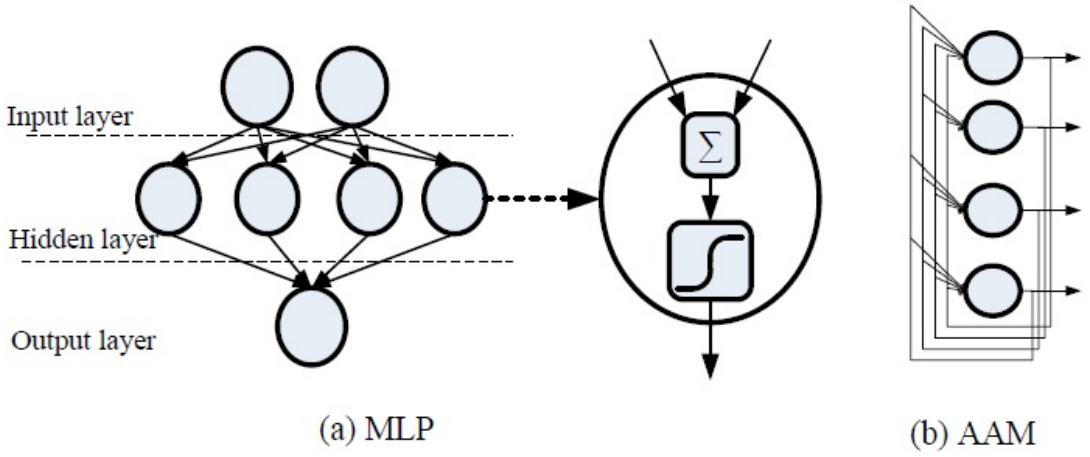

Figure 21. **(a) A 3-layer MLP; (b) A 1-layer AAM with 4 neurons.**

As an important operation of ANN, training determines the weight associated with each connection and prepares the ANN to properly respond to certain unseen data with desired outputs. The completion of training makes the ANN properly respond to certain unseen data with the desired outputs. In this work, we adopt back-propagation and delta rule [54] to perform the training of MLP and AAM. Our architecture level contributions mainly focus on the testing/computation process of the ANN by assuming the RENO has been trained by supervised algorithms for specific applications. No further modification on the configuration of RENOs is required during the computation except for the inline.

3.1.2 Memristor and memristor-based crossbar (MBC)

Memristor is defined as the 4th fundamental circuit element whose resistance (memristance) is determined by the total electric charge/flux through itself [55]. The existence of memristor was predicted in theory by Professor Chua in 1971 [55]. In 2008, HP Labs first reported a memristor device based on a TiO2 structure [56]. Afterwards, the memristive characteristic was observed in many other materials. In theory, a memristor can be programmed to any arbitrary resistance within its lowest and highest bounds by appropriately controlling the amplitude and duration of the programming current/voltage [57][58]. The recent research has obtained 7-bit programming resolution on a single memristor device [59] with sophisticated peripheral circuit.

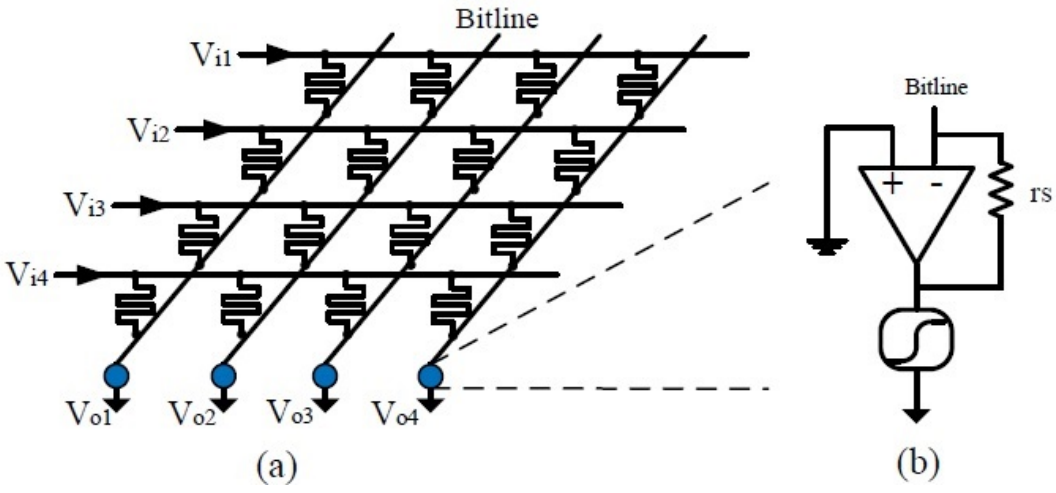

Figure 22 (a) A 4x4 MBC array; (b) the neuron logic.

Similar to biological synapses, a memristor device can “record” the historical profile of the applied excitations as its resistance change. This feature inspired many studies on memristor-

based synapse designs [60][61][62]. For example, memristors can be employed in spiking networks and trained by using spike timing dependent plasticity (STDP) learning rule [61][62][63][64]. Moreover, the recent studies presented the use of MBCs in perceptron network construction [65][66], and demonstrated extremely efficient, accurate, and fast ANN implementations. Figure 22 depicts the diagram of a MBC which represents the connections between two layers in a MLP. The relationship between the input voltages (V_i) and output voltages (V_o) can be defined as:

$$\boldsymbol{V_o = C \times V_i} \tag{3}$$

Here C is the connection matrix. In a real design, due to the existence of sensing circuits at the outputs of the MBC, the relationship between the connection matrix and the resistance matrix of a MBC is not a direct one-to-one mapping but an approximation. The implementation of a N-layer MLP requires $N-1$ MBC arrays connected in series. A large volume of ANN computations (i.e., weight multiplications) can be simultaneously performed by the MBC in analog form without any internal control logics. In this work, we adopt the MBC programming method in [66] where an adaptive write driver [59][58] is used to program the memristors to particular resistance states and many memristors on the same row are tuned simultaneously. In such a design, the sneak path issue is significantly suppressed.

3.2 THE RENO ARCHITECTURE

Figure 23 depicts the proposed RENO structure. It works as a complementary functional unit to CPU and particularly accelerates ANN-relevant executions. In the design, memristor-based crossbar (MBC) arrays are used to perform analog neuromorphic computation. And a

mixed-signal interconnection network (M-net) is developed to connect the MBC arrays and conduct the topological reconfiguration of RENO. To receive command and data and send result back to processor in digital form, input, output and configuration FIFOs are located at the interface of RENO.

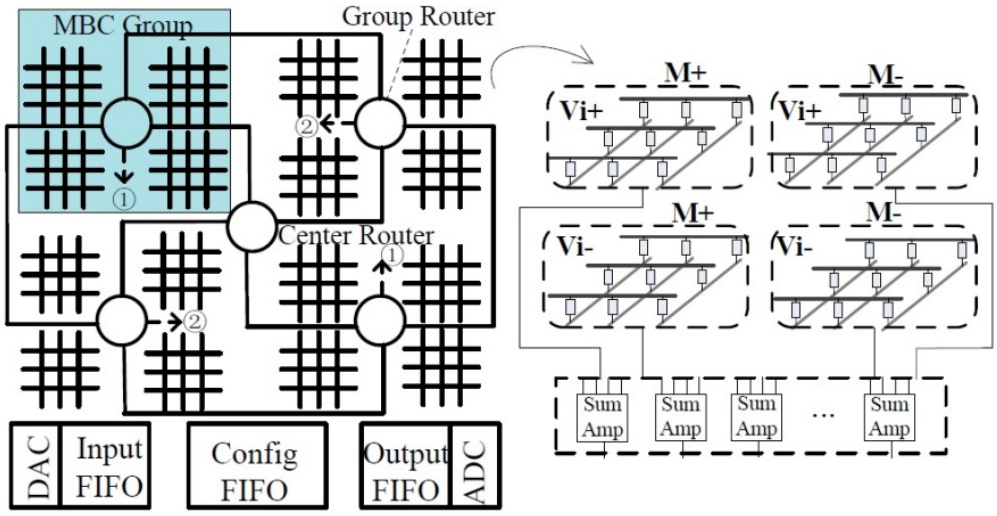

Figure 23 RENO architecture.

3.2.1 Hierarchical structure of MBC arrays

The hierarchical MBC array structure adopted in the RENO is depicted in Figure 23. MBC arrays are arranged in a metamorphous centralized mesh (MCMesh) manner to minimize the cost of the interconnection network [70]. A RENO is composed of four array groups, each of which is formed with four MBC arrays connected through a group router. Here an MBC array is partitioned into four sub-crossbars to implement the multiplication of the combination of the signed signals and the signed synaptic weights. In this work, the optimized MBC design has 64

rows and 64 columns. As we shall show in Section 6.4, such a design offers a good compromise between performance and reliability. Moreover, it covers the majority of learning applications where 80% of them have less than 60 neurons in the input layer [52]. Applications requiring larger connection matrices can be partitioned into smaller tasks and executed on multiple MBC arrays simultaneously or sequentially.

Without losing generality, we use a connection matrix $M_{n\times m}$ as an example to explain how to map a connection matrix to the MBC arrays. Here n and m denote the numbers of neurons in the input and output layers of the connection matrix, respectively. If $\max(n,m) \leq 64$, $M_{n\times m}$ can be directly mapped to a 64×64 MBC array; if $64 < n \leq 128$ and $m \leq 64$ or if $64 < m \leq 128$ and $n \leq 64$, $M_{n\times m}$ can be mapped to two MBC arrays; an even larger $M_{n\times m}$ need be partitioned into more MBC arrays and mapped into different MBC groups.

3.2.2 Mixed-signal interconnection network (M-Net)

3.2.2.1 Digital, analog, or mixed-signal?

The signal transmission within a RENO can be realized in either digital or analog form. Digital signal transfer has good controllability and supports high-frequency operations. However, as the computation of MBC arrays is in analog form, DA/AD conversions are required at the interface of MBC arrays and routers, which inevitably degrades the signal precision and results in significant area and power overheads. The small footprint of the MBC arrays limits the data communication distance, e.g., within 0:53mm, making it possible to transfer the computational signals in analog form. Moreover, the impact of signal distortion generated during the analog signal transmission on computation reliability can be tolerated by the intrinsic high fault resistance of ANN algorithms.

We propose a mixed-signal interconnection network called M-Net to assist the task mapping and data migration in the MBC arrays. M-Net maintains the data in analog form while transfers the control and routing information in digital form so as to simplify the synchronization and communication between CPU and routers. More specific, the signal communication is fully conducted through routers, each of which is divided into digital control logic and analog data path.

Figure 23 also shows the centralized hierarchical MBC array architecture where the data communication is performed at both inter-group and intra-group levels. The central router connects to the CPU and all the group routers. Each group router talks to the four local MBC arrays within the group, three other group routers, and the central router. Such a centralized scheme maximizes the number of parties that each router communicates with, minimizes the effective communication distance and the hop count, mitigates the bottleneck effect of the central router, and simplifies the control complexity.

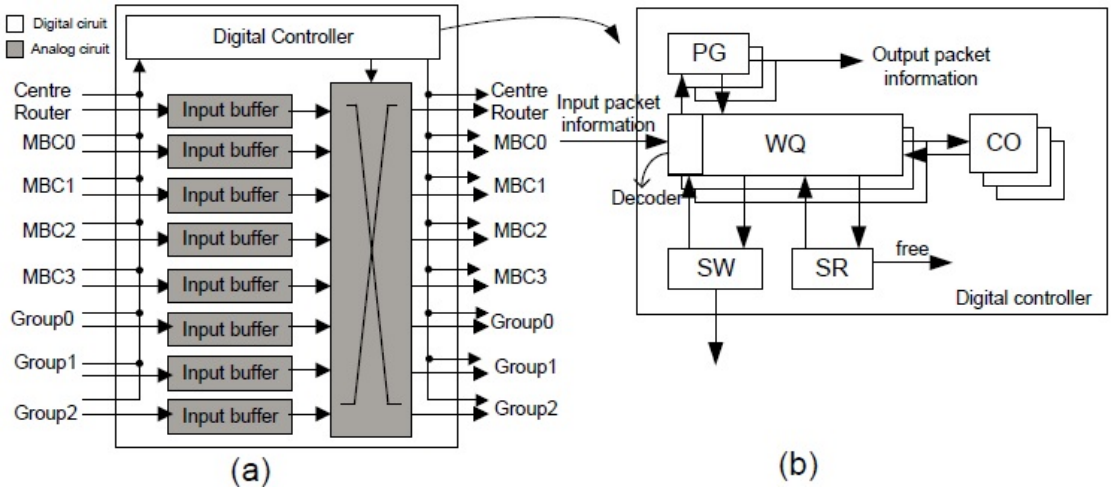

Figure 24 The mixed-signal router design: (a) architecture; (b) the digital controller.

3.2.2.2 Router design

Figure 24(a) shows the group router design. Its analog data path consists of input buffers and data multiplexer/switches. Each input port can receive up to 64 analog signals corresponding to a set of the inputs/outputs of a MBC array, referred as a packet. During RENO operations, a switched-op-amp (SOP) based sample-and-hold (S/H) circuit (see Figure 24(a) [72]) serves as an analog buffer, which holds and passes the analog data to the next destined MBC array or router. The S/H circuit adopts a pseudo-differential topology and turns off the transistor in saturation region. Such a design substantially minimizes the nonlinear distortion of the stored analog data caused by charge injection and clock feedthrough error, maintaining a good signal quality [72].

Figure 24(b) depicts the conceptual implementation of a 8×8 multiplexer based on transmission-gate based analog crossbar switches. The multiplexer can dynamically establish the routing path from one input port to one output port under the guidance of the digital control logic. 64 copies of such a multiplexer are required at each port of a group router to transmit a packet of up to 64 signals simultaneously.

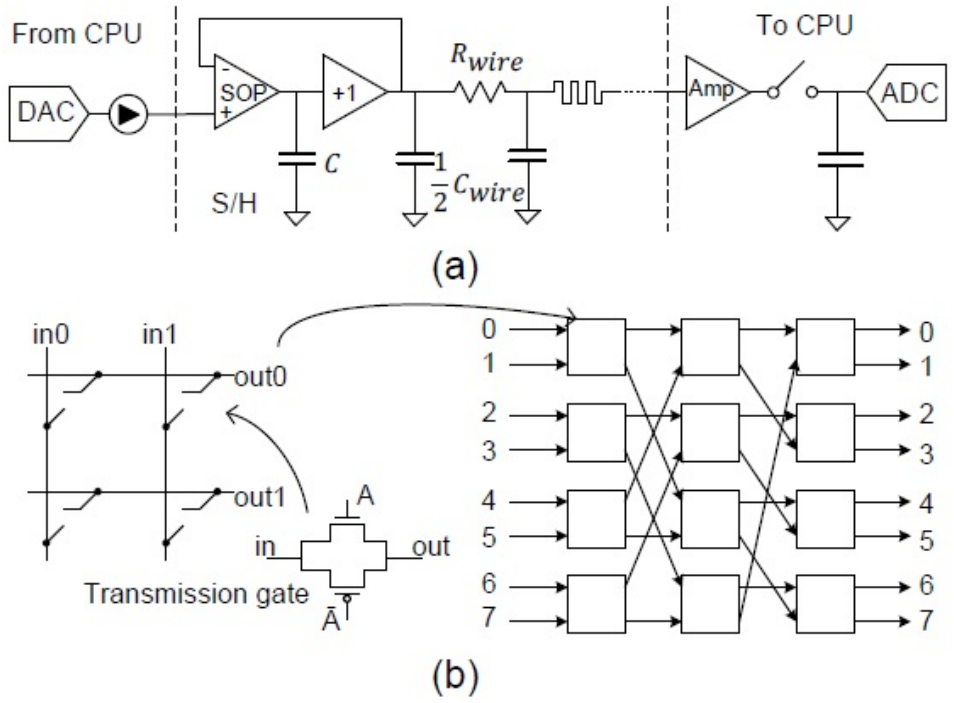

Figure 25. The analog component design in the mixed-signal router: (a) the transmission path; (b) the crossbar-based multiplexer.

The digital controller of a router is shown in Figure 25(b). The routers in the RENO are responsible for not only data transferring as traditional Network-on-Chip (NoC) does, but also routing information processing. Thus, a work queue (WQ) is introduced. Once the WQ receives the routing information of a data packet, it will decode the information to generate the control signals of other components in the router. The routing path configuration in the multiplexer is controlled through a switch allocator (SA). Each WQ entry is associated with a multi-bit computing counter (CO) to monitor the computation status of a local MBC array by counting the number of the executed loops. In this work, we utilize a 7-bit CO (supporting up to 128 loops) because all the selected benchmarks can complete executions within 100 loops. As a local MBC array approaches to the end of its computation, the CO notifies its WQ. The computation result

will be sent to the CPU or another MBC group with the routing information generated by packet generator (PG). At this time, the corresponding WQ entry is released and the updated routing information remains in the group router. Status recorder (SR) logs and broadcasts the availability of a local MBC array to all the connected routers.

The central router design is similar to that of the group router except that the central router is only responsible for establishing the data paths between the CPU and the four group routers. Although the group routers work independently, all the MBC arrays can perform computation simultaneously.

3.2.2.3 Routing management

Figure 26 shows the format of the routing information adopted in RENO, including 1-bit valid bit (V), 1-bit routing field (H), address field (Addri), and looping field (Loop). An address field contains 5 bits: Addri [1:0] identifies the group router, Addri[3:2] denotes a MBC array within the group, and Addri [4] indicates if the data shall be sent back to CPU. Corresponding to the CO design, the looping field contains 7 bits supporting up to 128 loops.

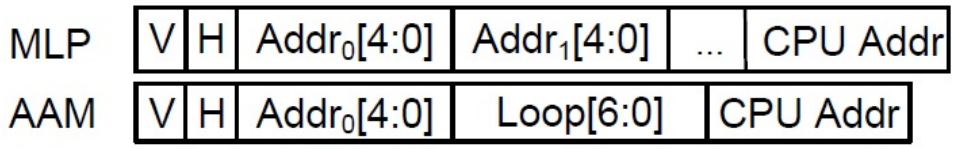

Figure 26. **Routing information format.**

Bit H represents the type of ANN implementations (MLP or AAM) and determines the format of routing information. The MLP configuration does not require looping field. The address fields of MLP include the addresses of the MBC arrays that the data will go through and the address representing the CPU. The AAM configuration needs both address and looping fields to guide the destined router address and the related number of computation loops, respectively. A routing information always ends at CPU address, indicating the completion of data transmission. Once an input data packet goes through the corresponding router, these address and looping fields can be recycled by the PG.

3.3 EXPERIMENTAL METHDOLOGY

3.3.1 Circuit level implementation and simulation

We created a Verilog-A memristor model by adopting the device parameters from [64] and scaling them to 65nm node based on the resistance and device area relation revealed by the physical experiments in [73]. All the RENO circuit components, including MBC, analog buffer, switch, sum amplifier, and sigmoid circuit, are designed with SMIC 65nm technology [74]. To achieve high-speed and small form factor, we adopt the flash analog-digital converter (ADC) and current steering digital-analog converter (DAC) [75] in our design. The resolution of DAC/ADC is set to 4-bit to comply with the data resolution required by the selected benchmarks. As depicted in Figure 23(a), a MBC array receives input data from the DAC and sends the computation result to the ADC. The DAC at the input side is coupled with a cascoded current to boost the output impedance. At the output side of the MBC array, a signal passes through an

amplifier (Amp) and a sample-and-hold (S/H) before reaching the ADC. The Amp boosts up the input signal to match with the ADC input window and performs correlated-double-sampling (CDS) to mitigate the DC offset caused by mismatch [76], while the S/H ensures a stable input during the analog-to-digital conversion. We estimate the delay and power of all these components and extract the layout areas under Cadence Virtuoso environment [77]. The detailed design parameters and area estimation can be found in Table 7.

The area of a RENO is mainly occupied by the routers. An analog signal transmission model is created to simulate the analog signal transmission in the concerned distance, e.g., among the routers. Our simulation shows that a voltage swing between 0V and 1V can be transferred from one end of an interconnection of 0.53mm to the other end in 0.5ns, after considering signal fluctuations.

Table 7. The simulation platforms

Memristor					
R_L=200Ω, R_H=160KΩ, V_{th}=2V					
MBC Array & M-Net					
	Neuron logic	Network	MBC	DAC	ADC
Power	126μW	0.88μW	0.72μW	5.2mW	3.8mW
Speed	0.93ns	4.9ns	3.1ns	333MHz	333MHz
Area Estimation					
	RENO area (mm^2)	NoC (mm^2)			DAC/ADC (mm^2)	MBC (mm^2)
		Input/output	Channel	Control		
M-Net	0.943	0.598	0.014	0.252	0.072	0.007
D-Net	1.793	0.268	0.065	0.301	1.152	0.007

All the major noise resources, including 1= f noise in amplifier, thermal noise produced by memristor and amplifier, and quantization noise caused by ADC, have been evaluated. The result shows that the quantization noise up to 18mV dominates the overall noise while the other

two noises are negligible. Such noise magnitude is much smaller than the resolution of 4-bit DAC/ADC (62.5mV) so that the introduced impact remains under a tolerable level in RENO operations. Finally, the device mismatch can be calibrated by a predetermined look-up table [78].

Reliability analysis is conducted using Monte-Carlo simulations. We assume both the resistance of the memristors and the analog inputs of the MBCs follow normal distributions. In each Monte-Carlo simulation, the initial memristor resistance of a MBC sample is fixed as it is decided by the offline training. Instead, the signal fluctuation is generated on-the-fly during the entire RENO execution.

3.3.2 Benchmarks

We choose seven representative learning benchmarks in our evaluations, as summarized in Table 8. Cancer, gene, mushroom and thyroid are selected from Proben1 [79]. connect-4 and lymphography from UCI machine learning repository [80] are tailored for neural network implementation. MNIST [81] is a widely-used benchmark of learning and recognition algorithms1.

We implement all the selected benchmarks by using MLP and AAM models and measure the execution quality in classification rate. These benchmarks naturally come with training and testing inputs, and we further divide the training vector into an actual training set and a so-called validation set which is used to evaluate the quality of a network.

ANN topology for each application is optimized based on FANN library [82] by comprising training time, computation accuracy, and network size. The enhanced device variation and signal noise aware MBC training scheme [54] is utilized to ensure training robustness. We define training error as the mean square error (MSE) between the actual and

target outputs under the training vectors. Table 8 summarizes the implementation details and the initial training errors.

Table 8. The Description and Implementation Details of the Seven Selected Benchmarks

Benchmark	Description	MLP			AAM	
		Training error	Topology	MBC array usage	Training error	MBC array usage
cancer	breast cancer diagnose	0.02%	36→16→2	2 arrays in 1 group	0.07%	2 arrays in 1 group
connect-4	connect-4 game	0.02%	42→30→3	2 arrays in 1 group	0.08%	3 arrays in 1 group
gene	nucleotide sequences detection	0.09%	120→100→3	6 arrays in 2 groups	0.03%	12 arrays in 3 groups
lymphography	lymph diagnose	0.05%	29→19→4	2 arrays in 1 group	0.02%	4 arrays in 1 group
MNIST	digit recognition	0.35%	64→128→32→10	5 arrays in 2 groups	0.02%	10 arrays in 3 groups
mushroom	poisonous mushroom discrimination	0.01%	125→32→2	3 arrays in 1 group	0.01%	8 arrays in 2 groups
thyroid	thyroid diagnose	0.15%	21→32→3	2 arrays in 1 group	0.11%	3 arrays in 1 group

3.3.3 Architecture level simulation setup

We modify MacSim [83], a PIN-based [84] cycle-level X86 simulator, by adding a cycle-accurate RENO module to conduct architecture level evaluations. The CPU is configured as an Intel Atom [85]-liked processor. The RENO-supported functions within a target code shall be identified and translated to RENO instructions. During trace generation, the modified PIN tool generates the simulation trace by replacing the RENO-supported functions with the corresponding RENO instructions according to the selected ANN topology in the specific application. Since all the selected benchmarks are ANN oriented, on average, 99% of execution time is consumed on running the target codes. Thus, in the following evaluations, the execution time of the target codes is used to represent the overall performance. Table 7 summarizes the parameters of our simulation platform.

Table 9. The Simulation Platforms

CPU	***CPU core***	1.1GHz, OOO 2-issue(up to 1 mem and 1 FP), 20-stage pipeline, gshare branch predictor, 256-entry BTB, global history length 14, 20-cycle branch misprediction penalty, 256-enrty ROB
	Cache & Memory	32KB L1ICache, 4-way, 1 bank, 4-port, 1-cycle latency, 64B line 32 KB L1DCache, 4-way, 4-bank, 4-port, 2-cycle latency, write back, 64B line 256 KB L2, 8-way, 4-bank, 1 R/W port, 8-cycle latency, write back, 64B line, 2 MB L3, 16-way, 8-bank, 1 R/W port, 16-cycle latency, 64B line, write back, 128 MSHRs 4GB main memory, fixed 50-cycle latency
RENO	***Computing Units***	4 MBC groups, 4 MBC arrays/group, 4 MBC/array, MBC size: 64×64
	IO	64×4-bit In-queue/Out-queue, 128×64-bit Config-queue, 64 parallel DACs, 64 parallel ADCs
	Network (M-Net)	Mixed-signal CMesh, 333 MHz for digital control, 4 group routers, 1 central router
D-NPU	***Computing Units***	16 digital PEs, Input/Output Buffer 64×4-bit, Weight Cache 4096×4-bit, Sigmoid Unit LUT 512×4-bit, 4-bit Multiply-add unit
	IO	64×4-bit In-queue/Out-queue, 128×64-bit Config-queue
	Network (D-Net)	Digital CMesh, 1.332 MHz, 64-bit datapath, 4 group routers, 1 central router

The energy consumption of the CPU core is estimated using McPAT [86]. We generate a detailed log of RENO utilization during the execution so that the energy consumption of the RENO can be calculated based on the characterized results from our circuit level simulations. The data traffic and the power consumption of the M-Net within the RENO are simulated by a modified Booksim simulator [87].

3.3.4 Implementation of other design alternatives

We also explore the potential of RENO by comparing with other ANN accelerator designs. First, we construct a digital neural processing unit (D-NPU) which adopts the RENO topology but replaces MBC arrays with digital processing elements (PEs) [68], as shown in Figure 27. Accordingly, the interconnect network is designed in digital format (namely, D-Net). To perform a fair comparison, the input/output FIFO and weight cache of each PE are scaled up to match the computational capacity of a MBC array. The latency and power of a PE are extracted from a

Verilog-HDL model synthesized with SMIC 65nm library using VCS and Design Compiler. The detailed D-NPU configuration is summarized in Table 7.

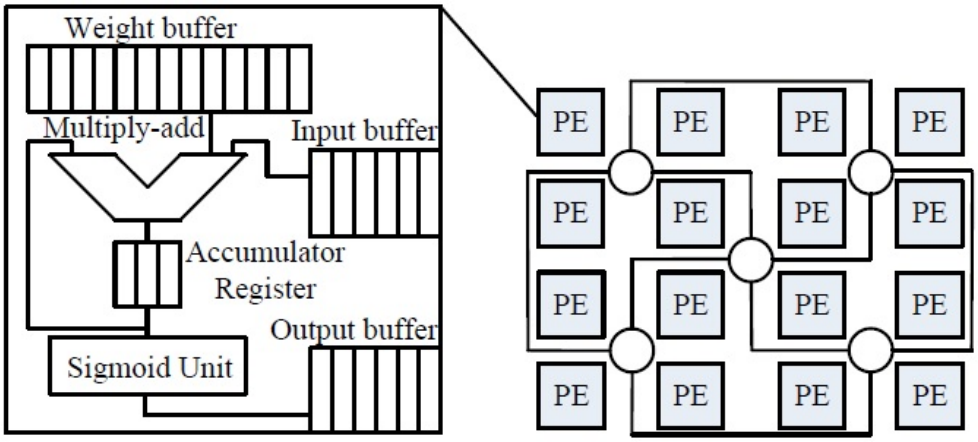

Figure 27. A D-NPU design built with digital PEs in [68].

To study the efficacy of M-Net, we construct an alternative design by solely replacing the M-Net in RENO with D-Net. The MBC arrays remain as the computing units. D-Net keeps the same topology and function as M-Net by transmitting both data and control signals between CPU and MBC arrays in digital format. To minimize the design cost of data bus while maintaining the same bandwidth, digital data can be packed and transmitted at a higher frequency. The evaluation in Booksim [87] shows that operating the D-Net with input buffers at 1.332GHz offers the similar transmission capacity as M-Net. Essentially, the boundary of digital and analog domains moves from CPU↔RENO to D-Net↔MBC arrays in such a "MBCs+D-Net" design. In other words, DAC/ADC pairs are required at the interface of each router and frequent DA/AD conversions before/after any MBC-based computation are indispensable. Compared to M-Net, digital transmission on D-Net suppresses signal precision loss and simplifies router design.

However, the increased number of DA/AD converters dramatically increases the design area and power consumption overheads of the non-computing parts, as illustrated in Table 5.

3.4 EXPERIMENTAL RESULTS

We investigate the design and optimization of RENO by thoroughly evaluating the impacts of MBC training effort, device variations and signal fluctuations, MBC array size, and memristor resistance shifting. The potentials of RENO from the perspectives of computation accuracy, performance, and energy consumption are also comprehensively explored by comparing to the general-purpose CPU and two other ANN accelerators: D-NPU and MBCs+D-Net.

3.4.1 MBC training effort

The computation accuracy and energy consumption of the RENO are greatly influenced by the precision of the input signal and the training effort which can be measured by the size of training data set used in MBC array training. The resolutions of the computation data are naturally provided in the selected benchmarks, which are all less than or equal to 4-bit. Thus, we fix the DAC/ADC resolution to 4-bit and focus on the impact of the training effort in the following evaluations.

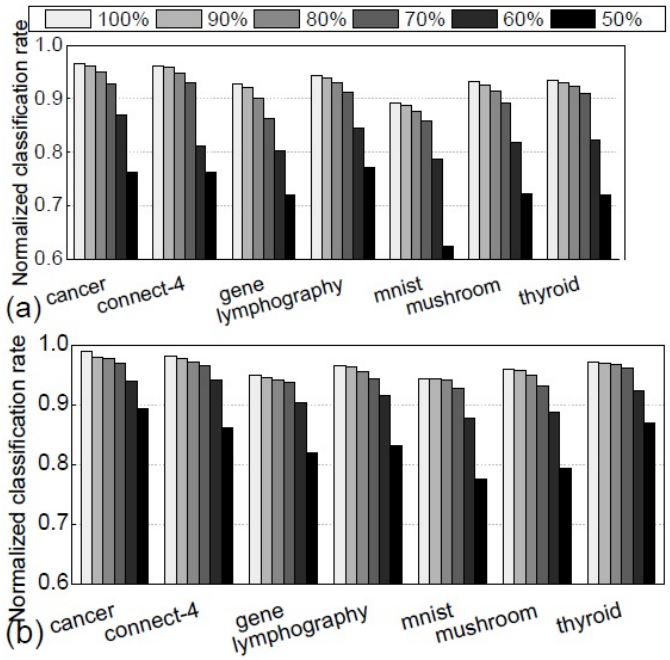

Figure 28. The normalized classification rates of (a) MLP and (b) AAM under different MBC training efforts. The DAC/ADC resolution is set to 4-bit.

For a specific benchmark, the computation accuracy can be improved by increasing the size of training data set. However, the computation accuracy will saturate to level when the number of the training data reaches a threshold, i.e., the saturated training data set size. Figure 28. The normalized classification rates of (a) MLP and (b) AAM under different MBC training efforts. The DAC/ADC resolution is set to 4-bit. Figure 28 (a) and (b) respectively compare the computation accuracy (i.e., the classification rate) degradations of MLP and AAM implementations under different training efforts. The classification rates have been normalized to the ideal case that the execution is performed by the floating-point unit of the CPU. Here the training effort is normalized to the saturated training data set size of each benchmark. Applying 100% training effort will produce a normalized classification rate very close to the ideal case. The classification rate decreases as the training effort reduces due to the degraded training

accuracy. Generally, the MLP implementation is more sensitive to the variation of training effort. In particular, gene, mnist and mushroom experience considerable reduction in classification rate due to their large network scale. When the training effort is set to 70%, the normalized classification rate is maintained above 86% for all benchmarks. Further reducing the size of training data set will quickly deteriorate the RENO computation accuracy.

Benefiting from the iterative feedback loop, the AAM implementation demonstrates much better computation accuracy than the MLP implementation under the same training accuracy. For the AAM implementations, the average classification rate of all benchmarks keep above 83% even the training effort is as low as 50%. In the following evaluations, we set a training effort of 70% which can simultaneously satisfy the computation accuracy requirements of both MLP and AAM implementations with reasonable hardware and performance overheads of training.

3.4.2 Impact of device variations and signal fluctuations

Figure 29 illustrates the impacts of device variations and signal fluctuations on the computation accuracy of RENOs. Here σ_p denotes the standard deviation of memristor resistance incurred by process variations; sf denotes the standard deviation of the magnitude of the analog signals generated from DA/AD conversion, routing/buffering, sum-amplifier and sigmoid function. Since σ_f has greater impact on the computation accuracy of MBCs than σ_p [54], we choose very pessimistic settings of sf in our simulations to cover even the very extreme cases.

As expected, the increase of device variations and signal fluctuations generally degrades the computation accuracy of the RENO with both MLP and AAM implementations. Interestingly, the normalized classification rate of mnist degrades slightly faster than other

benchmarks, indicating a less robust ANN topology. Nonetheless, both MLP and AAM implementations maintain a very moderate computation accuracy deterioration when σ_p and σ_f are within a realistic range, i.e., σ_p =0.05 and σ_f =0.1. Again, the AAM implementation demonstrates better tolerance to process variations and signal fluctuations than the MLP. We note that after this section, all the simulations are performed by considering only a nominal case. However, the statistical analysis can be easily conducted by following the same flow that generates Figure 28.

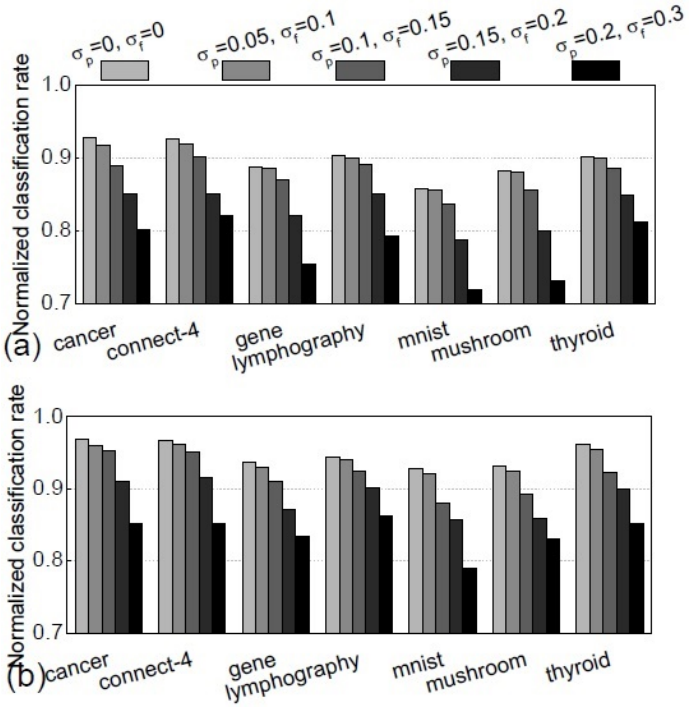

Figure 29. The impact of device variations and signal fluctuations on computation accuracy: (a) MLP, (b) AAM.

3.4.3 Impact of MBC sizes

On one hand, increasing MBC size improves the computation efficiency of the RENO as more calculations can be performed simultaneously. It also helps to reduce the overheads of the

computation partitioning and the signal routing among MBCs if the scale of the ANN topology is larger than the MBC size. On the other hand, a larger MBC is more vulnerable to process variations and signal fluctuations, resulting in a worse programming quality. Moreover, when the MBC size exceeds the scale of the ANN topology, part of power consumption and computation capacity of the RENO will be wasted.

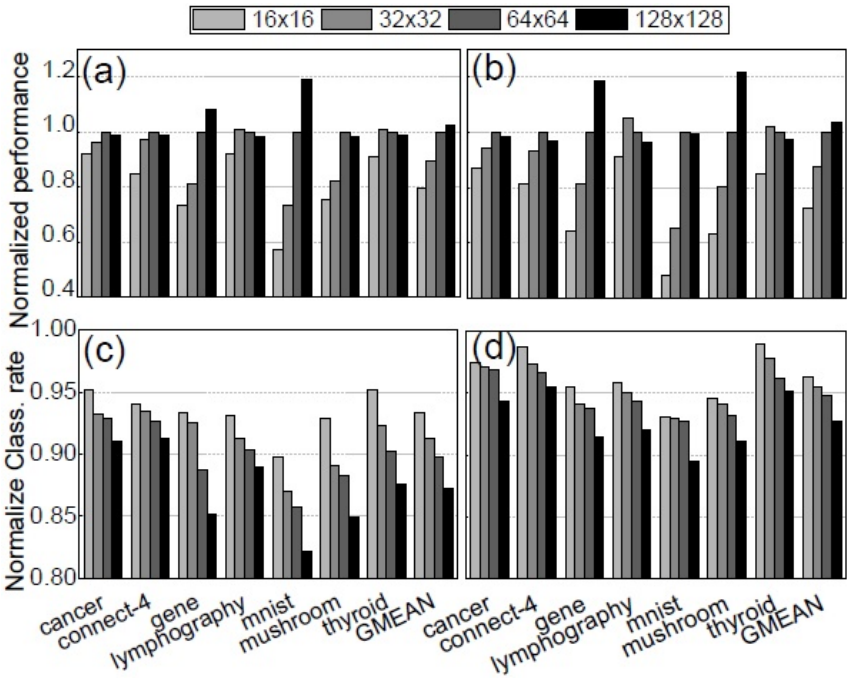

Figure 30. **The normalized RENO performance at different MBC sizes in (a) MLP and (b) AAM implementations. The results of 64x64 MBC is used as normalization baseline. The classification rate at different MBC sizes in (c) MLP and (d) AAM.**

In Figure 30, we compare the execution time and the classification rate of all benchmarks when the MBC size varies from 16x16 to 128x128. For a fair comparison, we keep the same computation capacity under all simulated MBC size configurations and adjust the routing

topology accordingly. For example, when 32x32 MBC is used, a RENO contains 64 MBC arrays and extends M-Net to connect all the arrays accordingly. Here sp and sf are set to 0.05 and 0.1, respectively. As the MBC size increases, the RENO performance of a particular benchmark keeps improving until the MBC size exceeds the largest scale of the ANN topology. Therefore, continuing to increase the MBC size does not further enhance the RENO performance. Nonetheless, the aggravated vulnerability of the RENO to process variations and signal fluctuations at a large MBC size causes slight degradation on the classification rate, as shown in Figure 30(c,d). Thus, in this work, we selected 64x64 MBCs as the optimized configuration that offers the balanced computation efficiency and accuracy.

3.4.4 Comparison to other design alternatives

Figure 31 compares the performance, energy efficiency, and classification rate of three ANN accelerator designs: D-NPU, MBC+D-Net, and RENO. Here, the energy efficiency is defined as the inverse of system energy consumption. The performance and energy efficiency are normalized to those obtained from the baseline CPU execution, which is, running the MLP/AAM implementation exclusively on the CPU based on FANN library. The results show that all the three ANN accelerators dramatically speedup the execution of the selected ANN benchmarks with slight degradation in computation accuracy compared to the baseline CPU.

As shown in Figure 31 (a,b), the geometric mean speedup (GMS) achieved by D-NPU in digital format is 11.9x or 1.7x for MLP or AAM implementation, respectively. As a PE can process only one multiply-add operation per cycle, the computation bandwidth of D-NPU is relatively limited compared to the other two designs. MBCs+D-Nets utilizes MBC arrays for analog computation, which dramatically boosts the GMS of its MLP and AAM implementations

to 117.2x and 20.1x, respectively. Compared with MBCs+D-Nets, the proposed RENO minimizes the costly DA/AD conversions and hence demonstrates even higher speedup: The corresponding GMS values further rise to 178.41x (MLP) and 27.06x (AAM). Relatively speaking, the AAM implementations obtain less speedup due to the costly iterations during the computation. The MLP implementations, however, achieve much faster execution because all the inputs traverse the network only once.

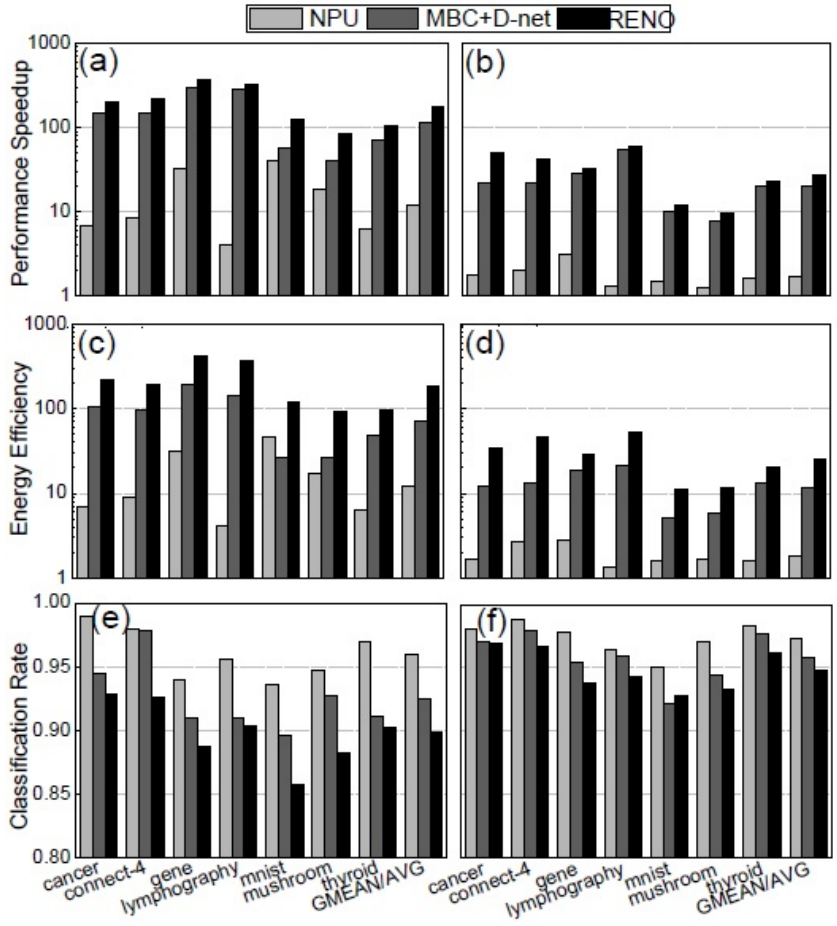

Figure 31. The performance speedup, energy efficiency and classification rate of three ANN accelerator designs with MLP (a,c,e) and AAM (b,d,f) implementations

Figure 31 (c,d) compare the energy efficiency result of each design, which demonstrates a trend very similar trend in the performance results. Compared to the baseline CPU architecture, MLP and AMM implementations of RENO achieve on average 184.24x and 25.23x energy savings, respectively. The energy efficiency of RENO is more than 2x higher than that of MBC+D-Net due to the dramatically reduced DA/AD conversion overhead. Note that in MLP, the energy efficiency of D-NPU under MNIST is higher than that of MBCs+D-Net. It is because the partitioning of MNIST onto multiple MBCs introduces considerably large amount of data traffic among the MBCs and hence, significantly raises the AD/DA energy consumption in MBCs+D-Net.

Figure 31 (e,f) compare the classification rate of each designs. In MLP implementations, RENO demonstrates the lowest computation accuracy. The computation accuracy is enhanced in MBC+D-Net design by utilizing digital network for signal transmission. As expected, the full digital implementation of D-NPU achieves the highest classification rate in all benchmarks. In AAM implementations, the three designs all obtain very high (i.e., 92%) classification rate in all benchmarks. The classification rates achieved in each design are also very close, say, with a variation less than 2.8%. This is because AAM can automatically compensate the adverse impact of the less reliable executions in each loop on the computation accuracy, by paying the cost of more iterations. As shown in Figure 31(a,b), compared to MBC+D-Net, the performance speedup achieved by RENO in the AAM implementation is only 1.3x, which is less than the 1.5x speedup achieved in the MLP implementation. In short, RENO exhibits extremely high performance and power efficiency while well maintaining the computation accuracy within an acceptable level. Moreover, MLP and AAM implementations present different tradeoffs between the computation efficiency and accuracy, offering valuable design flexibility adaptive to the

nature of the particular applications. For example, when using application (mushroom) to discriminate the poisonous mushroom, a user may be willing to tolerate the high computation cost of the AAM implementation to achieve a more confident result. In contrast, MLP could be a better choice when implementing (connect-4) because the response time is more critical to the player.

3.5 NOC CHALLENGES IN NEUROMORPHIC ACCELERATION SYSTEM

3.5.1 Background and motivation

3.5.1.1 Neural networks (NN) and neuromorphic acceleration system

In this work, we mainly focus on Multilayer Perceptron (MLP), which is a feedforward artificial neural network that widely utilized in classification algorithms such as the classification layer in deep neural networks (DNN) [95] and convolutional neural networks (CNN)[96], and approximate computing[50]. MLP presents not only the basic computation patterns of DNN, i.e., matrix multiplication followed by nonlinear activation functions (e.g., sigmoid etc.) at each layer, but also the intensive communications within the layers. Figure 32 depicts the hardware utilization of Alexnet [97] running on Nvidia GeForce GTX TITAN X GPU [98] where MLP processes the largest memory-to-computing ratio. As such a memory-to-computing ratio directly links to the traffic intensity of the NoC on a neuromorphic system, we choose MLP as the target in our study: the performance of MLP on each design reflects the worst-case efficacy of the NoC.

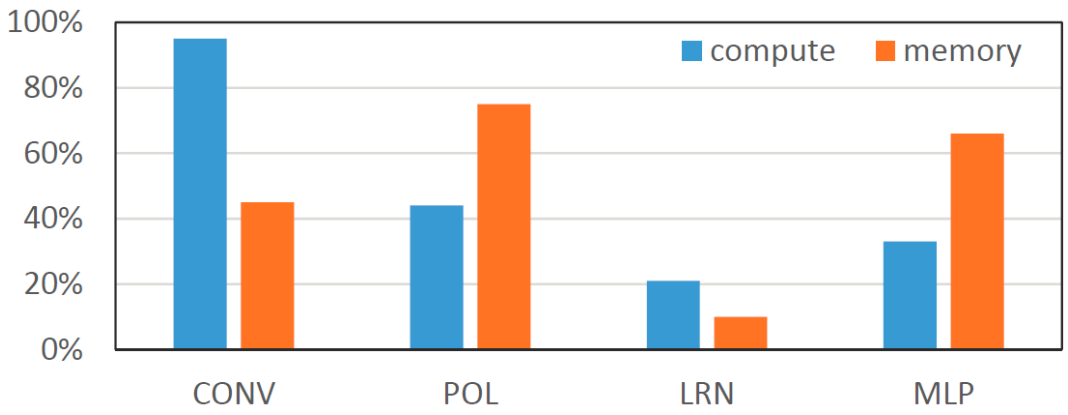

Figure 32. Hardware utilization of each layer in DNN.

Hardware acceleration for neural networks has been extensively studied on not only general-purpose platforms, e.g., graphic processing units (GPUs), but also domain-specific hardware such as field-programmable gate arrays (FPGAs) and custom chip (e.g., TrueNorth) [92]. On a neuromorphic computing system, data are stored in a distributed manner (i.e., the weights of synapses) and participates in the computation through local neurons. Although these architectures greatly mitigate the requirement of memory bandwidth, long-range connectivity across computation cores emerges as a new challenge in hardware development. For example, in IBM TrueNorth system, the local neuronal execution within a neurosynaptic core is extremely efficient, while the energy consumption of core-to-core communication increases rapidly with the distance between source and destination[100]: an inter-core data transmission could consume 224X energy of an intra-core one (i.e., 894pJ vs. 4pJ per spike per hop)[100]. As the scale and density of the NN increase, more inter-core interconnections are introduced; the effect of long-range connectivity and communication quickly becomes a severe design challenge.

Table 10. GPGPU-SIM configuration

Benchmark	Topology	mapping in PEs	NoC	
			Mesh	Neu_NoC
mnist_mlp_1	784-300-100-10	19-7-1	6x6	3x3
mnist_mlp_2	784-1000-500-10	63-32-1	10x10	4x4
mnist_mlp_3	784-1500-1000-500-10	94-63-32-1	14x14	5x5
mnist_mlp_4	784-2000-1500-1000-500-10	125-94-63-32-1	18x18	7x7
mnist_mlp_5	784-2500-2000-1500-1000-500-10	157-125-94-63-32-1	22x22	8x8
alexnet_cnn_cla	9216-4096-4096-1000	256-256-63	24x24	9x9

To overcome this challenge in NN acceleration, both hardware and software solutions are explored. The hardware approaches attempt to build a specialized circuit and/or architecture including some customized network models. These new models, however, are often inconsistent with their state-of-the-art version, resulting in low inference accuracy. In TrueNorth, for example, the NNs constructed in the tiled neurosynaptic cores could cause accuracy degeneration [99]. On the contrary, the software approaches mainly focus on reducing the scale and connectivity of DNN models while still retaining the accuracy [101]. However, implementing such pruned models on hardware can be very challenging. Such sparse NNs often generate scattered memory access patterns so that the realistic speedup can be very limited [102].

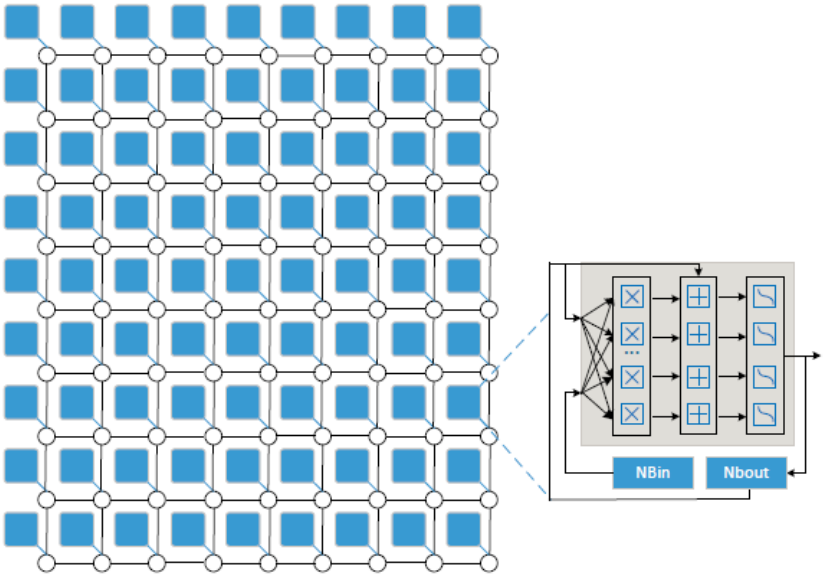

Figure 33. Baseline design of a neuromorphic computing system with NoC.

3.5.1.2 Motivation of our work

NoC is a critical part in neuromorphic computing system designs because their data tra_c patterns are significantly different from that in conventional multicore systems. Figure 33 depicts a neuromorphic computing system that composed of 648 processing engine (PE) [88]. A 9x9 Mesh NoC, which is the most widely used NoC design in multicore systems, connect these PEs and serve as our baseline. The weights are stored in the distributed local memories and fetched through a specific high-throughput memory bus within a short distance. The matching relations between neuron outputs and weights are stored as the source address information in head flit and also the sequential order of fits in a package.

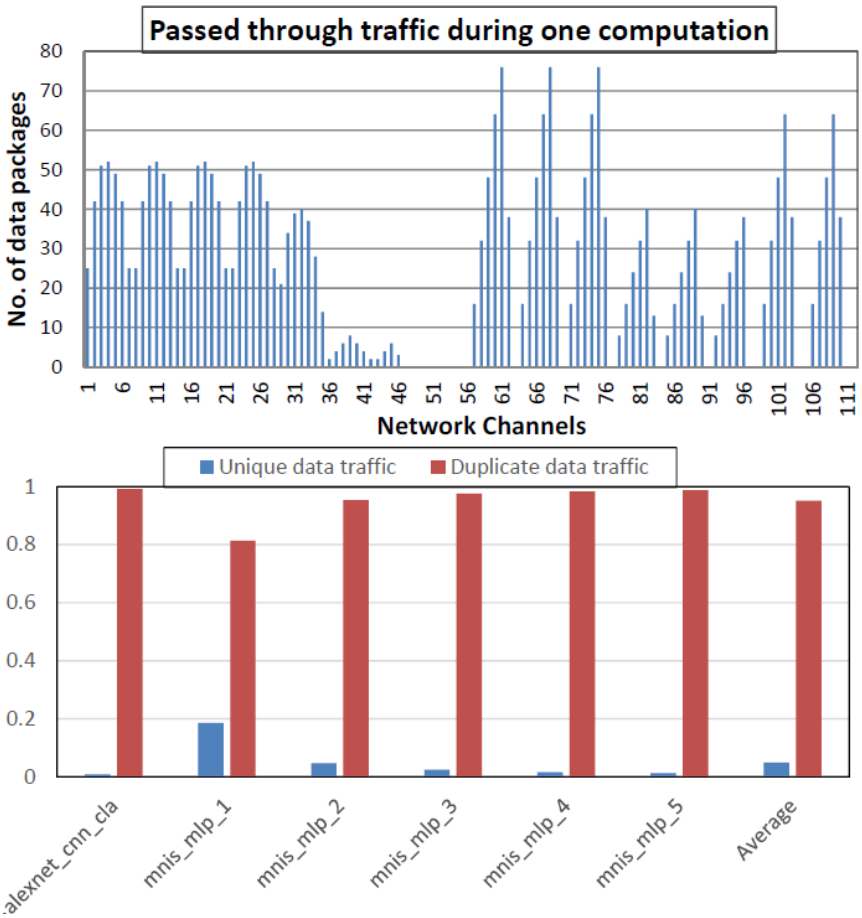

Figure 34. The volume of data packages transmitted over different channels; (b) The ratio of duplicated data packages.

In MLP, the neurons in one layer only communicate with the neurons in the next adjacent layer. As an initial study, we stimulate the data traffic of 6 selected NN benchmarks running on the NoC of the neuromorphic computing system depicted in Figure 33. More details about these benchmarks can be found in Table 11. Figure 34(a) shows the result of running mnist_mlp_1 that on average, 57.6% of the total data packages travel through only 29% of total channels during the NN computation. The data traffic is particularly concentrated on the transmission channels that connecting two adjacent layers while the channels between the neurons in one layer are idle

most of the time. Obviously such communication pattern leads to very unbalanced traffic patterns and causes congestions over the NoC.

In addition, every neuron in a layer of the NN sends the same value to its connected neurons in the next layer. It means that a node of a NoC could broadcast/send multiple packets containing the same data to a set of nodes. In our initial study about the 6 fully-connected networks, we compare the number of the packets with unique data and the number of the packets with the same data but sent to different destination nodes. The results in Figure 34(b) show that a significant number of packets are indeed used to deliver the same data to different nodes. Such traffic pattern is rare in a conventional computing model and offers a great improvement opportunity for the NoC design in neuromorphic computing systems.

3.5.2 Implementation of Neu-NoC

3.5.2.1 Hierarchical structure of Neu-NoC

Our initial analysis of the traffic patterns on traditional mesh NoC in neuromorphic systems inspired us to propose Neu-NoC -- an efficient NoC architecture that can suppress unnecessary data transfers of the same data and reduce the bandwidth consumption by consolidating neurons on the same neural network layer into local nodes.

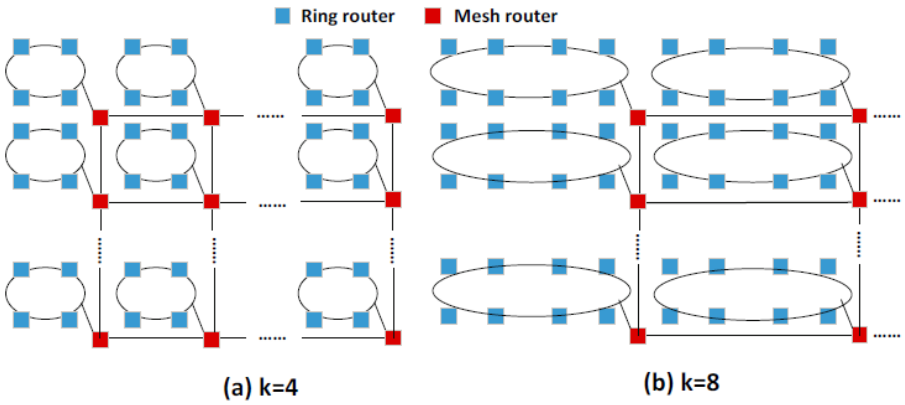

Figure 35. Different Neu NoC organizations of different configuration.

Figure 35 presents the overview of Neu-NoC architecture. The topology of Neu-NoC is a hybrid ring-mesh NoC structure. It consists of local rings and a global mesh that interconnects all the local rings. For convenience, here we inherit conventional notation like k-array n-cube [103] to describe the proposed hybrid ring-mesh NoC. k is the number of the nodes in a local ring; n and m represent the numbers of columns and rows of the global mesh, respectively. Figure 35 shows two configuration examples ($k = 4$ and $k = 8$) in a neuromorphic system with 64 PEs. In Neu-NoC, we use rings to cluster the neurons in the same layer to reduce the number of the data packages contenting the same data that need to be transferred: the neurons connected by one ring only send one copy of their data to the rings that connect the neurons in the next layer.

The local ring topology consists of two channels - a channel to receive data and a channel to pass the neuron output to the next layer, as shown in Figure 36(a). As a result, the output transferring of the neurons does not need to wait until the ring is idle. The traffic stalls are greatly reduced. Neu-NoC consists of two types of routers - a ring router connected to the PEs in each local ring and a mesh router connected to each local ring and other four mesh routers on its four neighbor directions. A block diagram of the router's microarchitecture is shown in Figure

36(b). The ring router consists of a 2-to-1 multiplexers, buffers, and function blocks for pack/unpacking data packages. Additionally, the ring router supports the arbitration with different priorities in Allocation function block, i.e., data packets in-fight always have a higher priority than the newly-injected packages. We adopt the typical mesh router design with Wormhole flit-based flow control for the global mesh network to maintain high edibility of mapping different NN topologies.

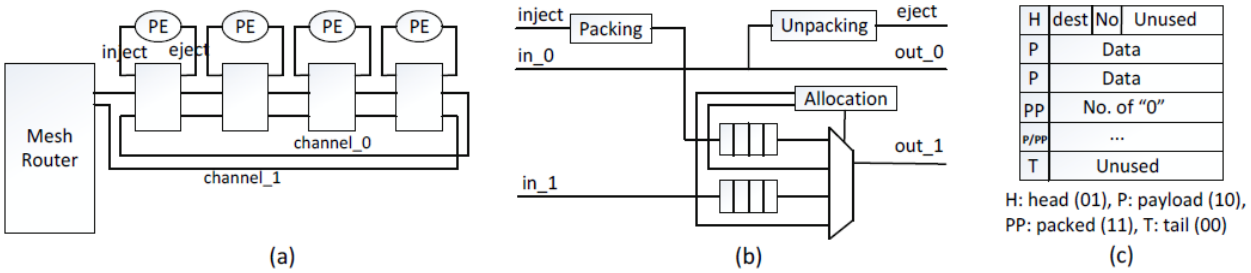

Figure 36. (a) Hierarchical Neu-NoC; (b) Ring router; (c) Package format.

3.5.2.2 NN-aware NoC mapping

a) Effect of NoC mapping

As the data transmission requirement (e.g., the source and destination neurons, amount of data) in a NN will not change during the computation, the data routing path is decided by NN-to-NoC mapping and routing algorithm. However, normal direct-mapping or random-mapping may not be optimal: if the connected neurons are allocated too far away from each other on the NoC, a high hop count may be introduced; also, if a large ratio of data packages go through the same

paths and follow the same direction, some of the paths may have serious congestion more likely than others and become bottleneck of data transmission.

As we use dimension order routing, only one possible path exists between each pair of the connected PEs. We assume that the row and the column numbers of the x^{th} PE are $row_{n,x}$ and $col_{n,x}$, respectively. Correspondingly, $row_{n+1,y}$ and $col_{n+1,y}$ are the row and the column numbers of the y_{th} connected PE in the adjacent layer. In a $M \times M$ mesh network that is mapped from a MLP, the average number of hops between one layer with P neurons and the adjacent layer with Q neurons ($(P+Q) < (M \times M)$) can be estimated by:

$$\boldsymbol{H_{avg}} = \frac{\sum(\boldsymbol{Vetical\ hops + Horizontal\ hops})}{\boldsymbol{Total\ number\ of\ paths}} = \frac{\sum_{y=1}^{Q}\sum_{x=1}^{P}|\boldsymbol{row_{n+1,y} - row_{n,x}}| + |\boldsymbol{col_{n+1,y} - col_{n,x}}|}{\boldsymbol{M^2(M-1)}} \quad (4)$$

The traffic load in each path can be measured by the number of the packages pass through during the NN computation, $Pack_{sum}$. If any of the paths has more packages need to pass than the others, it more likely become the bottleneck of the computation.

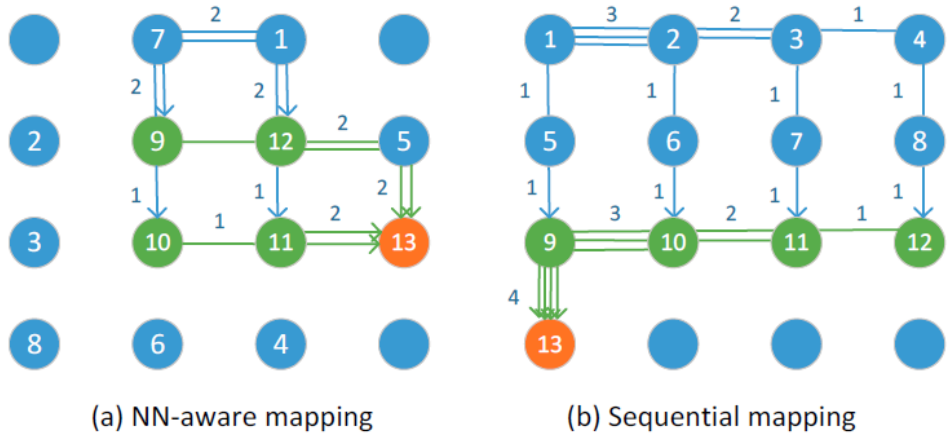

Figure 37. Different PE group placement in Neu-NoC for mnist_mlp_2. (a) NN-aware mapping, (b) Sequential mapping (MLP topology after mapping: 8-4-1).

Here we use an example to illustrate how different mapping schemes influences the average hop count and the congestion of the data traffic. Figure 37. Different PE group placement in Neu-NoC for mnist_mlp_2. (a) NN-aware mapping, (b) Sequential mapping (MLP topology after mapping: 8-4-1). Figure 37 shows the comparison between a random mapping and our proposed NN-aware mapping based on the NN topology. Here each square represents a local ring network and the number in each square labels the NN layers that the ring network resides on. The narrows show the path between routers, and the number on each narrow represents how many packages pass through the path at a moment of MLP computation. In direct or random mapping scheme, the layers are placed sequentially on the NoC, which may lead to a very high number of hops between the PEs in the adjacent layers. Our NN-ware mapping scheme, however, allows more PEs to access the PEs in adjacent layers with fewer hops.

To obtain the minimum hop counts and more balanced NoC, we design a NN-aware mapping which leads to the mapping solution with significantly reduced routing distance and distributed data traffic among the NoC. As the average distance of data transferring and tra_c congestions reduce, the average network latency will decrease too.

b) Problem formulation and definition

The problem of mapping a NN onto a NoC is a NP problem. We introduce the following definitions to help to formulate the problem that our proposed N-aware mapping algorithm is facing to find the optimal NoC mapping solution of the NN:

Definition 1: A Neural Network Communication Graph (NNCG) is a directed graph denoted by $G(N, A)$, in which each vertex n_i represent one neuron in the NN, and each directed

arch $a_{i,j}$ models a communication ow from one neuron n_i to one of its connected neurons n_j in the next NN layer.

Definition 2: An architecture characterization graph (ARCG) $G'(U, L)$ is also a directed graph that represents the physical NoC, where each vertex u_i denotes a node in the NoC and each edge l_i represents a physical link.

Definition 3: For an ARCG $G(U, L)$, a deterministic routing function R:$R \rightarrow P$ maps $r_{i,j}$ to one routing path $p_{i,j}$, where $p_{i,j} \in P_{i,j}$.

Based on these definitions, the problem of the NN-aware mapping can be formulated as follow: Give an NNCG and an ARCG, we need to find a mapping function $map()$ which satisfies:

$$\mathbf{min}\{\boldsymbol{T_{NoC}} = \frac{\sum_{y=1}^{Q}\sum_{x=1}^{P}|row_{n+1,y}-row_{n,x}|+|col_{n+1,y}-col_{n,x}|}{M^2(M-1)} + \boldsymbol{Pack_{sum}}\} \qquad \textbf{(5)}$$

such that:

$$\boldsymbol{3}\forall \boldsymbol{n_i} \in \boldsymbol{N}, \boldsymbol{map}(\boldsymbol{u_i}) \in \boldsymbol{U} \qquad (6)$$

$$\forall \boldsymbol{n_i} \neq \boldsymbol{n_i} \in \boldsymbol{N}, \boldsymbol{map}(\boldsymbol{u_i}) \neq \boldsymbol{map}(\boldsymbol{u_j}) \qquad (7)$$

$$\forall \boldsymbol{a_{i,j}} \in \boldsymbol{A}, \boldsymbol{p_{i,j}} \in \boldsymbol{P_{i,j}} \qquad (8)$$

The proposed algorithm is shown in Algorithm 1. The results of the hop count and max data load of the selected 6 benchmarks using the proposed NN-aware mapping algorithm are summarized in Algorithm 1, which is included and discussed in the next section.

Algorithm 1 NN-aware mapping algorithm

procedure NN-AWARE-MAPPING
input:
    $NNCG(N,A) \leftarrow$ NN communication graph
    $ARCG(U,L) \leftarrow$ architecture characterization graph
output:
    $map \leftarrow$ from$NNCG(N,A)$to$ARCG(U,L)$
    for $n_i \in NNCG(N,A)$ **do**
        map n_i to $u_x \in ARCG(U,L)$
        Generate map$(n_i) \in U$
        for each $a_{i,j}$ in A **do**
            $H \leftarrow \sum |row_i - row_j| + |col_i - col_j|$
            for each$p_{i,j} \in P_{i,j}$
            if $H \leq H_{min}$ **then**
                $H_{min} \leftarrow H$
                $map_{min} \leftarrow map(n_i)$
    for each $p_{i,j} \in P_{Hmin_{i,j}}$ **do**
        $l_{traffic} \leftarrow l_i + \cdots + l_j, l_i, \ldots, l_j \in p_{i,j}$
        if $l_{traffic} \geq l_{max}$ **then**
            $l_{max} \leftarrow l_{traffic}$
            recodel_{max} $foreach map(n_i)$
output $map(n_i)$ with smallest l_{max}

Algorithm 1. NN-aware mapping algorithm.

Table 11. The description and implementation details of the selected benchmarks

Benchmark	Accuracy	Accuracy drop 1%				Hops count		Max load		Packet count	
		4 0's	8 0's	16 0's	traffic	Random	NN	Random	NN	NN	Multicast
mnist_mlp_1	98%	43.3%	27.2%	4.9%	63.8%	12	4	3	1	-	-
mnist_mlp_2	98.27%	52.7%	33.1%	8.2%	55.8%	124	76	16	4	36	20
mnist_mlp_3	98.28%	49.7%	26.9%	4.4%	58.3%	550	421	24	14	132	87
mnist_mlp_4	98.32%	42.2%	22.0%	4.1%	53.2%	1774	1308	36	11	324	224
mnist_mlp_5	98.22%	43.6%	26.1%	6.7%	56.8%	3958	3074	27	16	644	419
alexnet_cnn_cla	56.60%	58.3%	36.1%	9.7%	48.3%	8218	7024	90	54	1260	563

c) Multicast transmission

As shown in Figure 37, some of the data packets sent from one neuron to its adjacent neurons in the next layer share the same routing path on NoC. For example, all the data packets that sent from router 1 to router 10 always go through router9 first and all the data packets sent from router 1 to router 11 always go through router 12 first. Since each neuron sends the same data to all its connected neurons in the next layer and the corresponding routing path is fixed due to xy routing topology, we design a multicast type of data transmission to combine the data packets from the same source node into one data packet, which will take same path during routing. For simplicity of the hardware, only the data packets that are completely contained by a other packet will be merged to the latter one. Table 11 shows the total number of the hops that all the generated data packets pass through in an NN computation before and after multicast transmission is applied.

To support the multicast transmission, we redesign the header of packet by introducing a bit string encoding scheme to carry multiple destination nodes and add a decoding and bit reset function in the router, as shown in Figure 38. In bit string encoding [8], each destination can be represented by only one bit. Figure 38 depicts an example of a multicast and its corresponding packet header. The length of the encode bit string in the header equals the number of nodes in the Mesh NoC; each bit represents a node and setting the bit to 1'b1 means that the node is one of the destinations. When a packet arrives at one of its destination, the bit corresponding to the destination will be reset to 1'b0. Note that here the packet itself carries all the routing information, which must be computed before the packet starts to transfer. Among all the selected benchmarks, the longest routing path is 16 links, which exists in *alexnet_cnn_cla*.

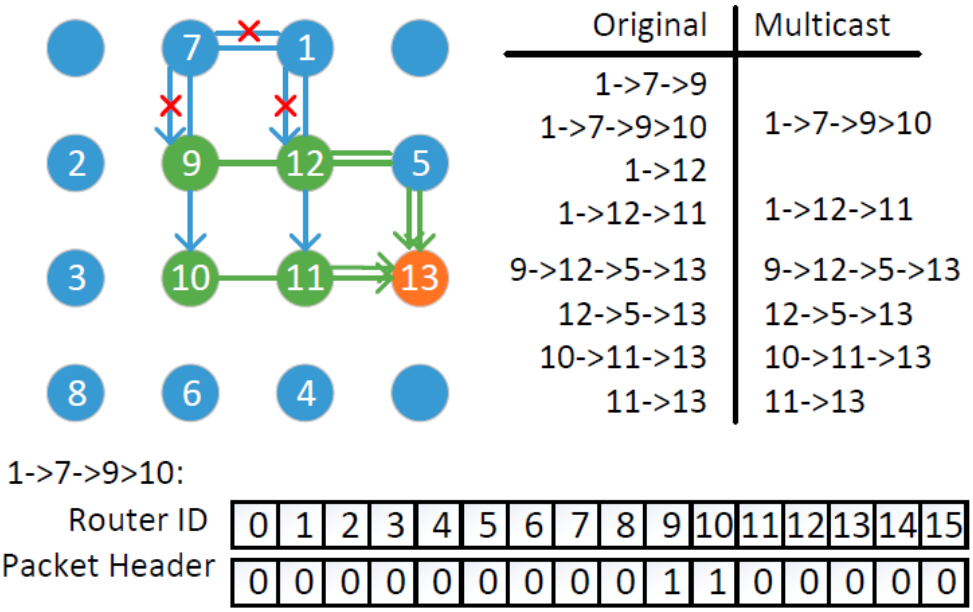

Figure 38. MLP topology after mapping: 8-4-1 Multicast and Packet Header Example.

3.5.2.3 Sparsity-aware traffic reduction

A DNN learns the object's features in a hierarchical way: concepts represented by the feature map of a DNN layer becomes more and more abstract when the layer goes deeper, e.g., from edges, shapes, object parts, to objects. As many prior-arts have proven, such highly abstracted feature map can be very sparse [104]. Very recently, pruning techniques [105] were also proposed to obtain sparse NN for computation cost reduction while still retaining the accuracy. The sparsity of the DNN can also help to reduce the traffic on the NoC.

We introduce a new type of it, namely, all-zero flit, to present a sequence of 0`s in the data. The all-zero it can be sent in any order between the head it and the tail flit in a packet. Such a scheme allow only one it indicate multiple sequential its of 0`s in the original it designs: We add one bit as a ag to denote if the packet is packed with all zero data, and use the payload area to denote the number of the continuous 0`s. We implement a function block at each input/output

port that connecting the mesh and the local ring to pack/unpack all-zero its, as shown in Figure 36. The original unpacked data only exists on the local ring bus in order to keep the bus interface simple and to reduce the buffer usage of the input/output virtual channel. In the packing block, we use a counter to detect the sequence of 0`s and a “pack" signal will be asserted if packing is needed. Similarly, in the unpacking block, as soon as an all-zero it is detected, the router will unpack the all-zero its and restore the sequence of 0's. The packing/unpacking blocks only introduce 3.1% area overhead to the router design.

Figure 36(c) depicts the format of a data packet. We add a new flit type – “PP” to represent the all-zero it in which the 2-bit head is 2'b11 and the “body" gives the number of 0`s. The formats of the other flits are similar to that in conventional wormhole flit-based flow control NoC.

3.5.3 Experimental methodology

In our experiments, we use MLP on MNIST database and the classification layer of *AlexNet* on ImageNet as our NN examples. Structures like those in [106] are also adopted in our MLP designs. The only modification is the dimensions of input images: we use the standard 28x28 images as the 784 input neurons instead of using the distorted 29x29 images in [106]. MLPs on MNIST are trained without data augmentation and AlexNet [97] is trained using Caffe. During the forwarding of the NNs, we zero out the activations propagated to the hidden layers when their absolute values are smaller than a predetermined threshold. We define the accuracy as the mean square error (MSE) between the actual and target outputs under the training vectors. Table 11 gives the benchmark information such as the implementation details, the initial training accuracy, and the accuracy and data traffic information after feature maps are pruned.

We modify *Booksim* [87] -- a cycle-accurate NoC simulator, by adding NN types of traffic module to mimic the data transfer in neuromorphic acceleration systems during the operations. Each Node is assumed to be a processing engine (PE) that has the design and computing ability similar to the one in [88]. A constant delay is added as the calculation delay before the output packages are generated when all the input data from the previous layer are collected. Table 12 summarizes the parameters of our simulation platforms. The energy consumption of the NoC is estimated by also *Booksim* with 45nm technology.

Table 12. The system simulation configuration

PE	256 16-bit multipliers, 256 16-bit adders
Mesh	24×24 mesh topology, XY routing, wormhole switching, 4-stage pipeline, 4 virtual channels per port
Fattree_Mesh[52]	4-ary fat-tree with a height of 3 in local, 6×6 mesh in global connection, nearest-common ancestor routing for fat-tree, XY-routing for mesh
Neu_NoC	8 nodes in local ring topology, 8×8 mesh in global with XY-routing, 1 cycle router delay for passing through, minimum 3 cycle router delay for injection/ejection

3.5.4 Experimental results

3.5.4.1 Impact of concentration degree

Allocating more PEs on each ring network not only reduces the redundant data packages sent to the PEs of the next NN layer but also reduces the latency and energy overhead of the

routing on the global mesh. However, increasing the length of the ring network will also increase the wire delay. Figure 39 compares the execution times of all the NN benchmarks using a Neu-NoC with ring and mesh configurations. Here the total number of the PE is set to 648. When the number of PEs located in the same ring increases from 1 to 8, the NoC performance keeps improving due to the reduction of the global hops count. However, such trend stops when the scale of the ring network is too large (i.e., 16 PEs) so that the increase of the local wire delay has surpassed the reduction of the global hops count. Among all the tested benchmarks, *alexnet_cnn_cla* is most sensitive to the scale of the ring network because its large hidden layers (with 4096 neurons) fully occupies all the PEs in each ring. In the following experiments, we chose $k = 8, n = 9$ as our default configuration which demonstrates the best balance between the local wire latency and global hops count.

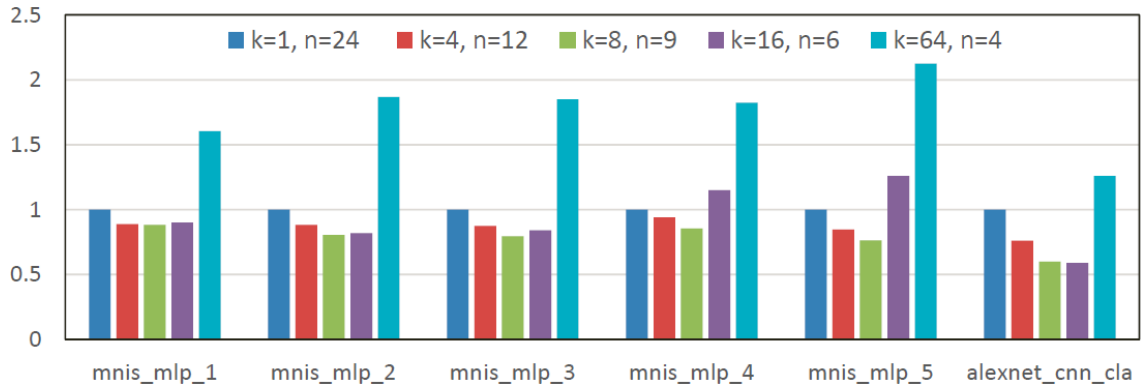

Figure 39. Impact of concentration degree (n=m).

3.5.4.2 Impact of feature map sparsity

We also evaluate the impact of the NN sparsity on the efficacy of Neu-NoC. Here the sparsity is defined as the percentage of 0's in the feature map. It is known that increasing the sparsity will degrade the accuracy of the NN. However, as shown in Figure 40, the accuracy degradation of all the benchmarks can be maintained at a very low level even the corresponding NN sparsity is very high: In *alexnet_cnn_cla*, which is the worst case, the average sparsity is 78.6% across three feature maps while the incurred accuracy is less than 1%. Figure 40 also illustrates the tradeoffs between the sparsity of the NN and its accuracy. In Neu-NoC, we can dynamically sparsify the NN by zeroing out any features smaller than a pre-determined threshold €. The columns of “Accuracy drop 1%” “Accuracy drop 2%” of Table 11 show the occurrence percentages of 4, 8, and 16 0's in sequence after NN pruning with 1% accuracy drop. The ratios between the corresponding traffic and the baseline traffic.

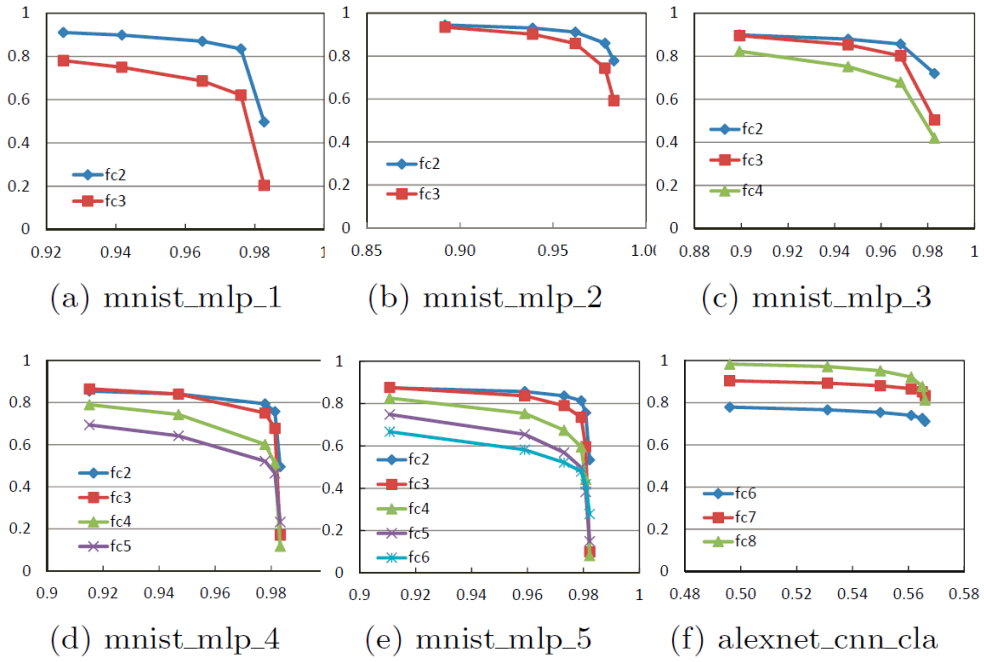

Figure 40. Accuracy impact of feature map sparsity (y-axis: sparsity).

3.5.4.3 Effectiveness of NN-aware mapping

Figure 41 illustrates the impacts of different NN mapping schemes on the average latency of packet transmission in Neu-NoC in the simulated neuromorphic acceleration system. Here x-axis is the data injection rate of the input neurons of the first layer, which is defined as the number of packets are inputted in each node in every cycle. The injection of the data transmission between the NN layers is triggered after the neuron collects all the inputs from the previous layer. To illustrate the effectiveness of our proposed NN-aware mapping, we compare the average packet latency of NN-aware mapping with that of a random mapping which randomly map the neurons to the accelerators and a naive sequence mapping places the neurons in the sequence of the accelerators' addresses. As shown in Figure 41, NN-aware mapping always achieves the best performance in all 6 tested benchmarks as well as the best tolerance to the traffic load (data injection rate) increase. For example, in 5 out of 6 benchmarks, the average packet latency of NN-mapping maintains low until the data injection rate exceeds about 0.09 while that of the other two mapping schemes starts to rocket when the data injection rate reaches about 0.05. In addition, NN-aware mapping demonstrates great scalability by showing more significant average packet latency reduction when the network scale is large, e.g., in *alexnet_cnn_cla* (see Figure 41(f)).

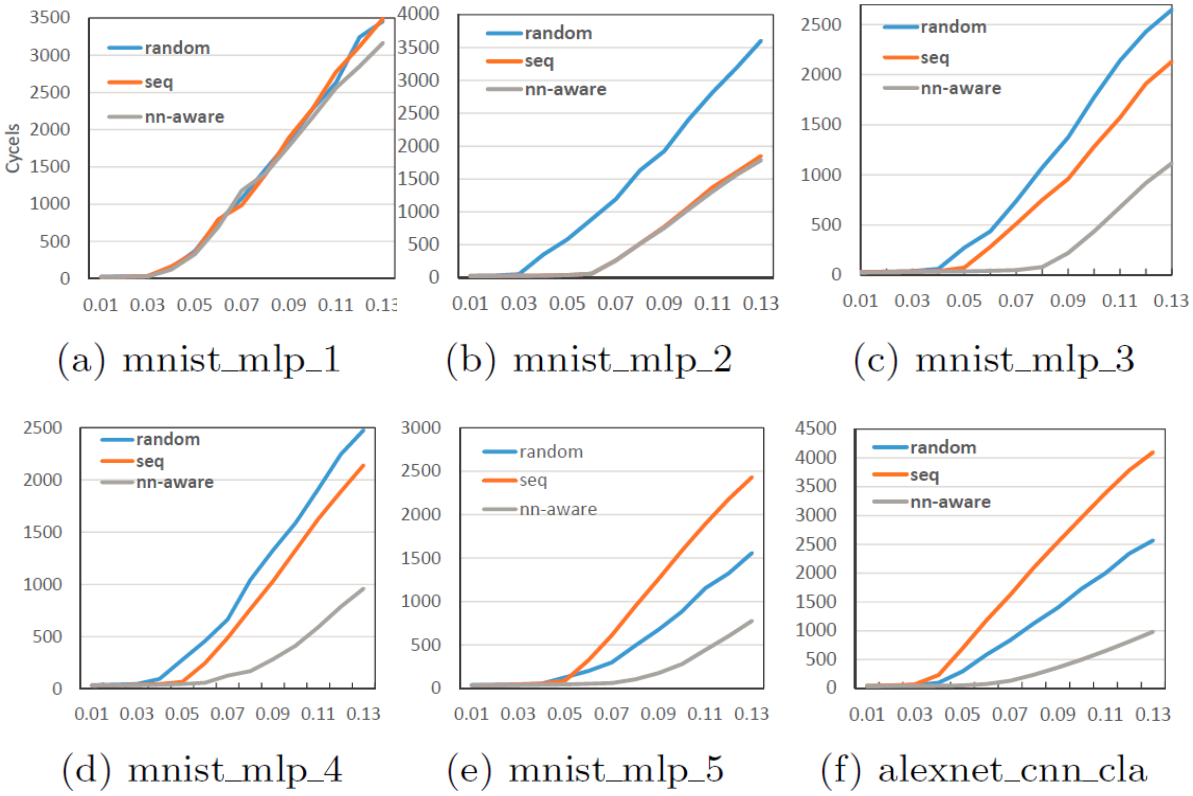

Figure 41. Average packet latency of different mappings (x-axis: injection rate).

3.5.4.4 Effectiveness of multicast

Figure 42 compares the average packet latencies before and after applying multicast in 5 benchmarks: *mnist_mlp_2 ~ mnist_mlp_5* and *alexnet_cnn_cla*. Note that here we did not include *mnist_mlp_1* because *mnist_mlp_1* does not have any data packet can be represented by other packets which share the same source node and contain its routing path. The result shows multicast successfully reduces that the average packet latency in all 5 benchmarks, especially in *Alexnet_cnn_cla* and *mnist_mlp_2* which have less layers and hence, less complicated mapping and routing paths than *mnist_mlp_3 ~ mnist_mlp_5*. The packet counts of each benchmark before and after applying multicast are depicted in Table 12.

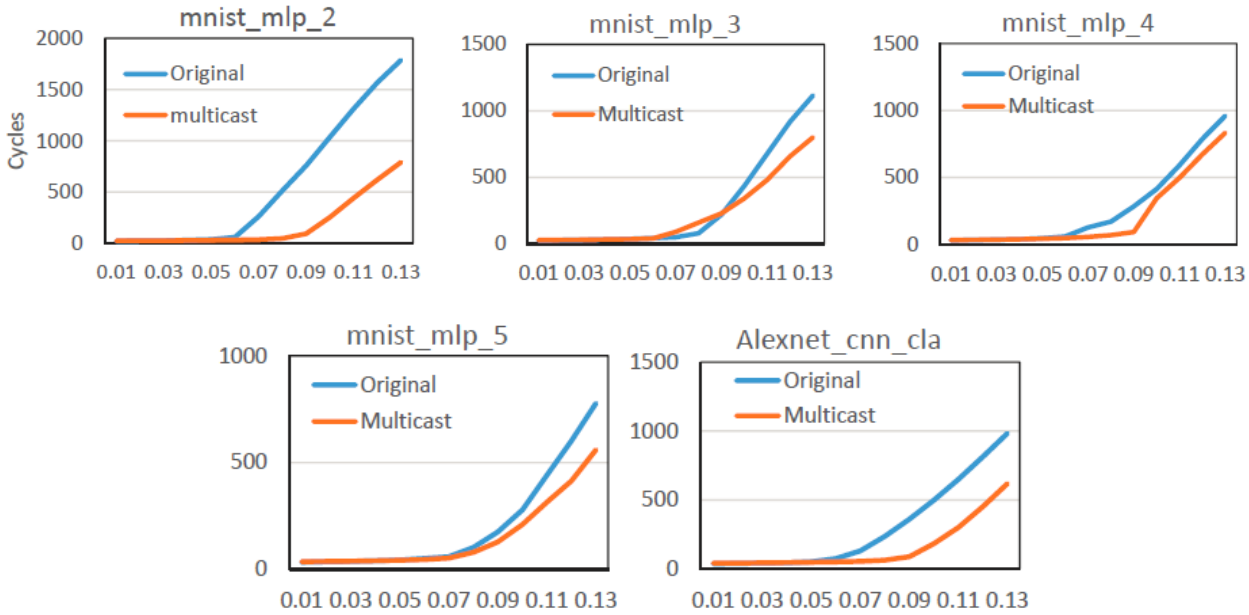

Figure 42. Average packet latency of before and after applying multicast (x-axis: injection rate).

3.5.4.5 Evaluation of Neu-NoC

We compare the execution time and energy consumption of three NoC designs: 2D Mesh, Fattree-Mesh [89], and Neu-NoC (including the design with and without zero flit design) for the 6 benchmarks, as depicted in Figure 43. All the results have been normalized to the baseline Mesh NoC design. Compared to Mesh NoC, Neu-NoC averagely reduce the average packet latency and energy consumption of the NoC by 23.2% and 31.1%, respectively. Compared to Fattree-Mesh NoC [89] which has been used in a neuromorphic acceleration system, Neu-NoC achieves on average 6% average packet latency reduction and 13% energy consumption saving, respectively. If we allow 1% accuracy degradation of NN (See Table 11), Neu-NoC with zero flit

design can further reduce the average packet latency and energy consumption of the NoC by 11.7% and 21.19%, respectively, compared to Fattree-Mesh NoC.

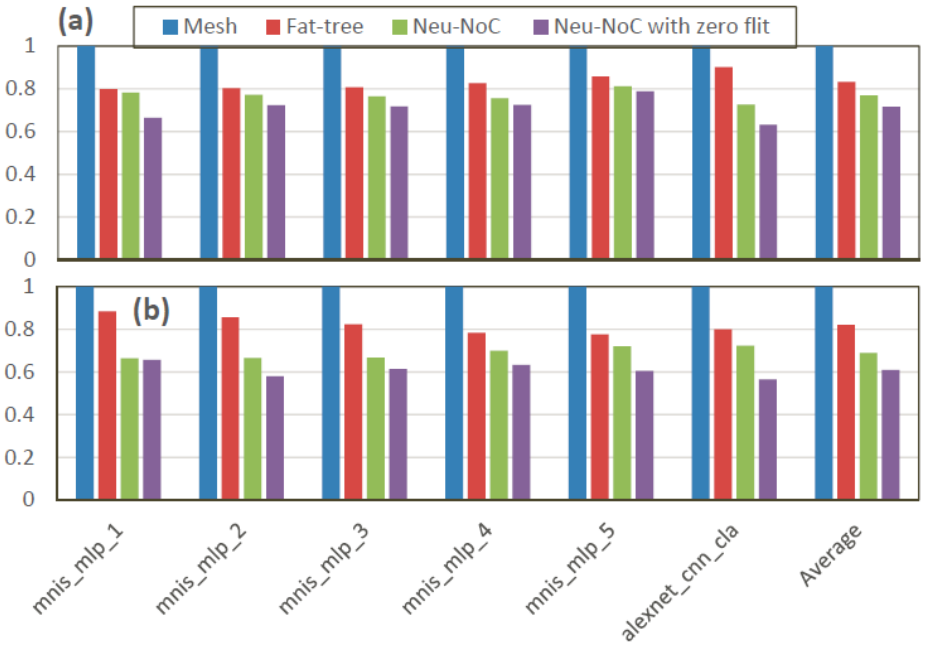

Figure 43. Normalized average packet latency and energy of all the NoC designs.

4.0 CONCLUSION AND FUTURE WORK

Emerging memory is a promising solution to combat the well-known memory bottleneck in both storage and computation systems of modern processor design. In this paper, we propose the use of STT-RAM in TLB for new virtually addressed GPUs. STT-RAM-based TLB introduces significant energy and performance advantages over SRAM implementation by realizing larger TLB capacity within the same area. We also present a novel STT-RAM-based dynamically-configurable TLB (STD-TLB) that leverages high read speed of STT-RAM differential sensing and dynamic configuration policy to retain the reduced miss rate from standard STT-RAM TLB while improving the translation performance of the heavily accessed pages. Compared to SRAM baseline, STD-TLB reduces TLB miss rate by 30%, boosts translation runtime performance by 55%, and improves the energy delay product by ~80%. As more systems employ on-board GPUs and more general purpose applications are mapped to these processors, alleviating the performance impact introduced by address mapping will become very difficult in GPU system design. For these applications, we expect STD-TLB will provide a dramatic system benefit (e.g., 10% performance improvement over standard STT-RAM TLB in MUM) by dynamically switching between high-performance and high-capacity mode based on real-time application needs.

In this work, we also first propose a MLC-STT register file (RF) design to overcome the limited scalability of SRAM-based RF in GPU architecture. By leveraging high integration

density and non-volatility of MLC-STT, our design dramatically reduces the area and leakage power. To overcome the performance degradation due to the slow access to MLC-STT cells, we propose a remapping strategy that allocates frequently-accessed registers into the faster soft-bit rows whenever it is possible, and a bank-status-aware-scheduling scheme to reorder the warp issuing to minimize the access stalls induced by the long write accesses. Experimental results show that the MLC-STT-RF design delivers a better system performance than that of conventional SRAM-based RF while achieving significant area reduction and system energy efficiency improvement.

Moreover, with the rediscovery of memristor, we propose a memristor-based neuromorphic computing accelerator (RENO) which tightly coupled with the general-purpose pipeline to largely improve the performance and energy efficiency of neuromorphic computing. Compared to conventional CPU, RENO achieves on average 177.67x (27.2x) performance speedup and 184.71x (25.18x) energy reduction over the simulated benchmarks processed by MLP (AAM) neural networks. AAM generally show better computation accuracy than MLP by performing a costly iterative computation. Nonetheless, the computation accuracy degradation is well constrained within a reasonably low range in RENO. The high computation and energy efficiency of RENO mainly come from: 1) the high-throughput of the mixed-signal NCA computation; 2) the excellent re-configurability of the hierarchical memristor crossbar array structure in the NCA; 3) the low data transmission overhead on the mixed-signal interconnection network (called M-Net); and 4) the concise coordination interface between the general-purpose pipeline and the NCA. An inline calibration scheme is also developed to control the run-time NCA computation accuracy degradation incurred by memristor resistance shift. Although only two ANN implementations are presented and discussed in our work, RENO can support a variety

of ANN types by properly reconfiguring the M-Net in the NCA and guiding the data routing among the MBC arrays. In our future research, we plan to extend RENO to multicore platform for broader ANN and learning applications, and investigate the design optimization of task partitioning and scheduling. The techniques to enhance the run-time robustness of RENO in training and testing procedures will be also studied. Finally, we will explore the compatibility of RENO to other existing neuromorphic computing practices, e.g., the spiking-based computation model like STDP.

We also analysis the features of the data traffic in neural networks and demonstrated the bottleneck of using traditional NoC for neuromorphic acceleration system. A dedicated NoC architecture is designed to optimize the traffic in neuromorphic computing systems. A set of techniques, i.e., NN-aware mapping, multicast, and sparsity-aware traffic reduction, are proposed to reduce average hops account and the redundant traffics. Our results show that Neu-NoC can effectively reduce the average packet latency and energy consumption in the tested 6 MLP benchmarks by on average 23.2% and 31.1%, respectively without incurring any accuracy degradations. Moreover, slightly relaxing the accuracy requirement of the benchmarks by 2% can further save the average packet latency and energy consumption by 28.5% and 39.2%, respectively. Our future work will focus on applying Neu-NoC to accelerate other components of DNNs such as convolutional and pooling layers.

BIBLIOGRAPHY

[1] O. Temam, “The rebirth of neural networks,” In ISCA, 2010.

[2] V. Agarwal, M. S. Hrishikesh, S. W. Keckler, and D. Burger, “Clock rate versus ipc: the end of the road for conventional microarchitectures,” in ISCA, 2000, pp. 248–259.

[3] M. D. Hill and M. R. Marty, “Amdahl’s law in the multicore era,” IEEE Computer, vol. 41, no. 7, pp. 33–38, 2008.

[4] A. Lukefahr, S. Padmanabha, R. Das, F. M. Sleiman, R. G. Dreslinski, T. F. Wenisch, and S. A. Mahlke, “Composite cores: Pushing heterogeneity into a core,” in MICRO, 2012, pp. 317–328.

[5] K. Fan, M. Kudlur, G. Dasika, and S. Mahlke, “Bridging the computation gap between programmable processors and hardwired accelerators,” in HPCA, 2009, pp. 313–322.

[6] A. R. Putnam, D. Bennett, E. Dellinger, J. Mason, and P. Sundararajan, “Chimps: A high-level compilation flow for hybrid cpu-fpga architectures,” in FPGA, 2008, pp. 261–261.

[7] F. Song, S. Tomov, and J. Dongarra, “Enabling and scaling matrix computations on heterogeneous multi-core and multi-gpu systems,” in Supercomputing, 2012, pp. 365–376.

[8] H. Wong *et al.*, “Demystifying gpu microarchitecture through microbenchmarking,” in *ISPASS*, 2010, pp. 235–246.

[9] Intel, Performance Analysis Guide for Intel Core i7 Processor and Intel Xeon 5500 processors, http://software.intel.com/sites/products/collateral/hpc/vtune/performance analysis guide.pdf.

[10] K. L. Spafford, J. S. Meredith, S. Lee, D. Li, P. C. Roth, and J. S. Vetter, “The tradeoffs of fused memory hierarchies in heterogeneous computing architectures,” in CF, 2012, pp. 103–112.

[11] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, “Gpuwattch: Enabling energy optimizations in gpgpus,” in ISCA, 2013, pp. 487–498.

[12] J. Sampson, G. Venkatesh, N. Goulding-Hotta, S. Garcia, S. Swanson, and M. B. Taylor, “Efficient complex operators for irregular codes,” in HPCA, 2011, pp. 491–502.

[13] G. Venkatesh, J. Sampson, N. Goulding, S. Garcia, V. Bryksin, J. Lugo-Martinez, S. Swanson, and M. B. Taylor, “Conservation cores: Reducing the energy of mature computations,” in ASPLOS, 2010, pp. 205–218.

[14] V. Govindaraju, C.-H. Ho, and K. Sankaralingam, “Dynamically specialized datapaths for energy efficient computing,” in HPCA, 2011, pp. 503–514.

[15] S. Gupta, S. Feng, A. Ansari, S. A. Mahlke, and D. I. August, “Bundled execution of recurring traces for energy-efficient general purpose processing,” in MICRO, 2011, pp. 12–23.

[16] B. Belhadj, A. Joubert, Z. Li, R. Héliot, and O. Temam, “Continuous real-world inputs can open up alternative accelerator designs,” in *ISCA*, 2013, pp. 1–12.

[17] M. Hosomi, H. Yamagishi, T. Yamamoto, K. Bessho, Y. Higo, K. Yamane, H. Yamada, M. Shoji, H. Hachino, C. Fukumoto et al., “A novel nonvolatile memory with spin torque transfer magnetization switching: Spin-ram,” in Electron Devices Meeting, 2005, pp. 459–462.

[18] Y. Zhang, I. Bayram, Y. Wang, H. Li, and Y. Chen, “ADAMS: Asymmetric differential STT-RAM cell structure for reliable and high-performance applications”, in ICCAD, pp. 9-16, 2013.

[19] Y. Chen, X. Wang, W. Zhu, H. Li, Z. Sun, G. Sun, and Y. Xie, “Access scheme of multi-level cell spin-transfer torque random access memory and its optimization,” in MWSCAS, 2010, pp. 1109–1112.

[20] Y. Zhang, L. Zhang, W. Wen, G. Sun, and Y. Chen, “Multi-level cell stt-ram: Is it realistic or just a dream?” in ICCAD, 2012, pp. 526–532.

[21] X. Bi, M. Mao, D. Wang, and H. Li, “Unleashing the potential of mlc stt-ram caches,” in ICCAD, 2013, pp. 429–436.

[22] L. Jiang, B. Zhao, Y. Zhang, and J. Yang, “Constructing large and fast multi-level cell stt-mram based cache for embedded processors,” in DAC, 2012, pp. 907–912.

[23] Nvidia, Fermi specifications, http://www.nvidia.com/object/fermiarchitecture.html.

[24] AMD, GCN Architecture, http://www.amd.com/us/products/technologies/gcn/Pages/gcn-architecture.aspx.

[25] www.itrs.net, in International Techonology Roadmap for Semiconductors, 2011.

[26] W. Wen *et al.*, “PS3-RAM: A fast portable and scalable statistical STTRAM reliability analysis method,” 2012, pp. 1187–1192.

[27] Z. Sun *et al.*, “Multi retention level stt-ram cache designs with a dynamic refresh scheme,” in *MICRO*, 2011, pp. 329–338.

[28] X. Wu *et al.*, “Hybrid cache architecture with disparate memory technologies,” in *ISCA*, 2009, pp. 34–45.

[29] Y. Chen *et al.*, “On-chip caches built on multilevel spin-transfer torque RAM cells and its optimizations,” *J. Emerg. Technol. Comput. Syst.*, vol. 9, no. 2, pp. 16:1–16:22, 2013.

[30] J. Zhao and Y. Xie, “Optimizing bandwidth and power of graphics memory with hybrid memory technologies and adaptive data migration,” in *ICCAD*, 2012, pp. 81–87.

[31] P. Shivakumar and N. P. Jouppi, “Cacti 3.0: An integrated cache timing, power, and area model,” Technical Report 2001/2, Compaq Computer Corporation, Tech. Rep., 2001.

[32] W. Zhao and Y. Cao, “New generation of predictive technology model for sub-45 nm early design exploration,” Electron Devices, vol. 53, no. 11, pp. 2816–2823, 2006.

[33] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, “Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory,” TCAD, vol. 31, no. 7, pp. 994–1007, 2012.

[34] C. Xu *et al.*, “Device-architecture co-optimization of stt-ram based memory for low power embedded systems,” in ICCAD, 2011, pp. 463–470, 2011.

[35] W. Zhao and Y. Cao, New Generation of Predictive Technology Model for Sub-45nm Early Design Exploration, 2006, vol. 53.

[36] A. Bakhoda, G. L. Yuan, W. W. Fung, H. Wong, and T. M. Aamodt, “Analyzing cuda workloads using a detailed gpu simulator,” in ISPASS, 2009, pp. 163–174.

[37] Nvidia,QuadroSpecifications,http://www.nvidia.com/object/product_quadro_fx_5800_us.html.

[38] J. A. Stratton *et al.*, “Parboil: A revised benchmark suite for scientific and commercial throughput computing,” in IMPACT Technical Report, 2012.

[39] A. Bhattacharjee and M. Martonosi, “Inter-core cooperative tlb for chip multiprocessors,” in ASPLOS, pp. 359–370, 2010.

[40] S. Ryoo, C. I. Rodrigues, S. S. Baghsorkhi, S. S. Stone, D. B. Kirk, and W.-m. W. Hwu, “Optimization principles and application performance evaluation of a multithreaded gpu using cuda,” in PPoPP, 2008, pp. 73–82.

[41] Nvidia, GeForce GTX 480, http://www.geforce.com/hardware/desktopgpus/geforce-gtx-480/specifications.

[42] R. Dorrance, F. Ren, Y. Toriyama, A. A. Hafez, C.-K. Yang, and D. Markovic, “Scalability and design-space analysis of a 1t-1mtj memory cell for stt-rams,” Electron Devices, vol. 59, no. 4, pp. 878–887, 2012.

[43] B. W. Coon, J. E. Lindholm, S. Liu, S. F. Oberman, and M. Y. Siu, “Operand collector architecture,” Nov. 16 2010, US Patent 7,834,881.

[44] N. Goswami, B. Cao, and T. Li, “Power-performance co-optimization of throughput core architecture using resistive memory,” in HPCA, 2013, pp. 342–353.

[45] M. Mao, W. Wen, Y. Zhang, Y. Chen, and H. H. Li, “Exploration of gpgpu register file architecture using domain-wall-shift-write based racetrack memory,” in DAC, 2014, pp. 1–6.

[46] X. Liu, Y. Li, Y. Zhang, A. K. Jones, and Y. Chen, “Std-tlb: A sttram-based dynamically-configurable translation lookaside buffer for gpu architectures.” in ASP-DAC, 2014, pp. 355–360.

[47] R. Venkatesan, S. G. Ramasubramanian, S. Venkataramani, K. Roy, and A. Raghunathan, “Stag: Spintronic-tape architecture for gpgpu cache hierarchies,” in ISCA, 2014, pp. 253–264.

[48] N. Jing, Y. Shen, Y. Lu, S. Ganapathy, Z. Mao, M. Guo, R. Canal, and X. Liang, “An energy-efficient and scalable edram-based register file architecture for gpgpu,” in ISCA, 2013, pp. 344–355.

[49] S. O. Haykin, Neural Networks and Learning Machines. London: Prentice Hall, 2008.

[50] K. Hornik, M. B. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” Neural Networks, vol. 2, no. 5, pp. 359–366, 1989.

[51] T. Chen, Y. Chen, M. Duranton, Q. Guo, A. Hashmi, M. H. Lipasti, A. Nere, S. Qiu, M. Sebag, and O. Temam, “Benchnn: On the broad potential application scope of hardware neural network accelerators,” in IISWC, 2012, pp. 36–45.

[52] O. Temam, “A defect-tolerant accelerator for emerging high-performance applications,” in ISCA, 2012, pp. 356–367.

[53] H. Wang, Y. Wu, B. Zhang, and K. L. Du, “Recurrent neural networks: Associative memory and optimization,” J Inform Tech Soft Engg, 2011.

[54] B. Liu, M. Hu, H. Li, and Y. chen, “Digital assisted noise eliminating training for memristor crossbar based analog neuromorphic computing engine,” in DAC, 2013.

[55] L. O. Chua, “Memristor-the missing circuit element,” IEEE Transactions on Circuit Theory, vol. 18, no. 5, pp. 507–519, 1971.

[56] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, “The missing memristor found,” Nature, vol. 453, pp. 80–83, 2008.

[57] S. Shin, K. Kim, and S.-M. Kang, “Memristor applications for programmable analog ics,” IEEE Transactions on Nanotechnology, vol. 10, no. 2, pp. 266–274, 2011.

[58] W. Yi, F. Perner, M. S. Qureshi, H. Abdalla, M. D. Pickett, J. Yang, M.-X. M. Zhang, G. Medeiros-Ribeiro, and R. S. Williams, “Feedback write scheme for memristive switching devices,” Applied Physics A, vol. 102, no. 4, pp. 973–982, 2011.

[59] F. Alibart, L. Gao, B. D.Hoskins, and D. B.Strukov, “High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm,” Nanotechnology, vol. 23, no. 7, 2012.

[60] L. O. Chua and S.-M. Kang, “Memristive devices and systems,” Proceedings of the IEEE, vol. 64, no. 2, pp. 209–223, 1976.

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

[62] K. K. Likharev, “Crossnets: Neuromorphic hybrid cmos/nanoelectronic networks,” Science of Advanced Materials, vol. 3, no. 3, pp. 322–331, 2011.

[63] G. Indiveri, B. Linares-Barranco, R. Legenstein, G. Deligeorgis, and T. Prodromakis, “Integration of nanoscale memristor synapses in neuromorphic computing architectures,” Nanotechnology, 2013.

[64] K.-H. Kim, S. Gaba, D. Wheeler, J. M. Cruz-Albrecht, T. Hussain, N. Srinivasa, and W. Lu, “A functional hybrid memristor crossbararray/cmos system for data storage and neuromorphic applications,” Nano letters, vol. 12, no. 1, pp. 389–395, 2011.

[65] F. Alibart, E. Zamanidoost, and D. B. Strukov, “Pattern classification by memristive crossbar circuits using ex situ and in situ training,” Nature communications, vol. 4, 2013.

[66] M. Hu, H. Li, Q. Wu, and G. S. Rose, “Hardware realization of bsb recall function using memristor crossbar arrays,” in DAC, 2012, pp. 498–503.

[67] A. Joubert, B. Belhadj, O. Temam, and R. Heliot, “Hardware spiking neurons design: analog or digital?” in International Joint Conference on Neural Networks, 2010.

[68] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, “Neural acceleration for general-purpose approximate programs,” in MICRO, 2012, pp. 449–460.

[69] S. Yu, Y. Wu, and H.-S. P. Wong, “Investigating the switching dynamics and multilevel capability of bipolar metal oxide resistive switching memory,” Applied Physics Letters, vol. 98, no. 10, pp. 103 514–103 514–3, 2011.

[70] J. Balfour and W. J. Dally, “Design tradeoffs for tiled cmp on-chip networks,” in ICS, 2006, pp. 187 – 198.

[71] O. Temam, “A defect-tolerant accelerator for emerging high-performance applications,” in ISCA, 2012, pp. 356–367.

[72] L. Dai and R. Harjani, “Cmos switched-op-amp-based sample-andhold circuit,” in SOLID-STATE CIRCUITS, 2000.

[73] B. J. Choi, A. B. Chen, X. Yang, and I.-W. Chen, “Purely electronic switching with high uniformity, resistance tunability, and good retention in pt-dispersed sio2 thin films for reram,” Advanced Materials, vol. 23, no. 33, pp. 3847–3852, 2011.

[74] W. Zhao and Y. Cao, “Predictive technology model for nano-cmos design exploration,” JETC, vol. 3, no. 1, p. 1, 2007.

[75] M. Gustavsson, J. J. Wikner, and N. Tan, CMOS data converters for communications, 2000.

[76] F. Krummenacher, “Micropower switched capacitor biquadratic cell,” Solid-State Circuits, vol. 17, no. 3, pp. 507–512, 1982.

[77] “Virtuoso,” http://www.cadence.com/products/cic/pages/default.aspx.

[78] S. Shapero and P. Hasler, “Mismatch characterization and calibration for accurate and automated analog design,” Circuits and Systems, vol. 60, no. 3, pp. 548–556, 2013.

[79] L. Prechelt, “Proben1-a set of neural network benchmark problems and benchmarking rules,” University of Karlsruhe, Tech. Rep., 1994.

[80] “Uci machine learning repository,” http://archive.ics.uci.edu/ml/.

[81] “The mnist database of handwritten digits,” http://yann.lecun.com/exdb/mnist/.

[82] S. Nissen, “Implementation of a fast artificial neural network library (fann),” Department of Computer Science, University of Copenhagen, Tech. Rep., 2003.

[83] “Macsim,” http://code.google.com/p/macsim/.

[84] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, “Pin: building customized program analysis tools with dynamic instrumentation,” in PLDI, 2005, pp. 190–200.

[85] “Intel atom proccesor,” http://ark.intel.com/products/family/29035/.

[86] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, “Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures,” in MICRO, 2009, pp. 469–480.

[87] N. Jian, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, J. Kim, andW. J. Dally, “A detailed and flexible cycle-accurate networkon-chip simulator,” 2013.

[88] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, “Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning,” in ASPLOS, 2014.

[89] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun et al., “Dadiannao: A machine-learning supercomputer,” in MICRO, 2014, pp. 609–622.

[90] C. Park, R. Badeau, L. Biro, J. Chang, T. Singh, J. Vash, B. Wang, and T. Wang, \A 1.2 tb/s on-chip ring interconnect for 45nm 8-core enterprise xeon R processor," in ISSCC, 2010.

[91] B. Grigorian, N. Farahpour, and G. Reinman, “BRAINIAC: Bringing reliable accuracy into neurally-implemented approximate computing,” in the 21st IEEE International Symposium on High Performance Computer Architecture (HPCA), pp. 615-626, 2015.

[92] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, “Caffe: Convolutional architecture for fast feature embedding,” in Proceedings of the ACM International Conference on Multimedia, pp. 675-678, 2014.

[93] LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” Proceedings of the IEEE, vol. 86, pp. 2278-2324, 1998.

[94] R. Das, A. K. Mishra, C. Nicopoulos, D. Park, V. Narayanan, R. Iyer, M. S. Yousif, and C. R. Das, “Performance and power optimization through data compression in network-on-chip architectures,” in the 14th IEEE International Symposium on High Performance Computer Architecture, pp. 215-225, 2008.

[95] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath et al., “Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," Signal Processing, vol. 29, no. 6, pp. 82-97, 2012.

[96] T. N. Sainath, A.-r. Mohamed, B. Kingsbury, and B. Ramabhadran, “Deep convolutional neural networks for lvcsr," in ASSP, pp. 8614-8618, 2013.

[97] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classi_cation with deep convolutional neural networks," in Advances in neural information processing systems, pp. 1097-1105, 2012.

[98] “GTX titan x," http: //www.geforce.com/hardware/10series/titan-x-pascal.

[99] S. K. Esser, A. Andreopoulos, R. Appuswamy, P. Datta, D. Barch, A. Amir, J. Arthur, A. Cassidy, M. Flickner, P. Merolla et al., “Cognitive computing systems: Algorithms and applications for networks of neurosynaptic cores," in IJCNN, pp. 1-10, 2013.

[100] F. Akopyan, et al., “Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip," ICS, vol. 34, no. 10, pp. 1537-1557, 2015.

[101] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, “Squeezenet: Alexnet-level accuracy with 50x fewer parameters and < 0.5 mb model size," arXiv preprint arXiv:1602.07360, 2016.

[102] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky, “Sparse convolutional neural networks," in CVPR, pp. 806-814, 2015.

[103] W. J. Dally and B. P. Towles, Principles and practices of interconnection networks. Morgan Kaufmann Inc, 2004.

[104] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation," in CVPR, June 2014.

[105] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, “Learning structured sparsity in deep neural networks," in NIPS, pp. 2074-2082, 2016.

[106] D. C. Cire_san, U. Meier, L. M. Gambardella, and J. Schmidhuber, “Deep big multilayer perceptrons for digit recognition," in Neural Networks: Tricks of the Trade, pp. 581-598, 2012.