Energy-Efficient Adaptive Computing With Multifunctional Memory

Wenchao Qian, Pai-Yu Chen, Robert Karam, Ligang Gao, Swarup Bhunia, and Shimeng Yu

***Abstract*—Digital memory arrays, which serve as an integral part of modern computing systems, are traditionally used for information storage. However, recent reports show that memory can be used on demand as a reconfigurable computing resource, drastically improving energy efficiency for many applications. In this case, memory usage is limited to storing function responses as multi-input multi-output lookup tables. In this brief, we propose a novel multifunctional memory (MFM) framework, which can function as typical memory for storage as well as in a neuroinspired computing mode. The system is based on a modified memory array, which can be dynamically switched between these two modes. Using a promising emerging memory device, namely, resistive random access memory, we present device-level engineering, circuit-level modifications, and appropriate architecture to realize the MFM framework. Simulation results demonstrate significant improvements in both energy efficiency and performance compared to a general-purpose processor, a field-programmable gate array, and a recent memory-based, reconfigurable computing framework (MAHA) for a set of common application kernels.**

***Index Terms*—Field-programmable gate array (FPGA), general-purpose processor (GPP), malleable hardware (MAHA), multifunctional memory (MFM), neuromorphic computing, resistive random access memory (RRAM), static random access memory (SRAM).**

I. Introduction

MODERN computing platforms have seen an increase in novel computing schemes to meet the demands of data-intensive applications. Increasing memory capacity and computing bandwidth while keeping energy consumption low is a critical design challenge for these applications [1]. Recently, focus has shifted toward offloading computations to the memory, or computing in memory, where the memory array is used on demand as a reconfigurable computing resource. This is an attractive option because it offers reduced data transfer between the memory and processing units; the need for back-and-forth data transfer, known as the Von Neumann bottleneck, can be vastly reduced, leading to large improvements in energy efficiency.

In recent years, dramatic advances in emerging nanoscale nonvolatile memories (NVMs) have made computing in memory even more attractive. Among the emerging NVMs, resistive random access memory (RRAM) demonstrates promising properties, including low programming voltage, fast switching speed, high on/off ratio, excellent scalability, reasonable programming endurance, high data retention, and compatibility with silicon CMOS fabrication processes [2]. Hence, a computing-in-memory framework based on RRAM is a potential solution for data-intensive applications.

In this brief, we propose a novel **M**ulti**F**unctional **M**emory (MFM) framework, which supports data storage as well as neuromorphic computing in the same memory array. To the best of our knowledge, this is the first such framework that uses memory for both traditional storage and neuromorphic computing, with on-demand switching between the two modes. The MFM framework is built into an RRAM array and paired with the requisite peripheral logic, which makes it *malleable* by reusing the memory for multiple purposes. Compared to alternative computing systems, MFM brings the computation closer to the storage, mitigating the Von Neumann bottleneck while simultaneously helping improve the system performance, energy, and reliability. Many other emerging NVM devices such as phase change memory (PCM) or spin-transfer-torque memory (STT-RAM) are also amenable for computing in memory, and the techniques proposed in this brief can be similarly applied to these NVMs. We focus on using RRAM technology because it consumes less write energy than PCM, and it has higher integration density than binary STT-RAM since it allows multiple states per cell. In particular, this brief makes the following contributions.

1) It proposes a novel MFM framework using an RRAM array hybridized with CMOS peripheral logic. It presents device-level engineering of RRAM for enhanced reliability, and multilevel states that enable the two operation modes.
2) It describes appropriate circuit-level modifications and compares the proposed architecture with conventional stands for static random access memory (SRAM)-based solutions using chip macrosimulation results. It explains the system-level microarchitecture and the basic processing element (PE). It also describes the associated software framework for mapping applications into the MFM-based PE array.
3) By using realistic models of the hardware components and custom application mapping software, it evaluates the MFM platform. It describes the simulation setup and system-level results for a set of kernels. MFM demonstrates significant improvement in performance and energy efficiency compared to a general-purpose processor (GPP), a conventional field-programmable gate array (FPGA), and a MAHA, a recent memory-centric coarse-grain reconfigurable computing framework [3].

Manuscript received December 8, 2015; revised April 2, 2016; accepted April 11, 2016. Date of publication April 15, 2016; date of current version January 27, 2017. This work was supported in part by Semiconductor Research Corp. Grant 2015-EP-2650 and NSF-CCF-1552687. This brief was recommended by Associate Editor Jose G. Delgado-Frias.

W. Qian is with Case Western Reserve University, Cleveland, OH 44106 USA.

P.-Y. Chen, L. Gao, and S. Yu are with Arizona State University, Tempe, AZ 85281 USA (e-mail: pchen72@asu.edu).

R. Karam and S. Bhunia are with University of Florida, Gainesville, FL 32611 USA.

Color versions of one or more of the figures in this brief are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSII.2016.2554958

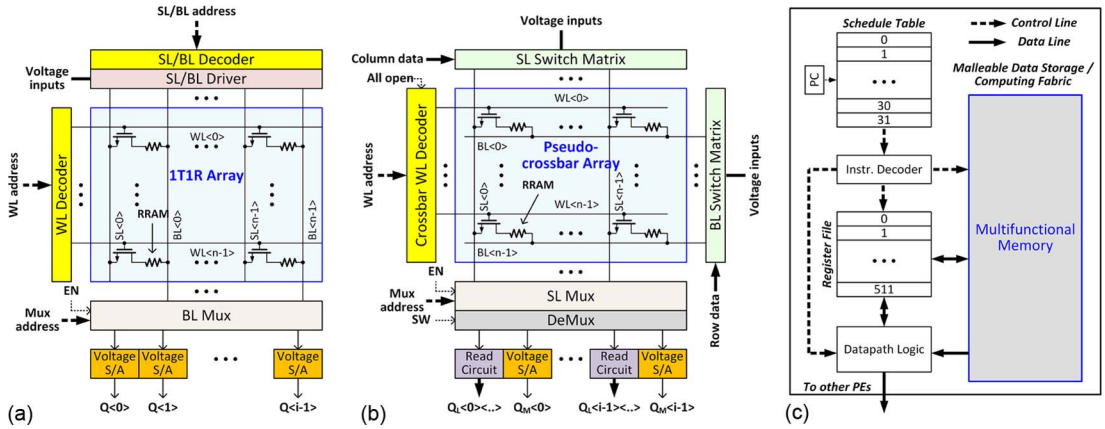

Fig. 1. Circuit block diagram of (a) the conventional 1T1R memory array and (b) the MFM with pseudocrossbar array. (c) System block diagram of one PE in the MFM framework.

The rest of this brief is organized as follows. Section II provides an overview of the MFM framework. Section III describes a case study of the proposed architecture using RRAM, including the device, circuit, and system-level details, as well as discussion on simulation results for performance, area, and power. Finally, Section IV concludes this brief.

II. Multifunctional Memory

Generally, NVM array architectures can be classified into two types: the crossbar and the 1-transistor–1-resistor (1T1R). The crossbar array consists of perpendicular bit lines (BLs) and word lines (WLs), with the NVM cells sandwiched in between. It can achieve high integration density ($4F^2$/bit, where F is the lithography feature size) but inevitably suffers from interference between cells and sneak paths, which degrade the write/read margin and increase power consumption [4]. In the 1T1R array, each cell is in series with a transistor for isolation from other unselected cells. Although the transistor increases the cell area to ($6 \sim 12F^2$/bit), it can prevent disturbing unselected cells. The conventional 1T1R array architecture for memory application is shown in Fig. 1(a). To write a cell in the conventional 1T1R array, the WL voltage is applied to turn on the transistor of the selected cell, and either the BL or source line (SL) voltage is applied with the other one grounded, depending on what value ("0" or "1") is to be written. The cell read operation is very similar but uses a lower voltage, allowing the resistance state to be read by the voltage sense amplifier (S/A). In this brief, we propose modifying the conventional 1T1R-based architecture to an MFM fabric that can also realize neuromorphic computing, as illustrated in Fig. 1(b). The neuromorphic computing in this brief refers to the weighted sum operation (vector-matrix multiplication) generally used in learning algorithms. The conductance of each array cell represents the weight, and the entire MFM array can be considered as a weight matrix. The vector is provided as a set of parallel voltage signals to the weight matrix, and the weighted sum currents can be obtained at the MFM output.

A. Circuit-Level Realization

To enable the weighted sum operation, we rotate the BLs by 90° from the conventional 1T1R array. We name this architecture the "pseudocrossbar array" because the transistors can be "transparent" when all WLs are turned on. Thus, the vectors are provided to the BLs, and the weighted sums are read out through SLs in parallel. The pseudocrossbar array can also function as a conventional memory when only one WL is enabled, giving us two independent modes of operation: the memory mode and the computing mode. To facilitate this, some changes in the peripheral circuits are required: 1) The switch matrix is used to enable multiple SLs and BLs of inputs to perform the neurocomputing operations, while it also supports operation decoding in the memory mode; 2) a read circuit [5] is used to convert the analog weighted sum current to a digital output in the computing mode; and 3) a demultiplexer is used to select the read circuit for the computing mode, or the voltage S/A for the memory mode. To summarize the proposed MFM framework, for the read operation in memory mode, the WL decoder selects one row of the array, and the SL multiplexer selects the columns that are to be read, with SL voltages being precharged with read voltage. The demultiplexer then selects the voltage S/A. For the read operation in the neurocomputing mode, the WL decoder selects all the rows of the array. All the SLs are virtually grounded by the read circuit, and the input vector from the BL switch matrix generates the weighted sum current that flows to the read circuit, where it is converted to a digital output. On the other hand, the write operation is similar in both modes. The WL decoder selects one row of the array, and the SL multiplexer selects the columns that are to be written, with SL/BL biased with write voltages. As the weight is usually an analog value, the weight update in the neurocomputing mode essentially fine-tunes the multilevel RRAM resistance states and thus requires much shorter write pulses than the conventional write operation in memory mode.

B. System-Level Architecture

Fig. 1(c) shows the system level architecture. Each PE consists of the following components. (1) *Data memory*: A 256 × 256 MFM array stores data in memory mode and computes in neuromorphic mode. (2) *Program counter*: tracks the instruction execution. (3) *Register file*: stores the intermediate inputs and outputs from the PE and uses 512 × 8-b registers to hold the 256 row and 256 column inputs to the MFM array. (4) *Data path logic*: determines the output destinations, with the output

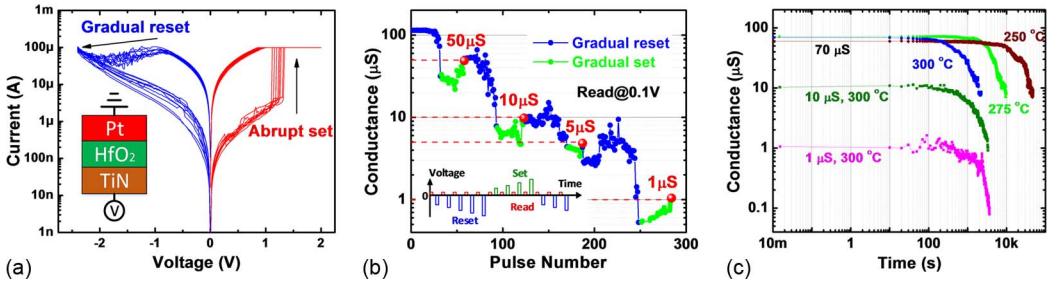

Fig. 2. Typical I–V characteristics of $Pt/HfO_2/TiN$ RRAM devices. The inset is the device structure. (b) Device weight tuning by iterative SET/RESET pulse programming. (c) Retention test of different device weights at very high temperatures.

TABLE I
CIRCUIT-LEVEL PERFORMANCE FOR DIFFERENT ARRAY ARCHITECTURES AT 32-nm TECHNOLOGY NODE

Architecture (array size = 256 x 256)	Area (μm^2)	Read Latency (ns)	Read Energy (pJ)	Write Latency (ns)	Write Energy (pJ)	Leakage (μW)
MFM (Memory Mode)	5683.86	1.96	2.03	5.56	4.37	10.56
MFM (Computing Mode)	5683.86	79.91	1993.78	10311.42	15215.2	11.18
1T1R Array for Memory	3437.63	2.34	0.53	6.19	2.41	7.66
Pseudo-crossbar Array for Computing	5601.04	75.51	1842.09	10311.42	15215.2	11.17
SRAM Array for Memory	9984.08	1.31	0.57	0.71	0.33	814.28
SRAM Array for Computing	39638.07	393.38	15143.86	114.55	1898.48	3247.93

(either the local register file or communicated to other PEs). (5) *Schedule table*: stores the instructions that implement the given application. A complete instruction set architecture has been created. Each instruction is 545 b wide, including a 2-b opcode (computing mode, memory mode, or no-op), 1 b to indicate a read or write operation, an 8-b WL identifier, 512 b for the status of the 256 BLs and 256 SLs, 16 b for the data source and destination, and 6 b for the mux and data path control. There are 32 entries, which, given the size and power of each instruction, is sufficient for most applications. (6) *Instr. decoder*: controls component operation.

The system uses a spatiotemporal computing model. PEs are spatially distributed and can communicate to each other through time-multiplexed interconnects [3]. The execution inside each PE is done temporally, executing one instruction per clock cycle. The required data inputs for memory operation come from the register file. In computing mode, outputs from the memory array are written to the register file, whereas in memory mode, data read from the memory are written to the register file or sent to other PEs as needed.

A custom software application mapping tool was also developed to statically schedule the operations for each application. The input to the mapping software is the data-flow-graph description of the application. Hardware specifications, such as the maximum number of PEs, the memory size, and the size of the register file, are also required to set the resource constraints for application mapping. The mapping tool schedules the operations based on data dependence and packs them into PEs under resource constraints. The placement and routing processes are performed to ensure correct communication between PEs. Finally, the tool reports the number of cycles, operations by type, and utilized PEs for the input application. Latency, energy, area, and leakage estimates are calculated based on the mapping results.

III. CASE STUDY WITH RRAM-BASED MFM

A. Device Engineering

At the device level, we fabricate $Pt/HfO_2/TiN$ RRAM devices to validate the feasibility of RRAM for both memory and computing modes. Fig. 2(a) shows typical I–V characteristics of the fabricated RRAM devices. As mentioned earlier, the computing mode usually requires multilevel states in RRAM to represent analog weight values rather than only the binary ON and OFF states in the memory mode. We have developed a programming protocol to optimize the weight tuning process in RRAM [6]. As shown in Fig. 2(b), fast convergence on the desired weight targets can be reached by iteratively applying a gradually increasing SET/RESET voltage pulse sequence. The reliability of RRAM is also an important characteristic for both modes. As shown in Fig. 2(c), the retention of different resistance states at 300 °C has a similar failure trend. The temperature-dependent measurement was also performed to extract the activation energy of the failure ($E_a = 1.7$ eV), indicating a lifetime of ~70 years at 125 °C by the Arrhenius extrapolation. The read fluctuation in resistance was also shown to be $< 10\%$ [6], which makes RRAM with multilevel states (4 b) feasible with a high on/off ratio of 100.

B. Circuit Simulation

We have developed a circuit-level macrosimulator to estimate the area, latency, dynamic energy, and leakage power consumption of different memory array architectures for both memory and computing modes. The framework of our simulator follows the principles of CACTI [7] for the SRAM cache and NVSim [8] for NVM, and our simulator was designed to support both memory and computing modes of MFM. The hierarchy of the simulator consists of different levels of abstraction from the memory cell parameters and transistor technology parameters to the gate-level circuit modules and the array architecture with peripheral circuits. Table I shows the performance benchmarks of different array architectures at the 32-nm technology node with an array size of 256 × 256. The proposed RRAM-based MFM has a comparable area to the pseudocrossbar array for the computing mode only but has a larger area than the 1T1R array for the memory mode only because the read circuit and switch matrix that enable the weighted sum and weight update for computing occupy more area than the simple voltage S/A and decoder in the memory mode. The latency and dynamic

energy are calculated based on one read and write operation in the memory mode, or one operation of weighted sum and weight update in computing mode. For one write operation, we assume a single row access of 1 B in memory mode or multiple row-by-row accesses in computing mode. In computing mode, the weights are written into 4-b precision with 50% activity for both rows and columns. For one read operation, we assume a single row access of 1 B in memory mode, or multiple eight-column-by-eight-column accesses in computing mode, as in the design in which there are eight read circuits with 4-b precision. Such time multiplexing is needed when the size of the read circuit and voltage S/A are relatively larger than the pitch of the column; thus, multiple columns have to share one read circuit and voltage S/A to improve the area efficiency. This explains the larger write/read latency and energy consumption in the computing mode compared to the memory mode.

We also evaluated the SRAM array architectures as a comparison. With the exception of lower write latency and energy, the SRAM's read latency, energy, area, and leakage power are much worse than RRAM, particularly for the computing mode. First, an SRAM cell has six transistors with standby leakage, while an RRAM cell has one transistor and is nonvolatile. Second, in the computing mode, the SRAM uses four cells to represent the 4-b data, while the RRAM stores 4 b in one cell. The weighted sum operation in SRAM has to be performed sequentially, as only one row can be accessed at a time, whereas RRAM can support parallel access to all rows simultaneously. As the weighted sum operation dominates in many computing applications, the proposed RRAM-based MFM demonstrates considerable potential to speed up the algorithms while reducing the energy/power consumptions.

C. System Modeling

An RTL model of the system-level peripheral logic elements was created to model the critical path delay, power consumption, leakage energy, and area of a single PE. The schedule table was modeled using CACTI [7]. Other components were coded in synthesizable SystemVerilog and mapped to a 32-nm technology with fast NMOS, fast PMOS (fast-fast corner), and 25 °C library from Synopsys using Design Compiler.

Each functional unit of the PE was synthesized separately with delay constraints placed along the overall data flow path throughout the PE. While this approach yields an inferior mapping result compared to a completely flattened netlist, it allows the energy use of each functional unit to be characterized individually. The critical path delay (and the maximum clock frequency) was then obtained by summing the individual component delays along the critical path. The circuit-level parameters of the MFM array were obtained from the macrosimulator in Section III-B. The energy for other supporting logic components was obtained from synthesis, assuming a 12.5% switching activity. Energy values for the schedule table were obtained from CACTI. The estimated area and leakage for a PE were also taken from the synthesis and CACTI results. In addition to the component-by-component synthesis of the RTL model, a complete RTL model for one PE was created and simulated using Synopsys VCS to ensure the functional correctness of the model.

D. Application Mapping

Our benchmark application suite consists of seven common kernels: the iterative, shrinking thresholding algorithm, stochastic gradient descent (SGD), K-means clustering (the centroid calculation), fast Fourier transform (the trigonometric functions), Gaussian blur, Sobel gradient, and radial basis function (RBF) in the artificial neural network. This provides a combination of neuro-inspired digital signal processing and graphics applications. Each of these benchmarks was implemented in the MFM system using the software described in Section II-A and modified to use the neurocomputing capabilities whenever possible. Applications were also mapped to FPGA (Altera Stratix V FPGA, 28-nm CMOS process, 0.85-V supply), in software (Intel Q8200 CPU) and in the SRAM-based MAHA framework [3]. The MFM, MAHA, FPGA, and GPP platforms differ greatly, and to facilitate a fair comparison, all devices are scaled to the same process.

Each application was mapped to the MFM system using the approach outlined in Section III-C. Latency values were computed assuming an 81-ns clock period and multiplying by the total number of cycles required. To obtain the dynamic energy values, the number of each operation type for a given application was counted and then multiplied by its energy consumption. Static power was computed by taking the total leakage power of the PEs.

The MAHA implementations were created following the MAHA mapping methods [3]. The FPGA implementations were written in Verilog and compiled to the Stratix V using Quartus II. Latency estimates were obtained using TimeQuest, and energy estimates were obtained by multiplying the number of effective cycles, cycle time, and power estimates reported in PowerPlay. To estimate the area, we multiply the adaptive logic module (ALM), memory block, and DSP block area by the number of utilized ALMs, memory blocks, and DSP blocks reported by Quartus II. The software implementations were written in C as single threaded applications, then compiled, and run on the target CPU. The programs run $3E + 5$ input vectors 200 times each and return the average energy and delay values per application to reduce measurement noise. The power consumption is assumed to be half the rated thermal design power (95 W) divided by 4 since only 1 of the 4 cores was active. We estimate that a single core occupies approximately one quarter of the die area, roughly 25 mm^2 at 45 nm, and scale this to 32 nm for the comparison.

E. Results and Discussion

Fig. 3(a)–(f) shows the comparison results for four implementations on different platforms. Fig. 3(a) shows the latency results. MFM has an average five orders of magnitude improvement over GPP, a 32X improvement over FPGA, and a 100X improvement over MAHA. These improvements are mainly due to the massive parallelism available in the neuroinspired computing model.

The computation in GPP is single threaded, so operations execute sequentially, resulting in poor performance compared to the MFM array. FPGA and MAHA are limited by I/O constraints and computing engines, but the high bandwidth in MFM makes the parallel computation possible. We observed that SGD and RBF latency improvements are lower than the

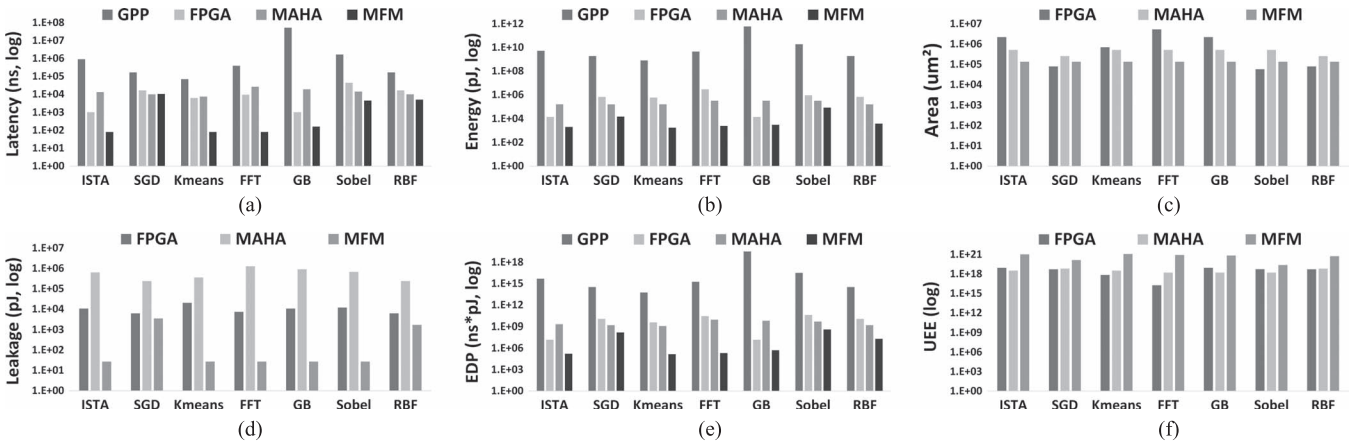

Fig. 3. Comparison of (a) latency, (b) energy, (c) area, (d) leakage, (e) EDP, and (f) UEE for seven applications mapped to the proposed MFM-based PE array as well as three alternative platforms, namely MAHA, FPGA, and GPP. The lower value is better, except for UEE.

other applications, likely because they are write intensive, whereas the other applications are read dominant. As shown in Table I, write latency in computing mode is more than 10X longer than read latency. As a result, read dominant applications will see more significant improvements.

Fig. 3(b) shows the energy comparison. MFM has an average seven orders of magnitude improvement versus GPP, a 247X improvement over FPGA, and a 68X improvement over MAHA. MFM employs the novel modified RRAM array and builds on a spatiotemporal computing architecture. As a result, it takes advantage of not only the RRAM benefits over SRAM but also the spatiotemporal computing, as highlighted by the MAHA architecture, to achieve even better energy.

The area comparison is shown in Fig. 3(c). MFM has an average 11X improvement over FPGA and a 3X improvement over MAHA. This is primarily due to the higher integration density of RRAM. Each cell can have 16 states, which is equivalent to storing 4 b in one cell. Additionally, MFM increases the hardware resource reusage by integrating storage and computing in one place rather than needing separate storage and computation units.

Fig. 3(d) shows the leakage comparison. MFM achieves an average 300X improvement over FPGA and four orders of magnitude over MAHA because of the low leakage nature of the proposed RRAM array, where unselected cells are effectively turned off. In comparison, the CMOS-based memory circuits of FPGA and MAHA have much higher static leakage.

We also compare energy efficiency using two metrics: energy–delay product (EDP) and unified energy efficiency (UEE). EDP results are calculated by multiplying energy and latency; UEE is calculated by dividing the total data being processed by the energy and again by the area. Results for EDP and UEE are shown in Fig. 3(e) and (f), respectively. MFM achieves an average 23 000X EDP improvement over FPGA and an 11 000X improvement over MAHA. Moreover, it has an average UEE improvement of 6000X over FPGA and 200X over MAHA.

IV. Conclusion

We have presented a novel MFM fabric, which can be dynamically configured into either traditional storage or a neuromorphic computing fabric. Using RRAM as a case study, we have presented device-level engineering and circuit/architecture level design choices to enable such a malleable framework. The modified pseudocrossbar array facilitates both regular memory read and write operations and computation through judicious changes in the peripheral circuits. Such MFM units can be effectively integrated into any PE, and a sea of such elements can be implemented inside a memory block, including primary and secondary memories of a processor, to mitigate the Von Neumann bottleneck for data-intensive applications. We have shown that such a fabric leads to major improvements in performance and energy efficiency over alternative platforms. It incurs minimal overhead in memory density/performance and is scalable to diverse emerging memory technologies.

References

[1] P. Ranganathan, "From microprocessors to nanostores: Rethinking data-centric systems," *Computer*, vol. 44, no. 1, pp. 39–48, Jan. 2011.

[2] H.-S. P. Wong *et al.*, "Metal-oxide RRAM," *Proc. IEEE*, vol. 100, no. 6, pp. 1951–1970, Jun. 2012.

[3] S. Paul, A. Krishna, W. Qian, R. Karam, and S. Bhunia, "MAHA: An energy-efficient malleable hardware accelerator for data-intensive applications," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 6, pp. 1005–1016, Jun. 2015.

[4] D. Niu, C. Xu, N. Muralimanohar, N. P. Jouppi, and Y. Xie, "Design trade-offs for high density cross-point resistive memory," in *Proc. ISLPED*, 2012, pp. 209–214.

[5] D. Kadetotad *et al.*, "Parallel architecture with resistive crosspoint array for dictionary learning acceleration," *IEEE J. Emerging Sel. Topics Circuits Syst.*, vol. 5, no. 2, pp. 194–204, Jun. 2015.

[6] L. Gao, P.-Y. Chen, and S. Yu, "Programming protocol optimization for analog weight tuning in resistive memories," *IEEE Electron Device Lett.*, vol. 36, no. 11, pp. 1157–1159, Nov. 2015.

[7] S. J. Wilton and N. P. Jouppi, "CACTI: An enhanced cache access and cycle time model," *IEEE J. Solid-State Circuits*, vol. 31, no. 5, pp. 677–688, May 1996.

[8] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "NVSim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 31, no. 7, pp. 994–1007, Jul. 2012.