SEAL-lab Technical Report – No. 2015-001 (April 29, 2016)

Processing-in-Memory in ReRAM-based Main Memory

Ping Chi[1] Shuangchen Li[1] Ziyang Qi[1] Peng Gu[1] Cong Xu[2] *
Tao Zhang[3] Jishen Zhao[4] Yongpan Liu[5] Yu Wang[5] Yuan Xie[1]

University of California, Santa Barbara[1] HP Labs[2] NVIDIA Corporation[3]
University of California, Santa Cruz[4] Tsinghua University[5]
{pingchi, shuangchenli, yuanxie}@ece.ucsb.edu

ABSTRACT

Process in memory (PIM) is a promising solution to address the "memory wall" issue, and most of the proposed PIM architectures integrate logic with memory. However, as long as the computation is done by logic, it needs to access memory for data. In this work, we propose a novel PIM system, utilizing metal-oxide resistive random access memory (ReRAM) as main memory and also for computation. As an emerging non-volatile memory, ReRAM has been considered as a promising candidate for future memory architecture, thanks to its high density, fast read access, and low leakage power. In addition, it has the capability to represent synaptic weights and execute neural networks with its crossbar architecture. Our proposed design is based on a ReRAM-based main memory system. In the proposed design, a portion of ReRAM crossbar arrays can be configured either as normal memory or to be used for neural computation (hence a morphable PIM structure). We design the required peripheral circuits by computing carefully to minimize the area overhead, and also to provide a hardware-software interface for the developers so that they can easily configure the proposed design for various neural network tasks. We evaluate the proposed design on two sets of benchmarks, and compare it with a CPU-only solution and two neural process unit (NPU) solutions (using NPUs as co-processors and as PIM-processors through 3D stacking). The experiment results show that the proposed design achieve the best energy-efficiency with the highest speedup, which is ∼9000× faster than the CPU-only solution for large neural networks. In addition, it is also a low-overhead PIM solution, with only $< 3.8\%$ *area overhead compared to the normal ReRAM chips.*

1. INTRODUCTION

Conventional computer systems favor separate computation (CPUs) and data storage (memory) components. While the volume of data that computer systems process has skyrocketed over the last decade, data movement between CPUs and the memory is becoming one of the most critical performances and energy bottlenecks in various computer systems ranging from cloud servers to end-user devices. For example, data transfer between CPUs and off-chip memory consumes two orders of magnitude more energy than a floating point operation [1].

Separate computation and memory has remained for the past several decades in the Von Neumann architecture for certain reasons. From a bottom-up perspective, traditional digital logic components and SRAM/DRAM cells are incompatible to each other; they are physically laid out separately. From a top-down perspective, most traditional applications require to perform intensive and accurate logic/arithmetic operations, which can saturate the sophisticated computation resource of CPUs.

However, this computer design principle is being undermined by recent progress in resistive RAM (ReRAM) technologies [2] (which incorporate both computation and memory capabilities) and the trend in applications (that are increasingly data intensive and tolerate approximate computation). While some prior work show that ReRAMs are promising DRAM/flash replacement, recent studies demonstrated that ReRAM also has the capability of performing logic and arithmetic operations [3, 4, 5, 6, 7]. This unique property promises a radical renovation of the relationship between computation and memory. Moreover, we can trade-off the accuracy of ReRAM-based computation for better performance [6], because modern applications abound use cases that allow certain levels of approximation in computation, such as media processing (audio, video, graphics, and image), recognition, and data mining.

Motivated by ReRAM's unique capability and application trends, we propose a design that process in ReRAM-based main memory, which accelerates neural network (NN) and approximate computing applications by exploiting the same set of ReRAM cells as both computation elements and memory. In particular, we redesign the peripheral circuits of a portion of ReRAM arrays, so that we can dynamically reconfigure the arrays as memory or as computation unit (hence we call it *morphable PIM*). As such, initializing computation on the local data only requires a reconfiguration

*Ping and Shuangchen have equal contribution to this work.

and provides high-bandwidth intra-memory-bank data movement.

The fundamental uniqueness of the proposed design is that it exploits ReRAM cells to perform both computation and memory functionality. Most prior work exploit ReRAM either as DRAM/flash replacement [8, 9, 10] or as synapses in artificial neural networks [3, 4, 5, 6, 7, 11] to perform computation. The proposed design's unique property also distinguishes itself from previous studies on process in memory (PIM) [12, 13, 14], Micron's Automata processor [15], and 3D stacked memories with a logic layer that encapsulates processing units to perform computation [16, 17]. Although these previous designs allow memory to have computation abilities, they employ CMOS-based processing elements to perform the memory-side computation. As such, they still implement memory and computation in separate components.

Recent work employs nonvolatile memory technologies (ReRAM, phase change memory, and spin-transfer torque RAM) to build ternary content addressable memories (TCAMs), which exploits memory cells to perform associative search operations [18, 19, 20]. However, to support such search operations, these studies require redesign of nonvolatile memory cell structures to enable much larger cell sizes, which inevitably increases the cost of memory. Since memory is extremely sensitive to the cost, it is critical to design a PIM with low cost overhead. Compared to previous TCAM designs, the proposed design obviates the cost of redesigning memory cells. Furthermore, the proposed design supports much more sophisticated computation than TCAMs.

In summary, we make the following contributions:

- We propose a morphable ReRAM main memory architecture built upon ReRAM crossbar arrays, in which some arrays can either work as normal memory or perform neural computations on demand.
- We present a complete circuit/microarchitecture design to enable the adaptive functionality while keeping the cost overhead low. To reduce the area overhead of the peripheral circuits, we propose the polymorphism of peripheral circuits and peripheral circuits sharing between adjacent arrays.
- The proposed design includes a hardware-software interface so that the developers can configure the arrays to implement different topologies of NNs. Also, we show how to make use of the bank-level parallelism to accelerate NN tasks with various inputs.

2. BACKGROUND AND RELATED WORK

In this section, we will introduce the basics of ReRAM technology and also discuss related work on ReRAM-based memory and computation.

2.1 ReRAM Basics

Resistive random access memory, known as ReRAM or RRAM, is a type of non-volatile memory that stores information by changing the cell resistances. The general definition does not specify the resistive switching material or technology. This work focuses on a subset of resistive memories, called metal-oxide ReRAM, which uses metal oxide layers as switching materials.

Figure 1 (a) demonstrates the metal-insulator-metal (MIM) structure of an ReRAM cell: a top electrode, a bottom electrode, and a metal-oxide layer sandwiched between electrodes. Almost 40 binary metal oxide materials have demonstrated resistive behavior under electrical field. Among them about ten are compatible with CMOS process, making them attractive candidates for building ReRAM cells [2]. By applying an external voltage across it, an ReRAM cell can be switched between a high resistance state (HRS or OFF-state) and a low resistance state (LRS or On-state), which are used to represent the logical "0" and "1", respectively. According to the polarity of the programming voltage, ReRAM cells can be classified into unipolar ReRAM and bipolar ReRAM. The resistance switching of a unipolar ReRAM only depends on the magnitude of programming voltage, whereas in bipolar ReRAM, HRS-to-LRS switching (SET operation) and LRS-to-HRS switching (RESET operation) require programming voltages with opposite polarities. Compared to unipolar ReRAM, bipolar ReRAM is more attractive because of its good cell characteristics, better switching uniformity, and operating margin. As a result, this work focuses on the bipolar ReRAM technology.

Figure 1 (b) shows the I-V characteristics of a typical bipolar ReRAM cell. Switching a cell from HRS (logic "0") to LRS (logic "1") is a SET operation, and the reverse process is a RESET operation. To SET the cell, a positive voltage that can generate sufficient write current is required. To RESET the cell, a negative voltage with proper magnitude is necessary. The reported endurance of ReRAM is up to 10^{12} [21, 22], making the lifetime issue of ReRAM-based main memory less concern than phase change memory (PCM) based main memory whose endurance has been assumed between 10^6-10^8 [23]. To read the state of a cell, a read voltage is applied which should be small enough in order not to flip the cell unintentionally.

A possible ReRAM array organization is based on the conventional one-transistor-one-resistor (1T1R) ReRAM cell structure. In this structure, a MOSFET transistor is required to be integrated with the ReRAM cell as the access device. With this design, it is possible to accurately control the current to the activated cells through their dedicated access transistors. On the flip side, the size of MOSFET should be large enough to satisfy the current requirement of the SET and RESET operations. This ultimately increases the cell area and cost. One solution to reduce the cost is to organize an ReRAM array in an area-efficient crossbar structure, which eliminates the access transistor, as shown in Figure 1 (c). By doing so, a $4F^2$ cell size can be achieved. The biggest

drawback of a crossbar design comes from the multiple possible current paths in an array [24]. A lot of previous studies proposed different solutions to the problems introduced by the sneak current [24, 25], [26, 27, 8, 9, 10]. At device level, a selector can be fabricated on the top of the ReRAM to construct a non-linear cell in order to suppress the cell current when biased at half voltage [26, 27]. At circuit level, different biasing schemes and peripheral circuit design options have been analyzed [25, 27, 24]. At architecture level, the variable write latency caused by the sneak current is optimized with an intelligent memory controller [10].

There are two common approaches to further improve the density and reduce the cost of ReRAM: multi-layer crossbar architecture [26, 27, 28, 9] and multi-level cell (MLC) [29, 30]. ReRAM cells can store more than one bit of information in a single cell. This MLC characteristic can be realized by changing the resistance of ReRAM cell gradually under finer SET current or RESET voltage control. Recent work has demonstrated 7-bit MLC ReRAM [31].

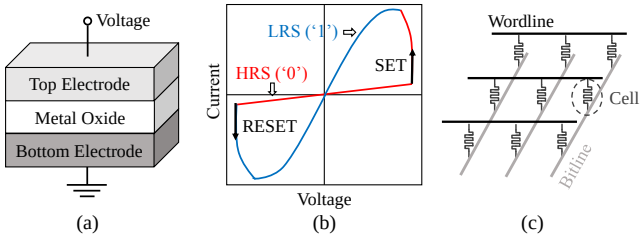

Figure 1: (a) Conceptual view of an ReRAM cell; (b) I-V curve of bipolar switching; (c) schematic view of a crossbar architecture.

2.2 ReRAM Based Memory/Storage Systems

ReRAM has been considered as a cost-efficient replacement of DRAM to build next-generation main memory [10]. With the crossbar array architecture, ReRAM can achieve a better density than DRAM. The read latency of ReRAM can be comparable to that of DRAM while its write latency is significantly longer than that of DRAM (*e.g.* 5×). One challenge for ReRAM crossbar architecture is the long RESET latency due to the voltage drop caused by sneak paths, because the switching time of an ReRAM cell is inversely exponentially related to the voltage applied. Several architectural techniques were proposed [10] to improve the write performance, bridging the performance gap between their optimized ReRAM and DRAM within 10%. In this work, we adopt a similar performance optimized design of the ReRAM based main memory.

In comparison with NAND flash, ReRAM has orders of magnitude lower read/write latency, lower write energy per bit, and higher endurance. The cost per bit of ReRAM remains to be the key challenge as a NAND flash replacement. One way to reduce the effective cell size is to stack multiple layers within a die without thermal interference. Recently, a two-layer cross-point 32Gb ReRAM chip, which has an effective cell size of $2F^2$, was demonstrated by Toshiba and SanDisk [27]. Jung *et al.* also proposed the hierarchical design of a large-scale storage-class ReRAM system, using multi-layer cross-point architecture [8].

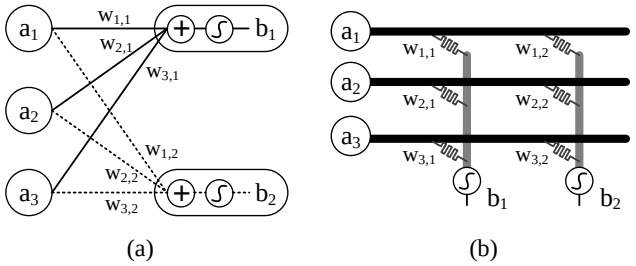

Figure 2: (a) An artificial neural network with one input layer and one output layer; (b) using an ReRAM crossbar array for neural computation.

2.3 Using ReRAM for Neural Computation

Artificial neural networks (ANNs) are a family of machine learning algorithms inspired by the human brain structure. They are presented as systems of interconnected neurons generally, containing an input layer, an output layer, and probably one or more hidden layers. Figure 2 (a) shows a simple neural network with an input layer of three neurons, an output layer of two neurons, and no hidden layers. The output b_j is calculated as,

$$b_j = \sigma(\sum_{\forall i} a_i \cdot w_{i,j}), \quad (1)$$

where $w_{i,j}$ are synaptic weights, σ is a non-linear function, $i = 1, 2, 3$, and $j = 1, 2$.

In the era of big data, machine learning is widely used to learn from and make predictions on huge amount of data. With the advent of deep learning, some neural network algorithms such as convolutional neural networks (CNNs) and deep neural networks (DNNs) start to show their power and effectiveness across a wide range of applications [32, 33]. NNs are also used to accelerate approximate computing [34, 35, 36]. There are many studies on accelerating NNs in hardware on the platform of GPU [37, 38, 39], FPGA [40, 41, 42] or ASIC [43, 44, 45, 32, 33].

Historically, it is challenging to realize a neuromorphic system in hardware with both neurons and synapses implemented by CMOS circuits, because there are thousands of synapses each neuron and each synapse occupies huge area with tens of transistors. As ReRAM emerges with the crossbar architecture, it becomes a good candidate to build area-efficient synaptic arrays for neuromorphic computing [3, 4, 5, 7]. Recently, a 12×12 ReRAM crossbar prototype is fabricated with a fully operational neural network successfully classifying 3×3-pixel black/white images into 3 categories [5]. Using ReRAM for neural computation can improve the execution time and power efficiency significantly because an ReRAM crossbar array can represent a connection matrix efficiently and implement the matrix-vector multiplication function in an analog manner. Figure 2 (b)

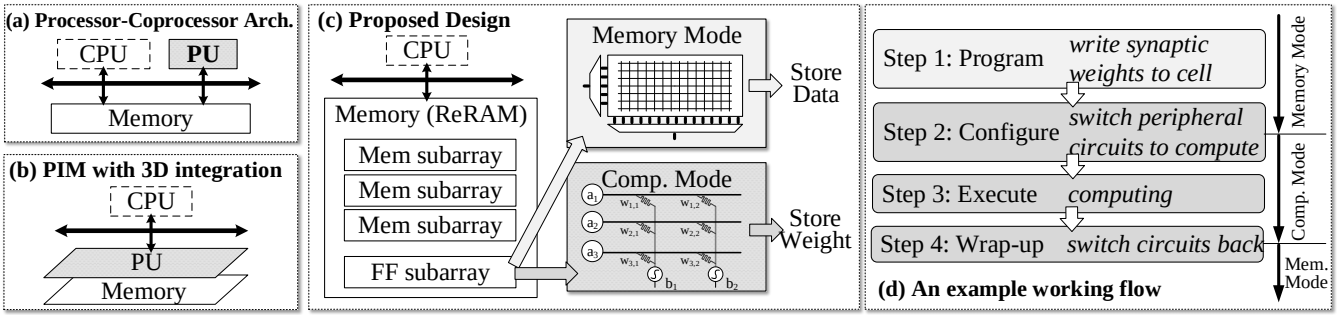

Figure 3: The proposed design overview. Comparison of (a) traditional shared memory based processor-coprocessor architecture, (b) process in memory (PIM) approaches that leverage 2.5D/3D integration technologies, and (c) the proposed process in ReRAM-based main memory design. (d) An example the proposed design working flow.

shows an example of using a 3×2 ReRAM crossbar array to execute the neural networks in Figure 2 (a). The input data a_i is represented by analog input voltages on the wordlines. The synaptic weights $w_{i,j}$ are programmed into the cell conductances in the crossbar array. Then the current flowing to the end of each bitline is viewed as the result of the matrix-vector multiplication, $\sum_i a_i \cdot w_{i,j}$. After the current on each bitline is sensed, a non-linear function unit is adopted to complete the execution.

Implementations of NNs with ReRAM crossbar arrays require specialized peripheral circuit design. First, the matrix-vector multiplication is analog computation with current. Therefore, the inputs require digital-to-analog converters (DACs) and the outputs need analog-to-digital converters (ADCs). Second, the conductances of ReRAM cells can only represent either positive or negative synaptic weights. To tackle this problem, the connection matrix of the synaptic weights are separated into two matrices, one containing all the positive weights and the other containing all the negative weights, and two ReRAM crossbar arrays are used to implement them in parallel, and then an analog subtraction circuit is used to process the results of the two crossbar arrays. Third, programming the synaptic weights to a crossbar array requires accurate control of cell resistance levels. This is the same procedure with a write operation in MLC ReRAM, and additional circuits to support the program-and-verify mechanism are needed.

2.4 Comparison To The State Of The Art

There is plenty of work on CMOS-based brain-like computing processor design [43, 44]. For example, TrueNorth [45] is a message passing based multi-core cognitive computing system, in which each core computes spiking neuron networks with an SRAM based crossbar structure. However, to implement a 6-bit precision synapse, TrueNorth's SRAM solution requires $876F^2$, whereas the proposed design's ReRAM solution only needs $4F^2$. Furthermore, the proposed design adopts a totally different communication architecture to support PIM with a low area overhead. There is also a lot of work on ReRAM-based neuromorphic computing system [3],[5],[7]. Taha *et al.* [11] proposed an NoC based many core system with ReRAM crossbar arrays for NN computing. Different from the proposed design, this is not a PIM design and does not use ReRAM as main memory. Therefore, for large-scale NNs, it requires much higher memory bandwidth and consumes much more energy than a PIM design which reduces the data movement effectively. Moreover, the whole system requires a large area footprint since it does not explore ReRAM's morphability between memory and computing functionalities.

3. ARCHITECTURE

Overview. We propose process in ReRAM-based main memory, which efficiently accelerates neuromorphic computing by leveraging ReRAM's unique yet largely overlooked property – having computation and data storage abilities in one device. Figure 3 depicts an overview of our design. While previous approaches require additional processing units (PU) (Figure 3(a) and (b)), our design directly leverages ReRAM cells to perform computation without the need for extra PUs. As shown in Figure 3(c), our design partitions an ReRAM bank into two regions: memory (Mem) subarrays and a small number of full function (FF) subarrays. The Mem subarrays only have data storage capability (the same as conventional memory subarrays). The FF subarrays have both computation and data storage capabilities, and they can work in two modes. In memory mode, the FF subarrays serve as conventional memory. In computation mode, the FF subarrays can execute NNs.

We tailor the peripheral circuits of the memory region with the FF subarrays to enable its switch between memory mode and computation mode. Figure 3(d) illustrates an example of the bi-directional mode switching. When the FF subarrays switch from memory mode to computation mode, if they store some data, the data should be migrated to some allocated space first. The computation mode works as follows. First, the synaptic weights are written to the corresponding crossbar arrays in the FF subarrays. Second, the peripheral circuits are configured for computation functionality after

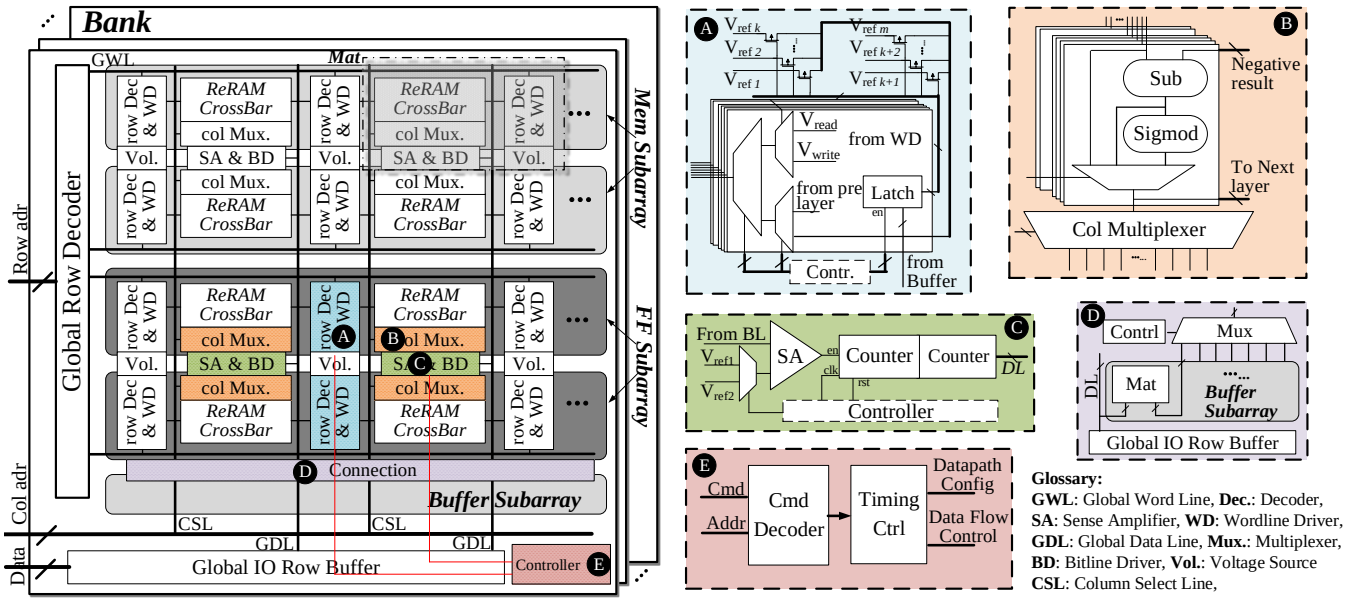

Figure 4: The proposed Architecture. Left: bank structure; Right: functional blocks modified/added in the proposed design. (a) wordline driver with multi-level voltage sources; (b) column multiplexer with analog subtraction and sigmoid circuitry; (c) SA uses counters to output multiple levels; (d) connection between FF and Buffer subarrays; (e) The controller.

receiving the commands from the controller. Third, the FF subarrays execute the mapped NNs. When the FF subarrays are about to switch back to memory mode, the wrap-up step reconfigures the peripheral circuits for memory functionality.

3.1 Full Function Subarray Design

The design goal of the FF subarrays is to support memory and computing functionalities simultaneously with minimum area overhead. The key idea is to maximize the reuse of the peripheral circuits for both memory and computation.

The microarchitecture design of the Mem subarrays is adopted from a performance-optimized ReRAM main memory [10]. As shown in Figure 4, each Mem subarray contains multiple mats, and each mat contains an ReRAM crossbar array of SLC 0T1R cells. There are two major parts of peripheral circuits in Mem subarrays. The first part includes the decoders and the wordline drivers. The wordline driver can provide a certain voltage (among 0, 1/2Vdd, and Vdd) to the selected wordline, and they are shared by horizontally adjacent two mats in the same subarray to reduce area overhead. The second part consists of the column multiplexers and the sense amplifiers (SAs). SAs are shared between the vertically adjacent two mats in different subarrays.

To enable the neural computing functionality, additional peripheral circuits are required in the FF subarrays, as described in Section 2.3. We choose 6-bit precisions for input, output, and synaptic weights for computation as it strikes a balance between complexity and performance after exploring a large design space. As such, 6-bit DACs, and 6-bit ADCs, and the circuit to support 6-bit MLC write operations are required. Subtraction and non-linear threshold (sigmoid) circuits are also introduced in the FF subarrays. As shown in Figure 4, we modify the decoders and drivers, the column multiplexers, and SAs. Global wires are used to route the data between the Buffer subarray and FF subarrays.

The design of the FF subarrays has two advantages. First, we make the best use of the peripheral circuits and share them between memory and computation functionality, which minimizes the area overhead in comparison to state-of-the-art. For example, in a typical ReRAM-based neuromorphic computing system [3], DACs and ADCs are used for input and output; while in most ReRAM-based memory design, write drivers and SAs are used for input and output. Write drivers and DACs serve for similar functions, providing us the opportunity to unify their design. In the proposed design, instead of adding the dedicated DACs and ADCs, the write drivers and SAs are slightly modified to serve the ADC and DAC functions as well. Second, the FF subarrays can be configured between memory and computation flexibly and efficiently. A more aggressive design can be easily extended to configure some mats in one FF subarray for computation and the other mats in the same FF subarray as memory. The feasibility of such design is based on the fact that multiplexers are used to connect all of the add-on circuits with the original peripheral circuits for memory functionality. Moreover, the Buffer subarray is also flexible to be configured as memory or buffer for computation with little modifications.

3.2 Buffer for the FF Subarrays

The purpose of the Buffer subarrays is two-fold. First, they are used to cache the input and output data for

the FF subarrays. By benefiting from the massive parallelism of the multiply-addition operations provided by ReRAM crossbar structures, the computation itself takes a very short time. For example, to execute a $256-256$ NN, the computation consumes only $\sim 1.4\mu s$. However, the input and output latencies become potential bottlenecks, since they may require to be provided or stored serially. Therefore, it is necessary to cache the input and output data. Second, the FF subarrays can communicate with the Buffer subarrays directly without the involvement of the CPU, so that the CPU can work with the FF subarrays in parallel.

We choose to configure one adjacent memory subarray to the FF subarrays as the Buffer subarray, which is close to both the FF subarrays and the global row buffer so as to minimize the delay. We do not utilize the local row buffer because it is not large enough to serve typical NNs. We do not implement the buffer with low-latency SRAM due to its large area and cost overhead.

3.3 Controller Design

Figure 4(E) illustrates the controller that decodes instruction and provide control signals for all the peripheral circuits in the FF subarray. A key role of the controller is to configure the FF subarrays between memory and computation modes. Table 1 lists the basic commands used by the controller. The left four commands generate control signals for the multiplexers in Figure 4, including the selection of the function of each mat among programming synaptic weights, computation, or memory, as well as the selection of the input source for computation, either from the Buffer subarray or from the output of the previous layer. The right four commands in Table 1 control the data movement. The left four commands are only performed once during the configuration of the FF subarrays. The right four data movement commands are applied during the computation phase.

Table 1: Controller Commands

Datapath Configure	**Data Flow Control**
prog/comp/mem [mat adr][0/1/2]	fetch [mem adr] to [buf adr]
bypass sigmod [mat adr] [0/1]	commit [buf adr] to [mem adr]
bypass SA [mat adr][0/1]	load [buf adr] to [FF adr]
input source [mat adr][0/1]	store [FF adr] to [buf adr]

4. SYSTEM-LEVEL DESIGN

In this section, we present the system-level design. The software-hardware interface framework is described first. Then, we focus on the optimization of NN mapping and data allocation during compile time. Next, we introduce the operating system (OS) support for switching FF subarrays between memory and computation modes at run time.

4.1 Software-Hardware Interface

Figure 5 shows the stack of the proposed design to support NN programming, which allows developer to easily configure the FF subarrays for neural network applications[1]. From software programming to hardware execution, there are three stages: programming (coding), compiling (code optimization), and code execution. In the programming stage, the proposed design provides application programming interfaces (APIs) so that they allow developers to: 1) map the topology of the NN to the FF subarrays *Map_Topology*, 2) program the synaptic weights into mats *Program_Weight*, 3) configure the data paths (peripheral circuits) of the FF subarrays *Config_Datapath*, 4) run computation *Run*, and 5) post-process the result *Post_Proc*. In this work, we assume that the NN has been off-line trained so that the inputs of each API are already known (*NN param.file*). A few previous studies have explored to implement training with ReRAM crossbar arrays [46, 47, 48, 49, 50, 51], and we plan to further enhance the proposed design with the training capability in future work.

In the compiling stage, the mapping from the NN to the FF subarrays and the input data allocation are optimized (Section 4.2). The output of compiling is the metadata for synaptic weights mapping, data path configuration, as well as execution commands with data dependency and flow control. The metadata is also the input for the execution stage. In the execution stage, the controller writes the synaptic weights to the mapped addresses in the FF subarrays; then the controller (re-)configures the peripheral circuits via the *Datapath Configure* commands (Table 1 left) to set up the data paths for computation; and finally, the controller executes *Data Flow Control* commands (Table 1 right) to manage data moving into or out of the FF subarrays at runtime.

4.2 Compile Time Optimization

4.2.1 NN Mapping Optimization

The mapping of the NN topology (from the off-line training result) to the physical ReRAM cells is optimized during compile time.[2] For different scales of NNs, we have different optimizations.

Small-Scale NN. When an NN can be mapped to a single mat (here, we only consider either the positive part or the negative part), it is small-scale. Although we can simply map a small-scale NN to some cells in one mat, the other cells in this mat may be wasted. Moreover, the speedup for very small NNs is not obvious, because the latency of the peripheral circuits may overwhelm the latency of the multiply-addition operations on ReRAM cells. Our optimization is to map the same NN to different independent portions of the mat, in order to increase the utilization of the ReRAM cells of the mat as well as improve the throughput. For example, to implement a $25-1$ NN, we duplicate it by 10 times and then map a $250-10$ NN to the target mat. Furthermore, if there is another FF mat available, we can also duplicate the mapping to the second mat, and

[1]Due to space limit, we only depict the key steps at high level while the design details of OS kernel, compiler, tool chains are left as engineering work.

[2] the proposed design supports convolution by mapping multiply-add operations to ReRAM crossbar. It also supports mean pooling of N numbers by applying the same weight $1/N$ to a linear combination of these numbers.

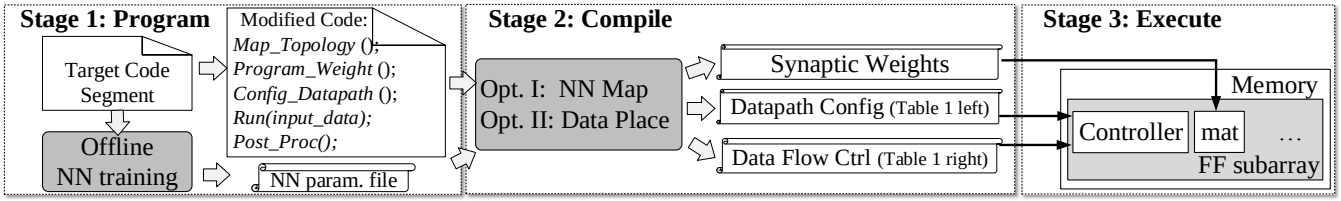

Figure 5: The software perspective: from source code to execution.

then the two mats can work simultaneously in a ping-pong manner: when the first mat is loading or storing data, the second mat is computing, and vice versa.

Medium-Scale NN. When an NN cannot be mapped to a single mat, but still can fit to several FF mats of one bank, it is medium-scale. During the mapping in compile time, a medium-scale NN has to be split into small ones so as to fit in the mats of a 256×256 size. For example, to implement a $1024-1024$ NN, it is split into 16 256×256 parts and mapped to 16 different mats. After they finish computation, their results have to be merged. In the example, 4 results will count for 1 output of the original NN. Note that due to the non-linearity of the sigmoid function, we need to finish the split-merge procedure before go to the sigmoid unit.

Large-Scale NN. A large-scale NN is one NN that cannot be mapped to a single bank. For the NN with such scale, intuitively it can be divided into small trunks and map each trunk to a bank serially. This naive solution requires reprogramming FF subarrays for each iteration, and the latency overhead of doing so will offset the benefits of the execution speedup. Alternatively, PRIME leverages multiple banks to implement the NN. Those banks can run in parallel to improve the throughput (Section 4.2.2). PRIME promises inter-bank data movements to allow communication between banks, which can be implemented by exploiting the internal bus shared across all banks [52]. The intra-bank data communication are then managed by the controller, which can handle arbitrary network connections. If the bank-level parallelism can be fully utilized, PRIME can handle a maximal single layer with $\sim1.7\times10^{10}$ synapses per chip, which is even larger than the largest NN to date (YouTube video object recognition [53], 1.7×10^{9} synapses) and the largest NN mapped on existing NPUs (TrueNorth [45], 1.4×10^{7} synapses). In Section 5, we implement *ConvNet VGG-D* [54] on PRIME, which has 1.4×10^{7} synapses and 1.7×10^{10} operations.

4.2.2 Bank-level Parallelism and Data Placement

Since FF subarrays reside in banks, the proposed design intrinsically inherits bank-level parallelism to speed up execution. Note that the FF subarrays in different banks have the same computing configurations. For instance, if a bank is considered as an NPU, the proposed design contains 64 NPUs per bank (8banks×8chips) so that 64 images can be processed in parallel. To take advantage of the bank-level parallelism, the OS is required to place one image in one bank, and evenly distribute images to all banks. As current page placement strategies expose memory latency or bandwidth information to the OS [55, 56], the proposed design further exposes the bank ID information to the OS, so that the data of each image can be mapped to a single bank.

4.2.3 A Mapping Case Study: CNN-1

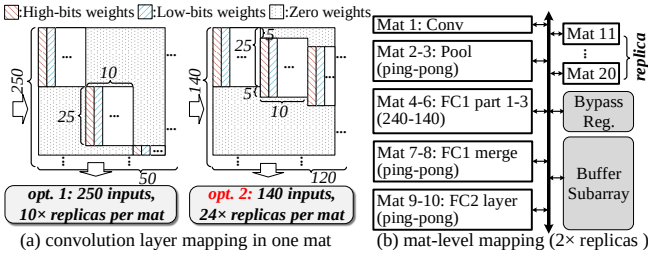

Figure 6: A Mapping Example. Left: (a) Two optimizations for the convolution layer. Right: (b) mat-level mapping and optimization.

We show how CNN-1 in Table 2 is mapped to PRIME in this section. We use 256×256 ReRAM mats, and assume that ReRAM cells in FF subarrays can be programmed in 4-bit precision and the synaptic weights of the NN are 8-bit in the application. Therefore, to present one weight, two cells are used. In CNN-1, the **convolution layer** has five 5×5 convolution kernels. Then, the weight matrix is of size $25(=5\cdot5)\times10(=5\cdot2)$. In order to improve latency and ReRAM utility, the compile-time optimization might replicate the matrix 10 times, and map a 250×100 weight matrix to a 256×256 mat. Actually, the optimization can be better for a convolution layer. Because two adjacent convolution operations share most of the inputs, e.g., 20 out of 25 inputs for a 5×5 kernel in this case, we are able to map more replicas of the 25×10 weight matrix in one mat in a smart way, as shown in Figure 6(a). In this way, we make 24 replicas in one mat, increasing the number of output data from 50 to 120 as well as reducing the number of input data from 250 to 140. In CNN-1, the **pooling layer** adopts $4:1$ max pooling. Since each pooling requires a 4×6 weight matrix, we can map 42 replicas in one mat. In CNN-1, the **fully connected layers** are $720-70-10$. Since the input dimension is larger than 256, we apply the *split-merge* mapping. First, the $720 \times 140(=70\cdot2)$ weight matrix is mapped into 3 mats, each of size 240×140. Then, the merging of the results from those three mats is performed in another mat to achieve the final results. In the merging mat, since the weight matrix is 3×1, we

can execute 85 merging operations in one mat at the same time.

In each mat, computation and data input/output can work in a pipelined way, thanks to the input latches and output registers in Figure 4 Ⓐ and Ⓒ. All the mats can work in parallel, and the data communication among them either goes through the Buffer subarray or bypasses it with the help of the bypassing registers (as shown in Figure 4 Ⓒ). When a mat has bypassed inputs (not from the Buffer subarray but from the bypassing registers) and is dedicated for a whole layer (not one of a set of mats for split-merge mapping), in order to improve the throughput, we can duplicate the mat and make the two mats work in a ping-pong mode, in which one mat receives its inputs from the previous layer while the other does the computation. As shown in Figure 6(b), the pooling layer, the merging layer of the fully connected layer 1, and the the fully connected layer 2, are configured to work in ping-pong mode. The whole system takes ten mats, Mats 1 to 10, and we duplicate it using Mats 11 to 20. We do not replicate more because with two copies the data communication latency totally hides the computation latency. More copies cannot further improve the throughput because the latency-dominated data communication is serial among mats through the bus-based Buffer subarray.

4.3 Run Time Optimization

When FF subarrays are configured for NN applications, the memory space is reserved and supervised by the OS so that it is invisible to other user applications. However, during the runtime, if none or only a few of their crossbar arrays are used for computing, and the page miss rate is higher than the predefined threshold (which indicates the memory capacity is insufficient), the OS is able to release the reserved memory addresses as normal memory. It was observed that the memory requirement varies between workloads and proposed to dynamically adjust the memory capacity by switching between SLC and MLC modes in PCM-based memory [57]. The page miss rate curve can be tracked dynamically by using either hardware or software approach [58]. In our design, the granularity to flexibly configure a range of memory addresses for either computing or memory is crossbar array (mat): when an array is configured for computing, it stores multi-bit synaptic weights; when an array is used as normal memory, it stores data as single-bit cells. The OS works with memory management unit (MMU) to keep all the mapping information of the FF subarrays, and decides when and how much reserved memory space should be released, based on the combination of the utilization of FF subarrays for computing and the page miss rate.

4.4 Discussion

Overhead of Writing Synaptic Weights. In the execution stage, before the computation starts, we need to program the target ReRAM cells according to the synaptic weights. We adopt a similar mapping engine described in prior work [59]. Programming cells to certain weights is equivalent to writing MLC ReRAM cells, which is both time- and energy-consuming. To reduce the overhead of programming, data comparison write schemes are used [60].

Endurance. PRIME does not deteriorate ReRAM's lifetime. Prior techniques that extend ReRAM's lifetime can be integrated into PRIME seamlessly. When morphed as memory, the ReRAM serves as SLC with 10^{12} [21, 22] lifetime. When morphed as accelerators, the ReRAM serves as MLC but only written at the programming step (writing synaptic weights). Given a 10-year lifetime and 10^9 MLC endurance[3], PRIME can be re-programmed every 300ms, which is far ahead of the expected programming rate that occurs daily, weekly, or even monthly.

Sneak Current and IR-Drop. When serving as memory, a ReRAM crossbar array suffers from multiple possible current paths. A lot of previous studies proposed different solutions to solve the sneak current problem at device, circuit, and architecture levels [24, 25, 26, 27, 8, 9, 10], which are adoptable in PRIME. However, when serving for computing, the sneak current does not matter, since all of the currents are part of the computation [6]. The IR-drop on a wordline is another challenge. Xu *et al.* [10] proposed a double-sided ground biasing technique to solve this problem, which is adopted in PRIME. Moreover, proper training [50] and layout engineering [62] can further address the IR-drop for ReRAM-based NN computation.

5. EVALUATION

In this section, we evaluate our design. We first describe the benchmark and experiment methodology, and then present the performance and energy results, and finally estimate the area overhead.

5.1 Experiment Setup

Benchmark. We use two sets of benchmarks in our experiments, as listed in Table 2. The first set (*AxBench*) is an approximate computing benchmark using NNs [34], and we choose five of them. The second set (*MlBench*) comprises six NN designs for machine learning applications. *CNN-1* and *CNN-2* are two CNNs, and *MLP-S/M/L* are multilayer perceptrons (MLPs) with different network scales: small, medium, and large. Those five NNs are evaluated on the widely used *MNIST* database of handwritten digits [63]. The sixth NN, ConvNet *VGG-D*, is well known for ImageNet Challenge [54]. It is an extremely large NN, containing 16 weight layers and 137.2MB synapses, and requires $\sim 1.7 \times 10^{10}$ operations.

Methodology. We compare the proposed design with several counterparts. The baseline is a CPU-only solution. We also evaluate different NPU solutions: using NPU as a co-processor (out of memory), and using NPU as a PIM-processor through 3D stacking. Because *AxBench* employs very small NNs for approximate com-

[3]MLC endurance is reported between 10^7 [61] to 10^{12} [22]. We adopt 10^9 to have a conservative lifetime estimation.

Table 2: The Benchmarks and Topologies.

MlBench	*CNN-1*	conv5x5-pool-720-70-10	AxBench	*blackscholes*	6-8-8-1
	CNN-2	conv7x10-pool-1210-120-10		*inversek2j*	2-8-2
	MLP-S	784-500-250-10		*jmeint*	18-32-8-2
	MLP-M	784-1000-500-250-10		*jpeg*	64-16-8-64
	MLP-L	784-1500-1000-500-10		*sobel*	9-8-1
	VGG-D	conv3x64-conv3x64-pool-conv3x128-conv3x128-pool conv3x256-conv3x256-conv3x256-pool-conv3x512 conv3x512-conv3x512-pool-conv3x512-conv3x512 conv3x512-pool-25088-4096-4096-1000			

puting whereas *MlBench* utilizes very large NNs for pattern recognition, we compare the proposed design with two different NPU designs for the two benchmark sets, a simple serial NPU for *AxBench* [34], and a complex parallel NPU for *MlBench* [32]. The configurations of these comparatives are described as follows.

Table 3: Configurations of CPU and Memory.

Processor	4 cores; 3GHz; Out-of-order
L1 I&D cache	Private; 32KB; 4-way; 2 cycles access;
L2 cache	Private; 2MB; 8-way; 10 cycles access;
ReRAM-based Main Memory	16GB ReRAM; 533MHz; 8 chips/rank; 8 banks/chip; tRCD-tRC-tRP-tWR 13.4-15.1-0.5-32.4 (ns)

CPU: The configurations of CPU and ReRAM based main memory are shown in Table 3, including the key memory timing parameters for simulation.

coNPU-s: This is a serial NPU co-processor, and its design is according to the NPU design for general-purpose approximate programs [34]. It contains eight processing engines (PEs) working at 500MHz. Each PE has an 8-bit integer multiply-add data path, a 128KB input FIFO queue, a 128KB output FIFO queue, a 512KB weight FIFO queue, and a 2KB Sigmoid LUT.

pimNPU-s: It has the same serial NPU design with the coNPU-s, and 64 such NPUs are 3D-stacked on top of ReRAM based main memory. Compared with coNPU-s, pimNPU-s enjoys a larger memory bandwidth through TSV. Moreover, 64 NPUs can work in parallel.

coNPU-p: This is a parallel NPU co-processor, and its design is based on the DianNao NPU design [32]. Each NPU is able to calculate a 16×16 NN at one time, through a three-stage pipeline: 1) multiplication with a 16×16 6-bit integer multiplier, 2) addition with a 256-to-1 adder tree, and 3) Sigmoid. Each NPU has a 2KB input buffer, a 2KB output buffer, and a 32KB weight buffer.

pimNPU-p: This is a PIM-processor with 64 parallel NPUs 3D-stacked on top of ReRAM based main memory.

We modeled the above NPU designs using Synopsys Design Compiler and PrimeTime with 65nm TSMC CMOS library. We also modeled ReRAM based main memory and our system with modified NVSim [64], CACTI-3DD [65] and CACTI-IO [66]. We adopt Pt/TiO2-x/Pt devices [67] with $R_{\text{on}}/R_{\text{off}} = 1k\Omega/20k\Omega$ and 2V SET/RESET voltage. The FF subarray is modeled by heavily modified NVSim, according to peripheral

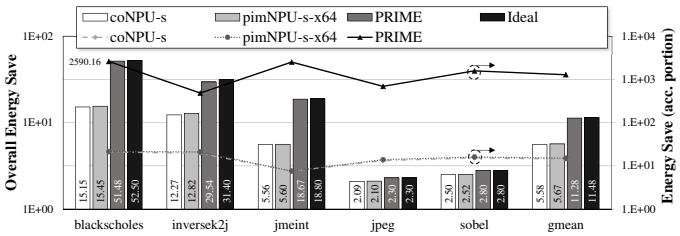

Figure 7: The Energy Saving Results of *AxBench* (*vs.* CPU).

circuit modifications, i.e., write driver [68] (serves as DAC), sigmoid [69], and sense amplifier [70] (serves as ADC) circuits. We built a trace-based in-house simulator to evaluate different systems, including CPU-only, PRIME, NPU co-processor, and NPU PIM-processor.

5.2 Energy Results

The overall energy saving for *AxBench* are presented in Figure 7 (left). All the NPU solutions and the proposed design are much more energy-efficient than the CPU-only solution. The proposed design almost reaches the ideal energy saving results assuming that the accelerated portions do not consume any energy at all. Moreover, the proposed design has about twice the energy saving of pimNPU-s on average. coNPU-s is almost as energy-efficient as pimNPU-s, because *AxBench* does not benefit from PIM due to its some memory footprint. Figure 7 (right) shows the energy saving results on the accelerated portions of *AxBench*. We find that using ReRAM for neural computation is much more energy-efficient than logic (NPU) solutions. On average, the proposed design saves more than 80× energy than coNPU-s and pimNPU-s in neural computations across *AxBench*.

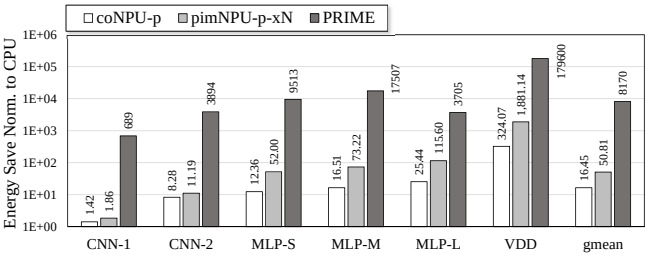

Figure 8: The Energy Saving Results on *MlBench* (*vs.* CPU).

5.3 Area Overhead

Given 1/16 of a bank is FF subarrays, the proposed design only incurs 3.77% area overhead, which is much smaller than pimNPU-s (6.37%) and pimNPU-p (35.09%), respectively. There are 39% area added-on for supporting the computing: increased driver takes 6%, subtraction and sigmoid circuit take 17%, and controlling, multiplexer, etc. cost 16%.

6. CONCLUSION

The proposed design explores the unique feature of ReRAM – having data storage and neural computation capabilities in a single device. In the proposed design, part of the ReRAM memory arrays is enabled for neural computations. They can either perform computation to accelerate neural network applications or serve as memory to provide a larger working memory space (hence we call it *morphable PIM structure*). We also provide circuit-level and system-level designs to support the proposed design for various tasks. With circuits reuse, the proposed design incurs only 3.77% area overhead to the original ReRAM chips. The experiment results show that, for approximate computing benchmarks, the proposed design reduces the execution time by ∼9× and saves the energy by ∼1000× compared with a CPU-only solution; and for machine learning benchmarks, it achieves a ∼9000× speedup and ∼4000× energy saving.

7. REFERENCES

[1] A. Farmahini-Farahani, J. H. Ahn, K. Morrow, and N. S. Kim, "Nda: Near-dram acceleration architecture leveraging commodity dram devices and standard memory modules," in *IEEE 21st International Symposium on High Performance Computer Architecture (HPCA),*, pp. 283–295, Feb 2015.

[2] H.-S. Wong, H.-Y. Lee, S. Yu, Y.-S. Chen, Y. Wu, P.-S. Chen, B. Lee, F. Chen, and M.-J. Tsai, "Metal-oxide rram," *Proceedings of the IEEE*, vol. 100, pp. 1951–1970, June 2012.

[3] M. Hu, H. Li, Q. Wu, and G. Rose, "Hardware realization of bsb recall function using memristor crossbar arrays," in *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, pp. 498–503, June 2012.

[4] B. Li, Y. Shan, M. Hu, Y. Wang, Y. Chen, and H. Yang, "Memristor-based approximated computation," in *Low Power Electronics and Design (ISLPED), 2013 IEEE International Symposium on*, pp. 242–247, Sept 2013.

[5] M. Prezioso, F. Merrikh-Bayat, B. Hoskins, G. Adam, K. K. Likharev, and D. B. Strukov, "Training and operation of an integrated neuromorphic network based on metal-oxide memristors," *CoRR*, vol. abs/1412.0611, 2014.

[6] P. Gu, B. Li, T. Tang, S. Yu, Y. Cao, Y. Wang, and H. Yang, "Technological exploration of rram crossbar array for matrix-vector multiplication," in *Design Automation Conference (ASP-DAC), 2015 20th Asia and South Pacific*, pp. 106–111, Jan 2015.

[7] Y. Kim, Y. Zhang, and P. Li, "A reconfigurable digital neuromorphic processor with memristive synaptic crossbar for cognitive computing," *J. Emerg. Technol. Comput. Syst.*, vol. 11, pp. 38:1–38:25, Apr. 2015.

[8] M. Jung, J. Shalf, and M. Kandemir, "Design of a large-scale storage-class rram system," in *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*, ICS '13, (New York, NY, USA), pp. 103–114, ACM, 2013.

[9] C. Xu, P.-Y. Chen, D. Niu, Y. Zheng, S. Yu, and Y. Xie, "Architecting 3d vertical resistive memory for next-generation storage systems," in *Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '14, (Piscataway, NJ, USA), pp. 55–62, IEEE Press, 2014.

[10] C. Xu, D. Niu, N. Muralimanohar, R. Balasubramonian, T. Zhang, S. Yu, and Y. Xie, "Overcoming the challenges of crossbar resistive memory architectures," in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, pp. 476–488, Feb 2015.

[11] Taha, T.M. and Hasan, R. and Yakopcic, C. and McLean, M.R., "Exploring the design space of specialized multicore neural processors," in *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, Aug 2013.

[12] D. A. Patterson and M. D. Smith, "First workshop on mixing logic and dram: Chips that compute and remember," 1997.

[13] J. Draper, J. Chame, M. Hall, C. Steele, T. Barrett, J. LaCoss, J. Granacki, J. Shin, C. Chen, C. W. Kang, *et al.*, "The architecture of the diva processing-in-memory chip," in *Proceedings of the 16th international conference on Supercomputing*, pp. 14–25, ACM, 2002.

[14] A. De, M. Gokhale, R. Gupta, and S. Swanson, "Minerva: Accelerating data analysis in next-generation ssds," in *Field-Programmable Custom Computing Machines (FCCM), 2013 IEEE 21st Annual International Symposium on*, pp. 9–16, IEEE, 2013.

[15] P. Dlugosch, D. Brown, P. Glendenning, M. Leventhal, and H. Noyes, "An efficient and scalable semiconductor architecture for parallel automata processing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 25, pp. 3088–3098, Dec 2014.

[16] S. H. Pugsley, J. Jestes, H. Zhang, R. Balasubramonian, V. Srinivasan, A. Buyuktosunoglu, A. Davis, and F. Li, "NDC: analyzing the impact of 3d-stacked memory+logic devices on mapreduce workloads," in *2014 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2014, Monterey, CA, USA, March 23-25, 2014*, pp. 190–200, 2014.

[17] R. Balasubramonian, J. Chang, T. Manning, J. H. Moreno, R. Murphy, R. Nair, and S. Swanson, "Near-data processing: Insights from a micro-46 workshop," *Micro, IEEE*, vol. 34, pp. 36–42, July 2014.

[18] F. Alibart, T. Sherwood, and D. Strukov, "Hybrid cmos/nanodevice circuits for high throughput pattern matching applications," in *Adaptive Hardware and Systems (AHS), 2011 NASA/ESA Conference on*, pp. 279–286, June 2011.

[19] Q. Guo, X. Guo, Y. Bai, and E. İpek, "A resistive tcam accelerator for data-intensive computing," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-44, (New York, NY, USA), pp. 339–350, ACM, 2011.

[20] Q. Guo, X. Guo, R. Patel, E. Ipek, and E. G. Friedman, "Ac-dimm: Associative computing with stt-mram," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, (New York, NY, USA), pp. 189–200, ACM, 2013.

[21] Lee, Myoung-Jae and Lee, Chang Bum and Lee, Dongsoo and Lee, Seung Ryul and Chang, Man and Hur, Ji Hyun and Kim, Young-Bae and Kim, Chang-Jung and Seo, David H. and Seo, Sunae and Chung, U-In and Yoo, In-Kyeong and Kim, Kinam, "A fast, high-endurance and scalable non-volatile memory device made from asymmetric Ta2O5-x/TaO2-x bilayer structures," *Nature Materials*, vol. 10, no. 8, pp. 625–630, 2011.

[22] Chung-Wei Hsu and I-Ting Wang and Chun-Li Lo and Ming-Chung Chiang and Wen-Yueh Jang and Chen-Hsi Lin and Tuo-Hung Hou, "Self-rectifying bipolar TaOx/TiO2 RRAM with superior endurance over 10^12 cycles for 3D high-density storage-class memory," in *VLSI Technology (VLSIT), 2013 Symposium on*, pp. T166–T167, June 2013.

[23] Moinuddin K. Qureshi and John Karidis and Michele Franceschini and Vijayalakshmi Srinivasan and Luis Lastras and Bulent Abali, "Enhancing lifetime and security of PCM-based main memory with Start-Gap wear leveling," in *Proc. of the 42nd annual IEEE/ACM international symposium on microarchitecture (MICRO'09)*, pp. 14–23, 2009.

[24] D. Niu, C. Xu, N. Muralimanohar, N. P. Jouppi, and Y. Xie, "Design trade-offs for high density cross-point resistive memory," in *Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design*, ISLPED '12, (New York, NY, USA), pp. 209–214, ACM, 2012.

[25] Yang, Yuanfan and Mathew, J. and Ottavi, M. and Pontarelli, S. and Pradhan, D.K., "Novel Complementary Resistive Switch Crossbar Memory Write and Read Schemes," *IEEE Transactions on Nanotechnology*, vol. 14, pp. 346–357, March 2015.

[26] A. Kawahara, R. Azuma, Y. Ikeda, K. Kawai, Y. Katoh, K. Tanabe, T. Nakamura, Y. Sumimoto, N. Yamada, N. Nakai, S. Sakamoto, Y. Hayakawa, K. Tsuji, S. Yoneda, A. Himeno, K. Origasa, K. Shimakawa, T. Takagi, T. Mikawa, and K. Aono, "An 8mb multi-layered cross-point reram macro with 443mb/s write throughput," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International*, pp. 432–434, Feb 2012.

[27] T. yi Liu, T. H. Yan, R. Scheuerlein, Y. Chen, J. Lee, G. Balakrishnan, G. Yee, H. Zhang, A. Yap, J. Ouyang, T. Sasaki, S. Addepalli, A. Al-Shamma, C.-Y. Chen, M. Gupta, G. Hilton, S. Joshi, A. Kathuria, V. Lai, D. Masiwal, M. Matsumoto, A. Nigam, A. Pai, J. Pakhale, C. H. Siau, X. Wu, R. Yin, L. Peng, J. Y. Kang, S. Huynh, H. Wang, N. Nagel, Y. Tanaka, M. Higashitani, T. Minvielle, C. Gorla, T. Tsukamoto, T. Yamaguchi, M. Okajima, T. Okamura, S. Takase, T. Hara, H. Inoue, L. Fasoli, M. Mofidi, R. Shrivastava, and K. Quader, "A 130.7mm2 2-layer 32gb reram memory device in 24nm technology," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2013 IEEE International*, pp. 210–211, Feb 2013.

[28] S. Yu, H.-Y. Chen, Y. Deng, B. Gao, Z. Jiang, J. Kang, and H.-S. Wong, "3d vertical rram - scaling limit analysis and demonstration of 3d array operation," in *VLSI Technology (VLSIT), 2013 Symposium on*, pp. T158–T159, June 2013.

[29] S. Yu, Y. Wu, and H. Wong, "Investigating the switching dynamics and multilevel capability of bipolar metal oxide resistive switching memory," *Applied Physics Letters*, vol. 98, p. 103514, 2011.

[30] M.-C. Wu, W.-Y. Jang, C.-H. Lin, and T.-Y. Tseng, "A study on low-power, nanosecond operation and multilevel bipolar resistance switching in ti/zro2/pt nonvolatile memory with 1t1r architecture," *Semiconductor Science and Technology*, vol. 27, p. 065010, 2012.

[31] 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, p. 075201, 2012.

[32] 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 *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, (New York, NY, USA), pp. 269–284, ACM, 2014.

[33] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "Dadiannao: A machine-learning supercomputer," in *Microarchitecture (MICRO), 2014 47th Annual IEEE/ACM International Symposium on*, pp. 609–622, Dec 2014.

[34] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural acceleration for general-purpose approximate programs," in *Microarchitecture (MICRO), 2012 45th Annual IEEE/ACM International Symposium on*, pp. 449–460, Dec 2012.

[35] R. St. Amant, A. Yazdanbakhsh, J. Park, B. Thwaites, H. Esmaeilzadeh, A. Hassibi, L. Ceze, and D. Burger, "General-purpose code acceleration with limited-precision analog computation," in *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ISCA '14, (Piscataway, NJ, USA), pp. 505–516, IEEE Press, 2014.

[36] T. Moreau, M. Wyse, J. Nelson, A. Sampson, H. Esmaeilzadeh, L. Ceze, and M. Oskin, "SNNAP: approximate computing on programmable socs via neural acceleration," in *21st IEEE International Symposium on High Performance Computer Architecture, HPCA 2015, Burlingame, CA, USA, February 7-11, 2015*, pp. 603–614, 2015.

[37] D. C. Cireşan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," in *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, IJCAI'11, pp. 1237–1242, AAAI Press, 2011.

[38] J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, CVPR '12, (Washington, DC, USA), pp. 3642–3649, IEEE Computer Society, 2012.

[39] A. Coates, B. Huval, T. Wang, D. J. Wu, B. C. Catanzaro, and A. Y. Ng, "Deep learning with cots hpc systems.," in *ICML (3)*, vol. 28 of *JMLR Proceedings*, pp. 1337–1345, JMLR.org, 2013.

[40] S. Sahin, Y. Becerikli, and S. Yazici, "Neural network implementation in hardware using fpgas," in *Neural Information Processing* (I. King, J. Wang, L.-W. Chan, and D. Wang, eds.), vol. 4234 of *Lecture Notes in Computer Science*, pp. 1105–1112, Springer Berlin Heidelberg, 2006.

[41] C. Farabet, C. Poulet, J. Han, and Y. LeCun, "Cnp: An fpga-based processor for convolutional networks," in *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, pp. 32–37, Aug 2009.

[42] J.-Y. Kim, M. Kim, S. Lee, J. Oh, K. Kim, and H.-J. Yoo, "A 201.4 gops 496 mw real-time multi-object recognition processor with bio-inspired neural perception engine," *Solid-State Circuits, IEEE Journal of*, vol. 45, pp. 32–45, Jan 2010.

[43] P. Merolla, J. Arthur, F. Akopyan, N. Imam, R. Manohar, and D. Modha, "A digital neurosynaptic core using embedded crossbar memory with 45pj per spike in 45nm," in *Custom Integrated Circuits Conference (CICC), 2011 IEEE*, pp. 1–4, Sept 2011.

[44] J. Seo, B. Brezzo, Y. Liu, B. Parker, S. Esser, R. Montoye, B. Rajendran, J. Tierno, L. Chang, D. Modha, and D. Friedman, "A 45nm cmos neuromorphic chip with a scalable architecture for learning in networks of spiking neurons," in *Custom Integrated Circuits Conference (CICC), 2011 IEEE*, pp. 1–4, Sept 2011.

[45] Esser, S.K. and Andreopoulos, A. and Appuswamy, R. and Datta, P. and Barch, D. and Amir, A. and Arthur, J. and Cassidy, A. and Flickner, M. and Merolla, P. and Chandra, S. and Basilico, N. and Carpin, S. and Zimmerman, T. and Zee, F. and Alvarez-Icaza, R. and Kusnitz, J.A. and Wong, T.M. and Risk, W.P. and McQuinn, E. and Nayak, T.K. and Singh, R. and Modha, D.S., "Cognitive computing systems: Algorithms and applications for networks of neurosynaptic cores," in *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–10, Aug 2013.

[46] 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.

[47] M. Hu, H. Li, Y. Chen, Q. Wu, and G. S. Rose, "Bsb training scheme implementation on memristor-based circuit," in *Computational Intelligence for Security and Defense Applications (CISDA), 2013 IEEE Symposium on*, pp. 80–87, IEEE, 2013.

[48] B. Li, Y. Wang, Y. Wang, Y. Chen, and H. Yang, "Training itself: Mixed-signal training acceleration for memristor-based neural network," in *ASP-DAC*, pp. 361–366, 2014.

[49] B. Liu, M. Hu, H. Li, Z.-H. Mao, Y. Chen, T. Huang, and W. Zhang, "Digital-assisted noise-eliminating training for memristor crossbar-based analog neuromorphic computing engine," in *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*, pp. 1–6, IEEE, 2013.

[50] B. Liu, H. Li, Y. Chen, X. Li, T. Huang, Q. Wu, and M. Barnell, "Reduction and ir-drop compensations techniques for reliable neuromorphic computing systems," in *Computer-Aided Design (ICCAD), 2014 IEEE/ACM International Conference on*, pp. 63–70, Nov 2014.

[51] M. Prezioso, F. Merrikh-Bayat, B. Hoskins, G. Adam,

K. K. Likharev, and D. B. Strukov, "Training and operation of an integrated neuromorphic network based on metal-oxide memristors," *Nature*, 2014.

[52] V. Seshadri, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Rowclone: Fast and energy-efficient in-dram bulk data copy and initialization," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-46, (New York, NY, USA), pp. 185–197, ACM, 2013.

[53] Coates, Adam and Huval, Brody and Wang, Tao and Wu, David and Catanzaro, Bryan and Andrew, Ng, "Deep learning with COTS HPC systems," in *Proceedings of the 30th international conference on machine learning*, pp. 1337–1345, 2013.

[54] Simonyan, Karen and Zisserman, Andrew, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *Proceedings of the International Conference on Learning Representations (ICLR)*, pp. 1–14, May 2015.

[55] B. Verghese, S. Devine, A. Gupta, and M. Rosenblum, "Operating system support for improving data locality on cc-numa compute servers," in *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS VII, (New York, NY, USA), pp. 279–289, ACM, 1996.

[56] N. Agarwal, D. Nellans, M. Stephenson, M. O'Connor, and S. W. Keckler, "Page placement strategies for gpus within heterogeneous memory systems," in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '15, (New York, NY, USA), pp. 607–618, ACM, 2015.

[57] M. K. Qureshi, M. M. Franceschini, L. A. Lastras-Montaño, and J. P. Karidis, "Morphable memory system: A robust architecture for exploiting multi-level phase change memories," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, (New York, NY, USA), pp. 153–162, ACM, 2010.

[58] P. Zhou, V. Pandey, J. Sundaresan, A. Raghuraman, Y. Zhou, and S. Kumar, "Dynamic tracking of page miss ratio curve for memory management," in *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XI, (New York, NY, USA), pp. 177–188, ACM, 2004.

[59] Hu, Miao and Strachan, John Paul and Yang, J. Joshua and Merced-Grafals, Emmanuelle and Graves, Catherine and Ge, Ning and Lam, Si-Ty and Montgomery, Eric and Li, Zhiyong and Williams, R. Stanley, "Dot-Product Engine for Neuromorphic Computing: Programming 1T1M Crossbars for Efficient Vector-Matrix Multiplication," *Tech Reports*, pp. HPL–2015–55, 2015.

[60] Sangyeun Cho and Hyunjin Lee, "Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance," in *42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 347–357, Dec 2009.

[61] Seung Ryul Lee and Young-Bae Kim and others, "Multi-level switching of triple-layered TaOx RRAM with excellent reliability for storage class memory," in *VLSI Technology (VLSIT), 2012 Symposium on*, pp. 71–72, June 2012.

[62] P.-Y. Chen, D. Kadetotad, Z. Xu, A. Mohanty, B. Lin, J. Ye, S. Vrudhula, J.-s. Seo, Y. Cao, and S. Yu, "Technology-design co-optimization of resistive cross-point array for accelerating learning algorithms on chip," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pp. 854–859, 2015.

[63] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, Nov 1998.

[64] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 31, no. 7, pp. 994–1007, 2012.

[65] K. Chen, S. Li, N. Muralimanohar, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, "Cacti-3dd: Architecture-level modeling for 3d die-stacked dram main memory," in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 33–38, EDA Consortium, 2012.

[66] N. P. Jouppi, A. B. Kahng, N. Muralimanohar, and V. Srinivas, "Cacti-io: Cacti with off-chip power-area-timing models," in *Proceedings of the International Conference on Computer-Aided Design*, pp. 294–301, ACM, 2012.

[67] Gao, Ligang and Alibart, Fabien and Strukov, Dmitri B, "A High Resolution Nonvolatile Analog Memory Ionic Devices," in *4th Annual Non-Volatile Memories Workshop, NVMW 2013*, pp. paper–57, 2013.

[68] Xu, Cong and Niu, Dimin and Muralimanohar, Naveen and Jouppi, Norman P and Xie, Yuan, "Understanding the trade-offs in multi-level cell ReRAM memory design," in *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*, pp. 1–6, IEEE, 2013.

[69] Li, B. and Gu, P. and Shan, Y. and Wang, Y. and Chen, Y. and Yang, H., "RRAM-based Analog Approximate Computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. PP, pp. 1–1, 6 2015.

[70] J. Li, C.-I. Wu, S. C. Lewis, J. Morrish, T.-Y. Wang, R. Jordan, T. Maffitt, M. Breitwisch, A. Schrott, R. Cheek, *et al.*, "A novel reconfigurable sensing scheme for variable level storage in phase change memory," in *Memory Workshop (IMW), 2011 3rd IEEE International*, pp. 1–4, IEEE, 2011.

[71] Chang, Meng-Fan and Wu, Jui-Jen and Chien, Tun-Fei and Liu, Yen-Chen and Yang, Ting-Chin and Shen, Wen-Chao and King, Ya-Chin and Lin, Chorng-Jung and Lin, Ku-Feng and Chih, Yu-Der and Natarajan, S. and Chang, J., "Embedded 1Mb ReRAM in 28nm CMOS with 0.27-to-1V read using swing-sample-and-couple sense amplifier and self-boost-write-termination scheme," in *Proceedings of the International Solid-State Circuits Conference (ISSCC)*, pp. 332–333, Feb 2014.