Exploring Processing In-Memory for Different Technologies

Saransh Gupta, Mohsen Imani, and Tajana Rosing
CSE Department, UC San Diego, La Jolla, CA 92093, USA
{sgupta, moimani, tajana}@ucsd.edu

ABSTRACT

The recent emergence of IoT has led to a substantial increase in the amount of data processed. Today, a large number of applications are data intensive, involving massive data transfers between processing core and memory. These transfers act as a bottleneck mainly due to the limited data bandwidth between memory and the processing core. Processing in memory (PIM) avoids this latency problem by doing computations at the source of data.

In this paper, we propose designs which enable PIM in the three major memory technologies, i.e. SRAM, DRAM, and the newly emerging non-volatile memories (NVMs). We exploit the analog properties of different memories to implement simple logic functions, namely OR, AND, and majority inside memory. We then extend them further to implement in-memory addition and multiplication. We compare the three memory technologies with GPU by running general applications on them. Our evaluations show that SRAM, NVM, and DRAM are 29.8× (36.3×), 17.6× (20.3×) and 1.7× (2.7×) better in performance (energy consumption) as compared to AMD GPU.

CCS CONCEPTS

• **Hardware → Emerging technologies**; *Integrated circuits*;

KEYWORDS

Processing in Memory; Non-volatile memories; SRAM; DRAM; Memristors; Energy efficiency; Analog computing

ACM Reference Format:
Saransh Gupta, Mohsen Imani, and Tajana Rosing. 2019. Exploring Processing In-Memory for Different Technologies. In *Great Lakes Symposium on VLSI 2019 (GLSVLSI '19), May 9–11, 2019, Tysons Corner, VA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3299874.3317977

1 INTRODUCTION

In this era of the internet of things (IoT), billions of devices are connected together to form a network. These devices generate a huge amount of data leading to the concept of big data. Many applications run several machine learning, neural network, and other emerging algorithms on this raw data to draw meaningful results [1–3]. In the current computing systems, all processing tasks are done on CPU/GPU cores while the data to be processed is stored in memory [4]. The process of bringing the entire data to the processing core becomes a bottleneck owing to the limited data bandwidth. The cache hierarchy in these systems tries to accelerate this process but it fails when the size of the data becomes too large to fit into various cache levels.

GLSVLSI '19, May 9–11, 2019, Tysons Corner, VA, USA

ACM ISBN 978-1-4503-6252-8/19/05…$15.00
https://doi.org/10.1145/3299874.3317977

Processing in memory present possible solutions to this data transfer bottleneck [5–8]. It processes data inside memory, *i.e.* the source of the data, without using any power hungry cores [9–11]. Although many PIM techniques have been proposed recently, the majority of them are application specific. For example, [5, 12–14] accelerate neural networks in memory. Similarly, work in [15] and [16, 17] designed a PIM architecture for acceleration of brain-inspired computing and graph processing applications. There are some works like [18] which support limited basic functions in memory but fail to implement complex functions like addition and multiplication. However, some work implemented basic logic functions in memory and extended them to implement advanced functions like addition and multiplication [12, 19–22]. These techniques can be extended to implement several functionalities in memory but they suffer from huge latency bottleneck mainly due to the slow switching speeds of NVMs.

The PIM techniques discussed above, as well as most of the other proposed techniques, are targeted towards NVMs and use the programmable resistance provided by them to implement logic. Since the current computing systems majorly use SRAMs and DRAMs, these techniques do not provide readily implementable solutions for the data transfer bottleneck that these systems face. The work in [23] tried to implement PIM in DRAM by decoupling logic and memory circuits in different dies using 3D-stacking. However, the designing and manufacturing challenges involved with such approaches have limited their use.

In this paper, we propose GPIM (general processing in-memory) to enable PIM in three major memory technologies, namely SRAM, DRAM, and the emerging NVM technologies. The major contributions of this paper are as follows:

- We propose novel circuits for SRAM, DRAM, and NVM to enable PIM. These circuits are configurable and the same circuit can implement multiple fundamental operations, including `OR`, `MAJ`, and `AND`. We also implement in-memory addition and multiplication in GPIM-enabled SRAM, DRAM, and NVM.
- For NVM and SRAM, GPIM exploits the differential effective resistance of logic states to generate data dependent current. In the case of DRAM, GPIM uses data-controlled capacitive charging to execute logic. Unlike the previous PIM approaches, neither does our implementation destroy the stored data, nor does it require creating temporary copies for processing.
- GPIM significantly reduces the number of memory device switches involved in PIM operations, thereby increasing the lifetime of the memory. Moreover, it also minimizes the number of memory cells required for performing PIM operations. It increases the effective memory utilization.

2 RELATED WORK

Processing in memory has recently gained interest owing to the emergence of new memory technologies with the capability to both store data as well as process it. Several PIM techniques have been proposed to accelerate emerging applications. For example, [13] accelerates neural networks by storing weights in memory and applying inputs directly to them in the form of current. The work in [5]

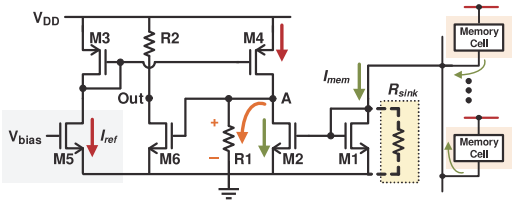

Figure 1: Current-based implementation of logic functions.

uses spiking-based neuromorphic computing to multiply inputs, applied at bitlines, with the weights, stored in the memory, to implement neural networks in memory. Such techniques do accelerate the process but depend upon utilizing multi-leveled RRAM devices. These multi-leveled devices suffer from severely low endurance issues. Moreover, some of these also use power-hungry ADCs and DACs, which contribute significantly to the power requirements of the system.

There are many PIM techniques which deal with general implementations [20, 22, 24]. The work in [18] implements the basic operations like bitwise OR, AND. However, it does not support functions like addition and multiplication, limiting the use of such technique for most of the applications. The work in [20, 25] introduces schemes to implement addition in crossbar memory. This is further extended in [22] to improve the efficiency of addition and implement multiplication in memory. These techniques are general but suffer from the high latency involved in a large number of device switches. Moreover, [22] uses configurable interconnects which add area overhead to the memory block.

Some researches also present PIM like techniques for SRAMs and DRAMs. For example, [26] presents approaches to improve the performance and energy efficiency of compute caches. Such caches are able to execute only simple bitwise functions on the data stored in SRAM cells using sense amplifiers. The work in [27] accelerates machine learning by doing computations in SRAM cell. However, it uses large DACs to implement it, introducing significant power and area overheads. The work in [23] exploits the concept of 3D stacking to separate the logic and memory circuits into different dies, overcoming the cost challenges involved in integrating memory and logic. These work do propose PIM techniques for SRAM and DRAM but they have limited functionality and are not directly applicable to more general systems running various kinds of applications.

On the other hand, we propose GPIM which implements basic logic functions by modifying the sense amplifiers for SRAM, DRAM, and emerging memory technologies. We further extend it to implement addition and multiplication in memory.

3 GENERAL PROCESSING-IN MEMORY

The dissimilarity in the behavior of different memory technologies calls for different types of circuits to enable PIM in memory. We propose two types of circuits, a current-based circuit for NVMs and SRAM and voltage-based circuit for DRAM.

3.1 Current-based Implementation

In this section, we design an analog circuit which exploits the difference in current flowing through the NVM and SRAM memory cells in different states. This reduces the number of transistors needed to implement logic. Also, it is faster than the slow purely in-memory implementations which rely on multiple data-driven device switches to implement various functions [20, 25]. Moreover, the proposed circuits do not affect the read/write characteristics of the memories.

3.1.1 Analog Properties of SRAM and NVM. **NVM:** Emerging NVM technologies, like Resistive Random Access Memory (RRAM), Phase Change Memory (PCM), save data in form of their resistance/matter state. In all the cases, a data readout occurs by applying a voltage across the memory cells and measuring the current through it. The sense amplifiers at the periphery of the memory blocks sense this current to output the stored data. Our design modifies this sense amplifier to enable the execution of logic.

The total current through the memory cells, say `d1`, `d2`, and `d3`, depends upon the resistance of the devices, hence the stored data. In our setup, we consider a low (high) resistive state as logic '1' ('0'). For example, in the case of memristors (ReRAM), we represent logic '1' and '0' with resistive states of $10k\Omega$ (R_{on}) and $10M\Omega$ (R_{off}) respectively. So, the current is maximum when all the three bits are logic '1' and minimum when they are all logic '0'.

SRAM: Unlike NVM, SRAM is volatile in nature. The data in SRAM is stored in the form of logic state maintained by two cross-coupled inverters as shown in Figure 2a. In a memory cell, one of the inverters stores logic '1' at the output node while the other stores logic '0.' The inverter with logic '0' at the output node has the NMOS transistor in on state. A transistor in on state has a much lower resistance than that in off state. We exploit this property to implement basic functions in SRAM.

3.1.2 Circuit Design. The memory cells in a column of the memory block share the same sense amplifier. So, the data bits to be processed are required to be present in the same column of the block. The memory cells are selected by activating the corresponding wordlines (which select the memory row) and bitlines (which select the required column sense amplifier). The voltage applied at wordlines makes the current flow through the memory cells. The total current through the selected memory cells is passed through the sense amplifier.

Proposed Circuit: Figure 1 shows the proposed circuit. It compares the current through memory cells with a reference current. For now, we ignore the resistor R_{sink} in the circuit. The transistor pairs M1-M2 and M3-M4 form current mirrors. M5 acts as reference current generator while M6 and R2 form the output stage. The total current from the memory cells is passed through M1. This current is copied to the M2 by the current mirror. The bias voltage, V_{bias} applied to the gate of M5 directly controls the current through it, making it a voltage controlled current source. We use this current source to generate the reference current. This same current flows through M3 as well which is copied by M4. In other words, the current through M4 is directly proportional to the bias voltage.

Since, M2 and M4 try to drive different currents, the resistor connected to the drains of both the transistors acts as the path to drain the excess current. This also develops a voltage difference across the resistor, R1. This voltage is dependent upon the amount of current flowing through R1. As the current through R1 increases, the voltage at node 'A' increases as well. Node 'A' controls the output transistor, M6. As the voltage at node 'A' increases beyond the threshold of M6, it turns on and pulls the output node to ground. While the output node remains at a higher voltage when M6 is off.

Implementing `MAJ`, `OR`, and `AND`: The current through a memory cell changes with data. This allows us to use the total current through multiple cells to directly implement logic functions. To implement a function like `MAJ`, the reference current is set close to the current that is generated when two of the memory cells store data '1'. Hence, in the case when two out of three data bits are '1,' the current through M2 is close to the current through M4. Hence, very little current flows through R1 and voltage generated at node A is less than the threshold voltage of M6. M6 remains off,

Table 1: Circuit parameters for RRAM and SRAM

	RRAM			RRAM (R_{on} <1$k\Omega$)			SRAM		
	OR	MAJ	AND	OR	MAJ	AND	OR	MAJ	AND
V_{bias} (V)	0.5	0.7	0.9	0.5	0.7	0.9	0.3	0.45	0.6
I_{ref} (uA)	10	30	50	10	30	50	30	15	5
Latency (ps)	50	50	50	50	50	50	150	150	150
Energy (fJ)	5.3	8.1	9.1	374.9	377.0	378.4	10.2	7.9	5.9

maintaining the output node high. When one or none of the bits are '1' the current through M2 reduces and the voltage at node A rises. This switches M6 on which pulls the output node low. In the case when all the three bits are high, the current through M2 is limited by the current that can be supplied by M4. However the difference between the two currents is very low and the M6 remains off, keeping output node high. The same technique is extended to implement OR and AND. The only required change is the bias voltage controlling M5. For implementing OR (AND), the bias voltage is reduced till the reference current becomes equal to the current through the memory cells when one (three) of the data bits is '1'.

Table 1 shows the required bias voltages, generated currents, latency, and energy consumption for different functions. Ideally, the current varies linearly with the change in data stored in the memory cells. However, the current sensing transistor of sense amplifier has an on resistance of its own which is comparable to R_{on} of the memory, which introduces non-linearity in the behavior of current. The current comparison technique still works since the change in current is significant and detectable by the proposed circuit.

3.1.3 Technology Dependent Variations. **NVM:** NVM characteristics depend upon the material used for making the device. Hence, the effective resistance of device as well as the performance vary with the process. While the proposed method works well for devices with R_{on} greater $1k\Omega$, other NVM devices require slight modifications in the circuit.

In such situation, the large ON resistance of the current mirroring NMOS transistor results in a huge resistance drop across the transistor. Since there is a very little drop across memory cells, the change in voltage across the cells and hence the current through them is not detectable. In order to overcome this issue, we introduce a resistor, R_{sink}, in parallel to the current mirroring transistor. The resistance of this resistor is equal to R_{on} of the memory cell. It acts as a current sink for the current through the memory cells. The voltage at the common node is now dependent upon the data stored in the memory. Majority of the current now flows through the sink resistor. However, the current into the current mirror is now determined by the voltage at the common node. Since the voltage is dependent upon data, the current becomes data dependent as well. The remaining circuit works as before. However, using a sink resistor introduces a low resistive path for the current to flow, resulting in a direct increase in the total power consumption of the circuit.

SRAM: In the case of SRAMs, we do not have access to the terminal which drains the current out of the memory cell like in case of RRAM. Hence, we connect a gated voltage source to the bitline. Whenever the logic is to be executed, the voltage source is activated and the current through it is copied to the logic circuit using PMOS current mirror. The original read/write mechanism remains the same, wherein the voltage source is deactivated and the conventional conditioning circuit is used. At the time of logic execution, the conditioning circuits are disabled with only the auxiliary circuit being activated. This circuit is similar to the one proposed previously with roles of NMOS and PMOS interchanged. The voltage source feeds current to the memory cells directly through bitlines.

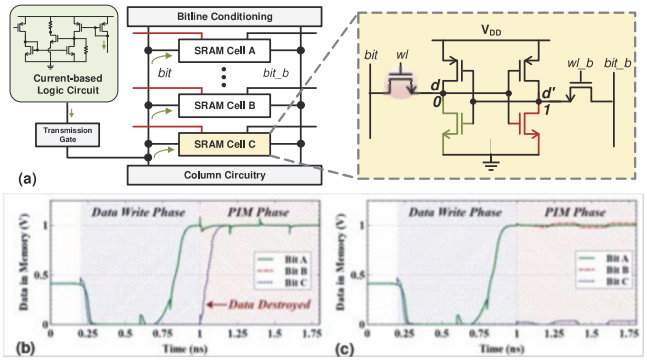

Figure 2: (a) Overview of GPIM in SRAM; (b) effect of GPIM circuitry on stored data; (c) SRAM data while using the data preserving techniques.

The amount of current discharging through a memory cell depends upon the data it stores. The node of the cross-coupled inverter storing '1' presents a higher resistance to the ground due to the NMOS transistor being switched off. On the contrary, the node with '0' has a lower resistance to ground. The data to be processed should be present in the memory cells connected to the same bitline. To execute logic in SRAM, we activate the wordlines corresponding to the concerned memory cells.

Preserving data in SRAM — The major difference between SRAM and NVM is the way that data is stored. In the case of SRAM, it is possible that logic execution destroys the data stored in a cell as shown in Figure 2b. For example, the node d in the SRAM cell C (storing '0') in Figure 2a can be charged due to the constant voltage applied during the GPIM execution phase, destroying the initial data. Our design adopts techniques to preserve data in SRAM while enabling easy logic execution [28]. We reduce the voltage for the access transistor encircled in Figure 2a in the execution phase to prevent the circuit from having a strong effect on the data stored in SRAM. This decreases the effective amount of current that flows through the memory cell and hence diminishes the possibility of data bits getting flipped. However, long and repeated access to the same memory cell can still lead to a situation where the data is destroyed. In order to prevent this, we split the wordline access. Here, only the access transistor directly connected to the GPIM circuit is activated while the other remains off during the execution stage. This ensures the presence of a strong logic state in the other half of the cell, hence preserving the data. The effectiveness of the techniques can be observed in Figure 2c, where the stored data remains intact throughout the execution phase.

3.2 Voltage-based Implementation

In this section, we introduce a circuit which uses the DRAM bitline voltage to selectively charge the output capacitor. The total charge developed at the capacitor is used to implement different functions in-memory.

3.2.1 Analog Properties of DRAM. DRAM cells are generally used in the folded bitline subarray arrangement as shown in Figure 3a. Figure 3b shows a DRAM cell. DRAM stores data in the form of charge on a capacitor, accessible via a transistor. A read operation on a DRAM cell is destructive in nature. Hence, a read is followed by a write operation to preserve data in memory. Moreover, the charge on the storage capacitor leaks out over time and requires a periodic refresh of data. Hence, unlike the previously discussed memories, the effective resistance of a DRAM cell is independent

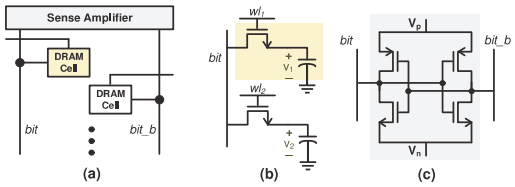

Figure 3: (a) Folded DRAM subarray configuration; (b) DRAM cells on the same bitline; (c) basic DRAM sense amplifier.

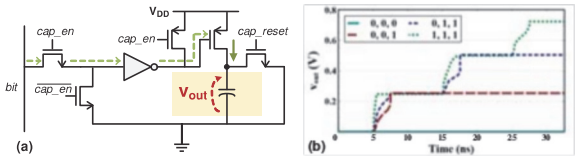

Figure 4: (a) GPIM circuit for DRAM; (b) data-dependent voltage at the output capacitor.

of the data stored. Hence, the usual current sensing circuit won't work for DRAM.

3.2.2 Circuit Design. **Differential Sense Amplifier:** Figure 3c shows the basic sense amplifier used in DRAM. It is made up of two cross-coupled inverters. The voltage across the inverters is controlled by signals V_p and V_n. Of the two bitlines, only one bitline reads in a cycle while the other acts as the reference. While reading, the access transistor of a DRAM cell is activated. The cell either pulls down or pushes up one of the bitlines by a small amount, depending upon the stored data. This difference generated between the two bitlines is amplified by pulling down V_n to 0 and then pulling up V_p to V_{DD}. This amplification results in rewriting the cell with the value that was just read. Hence, such a read operation is non-destructive in nature in contrast to the traditional DRAM read operation.

Proposed Circuit: We propose controlled capacitor charging based circuit to implement logic in DRAM. Figure 4a shows the proposed circuit. The circuit consists of a capacitor whose charging is regulated by the voltage at bitline. A signal, *cap_en*, activates the logic circuit once the bitline has read the data. Whenever the logic circuit is active and the voltage at bitline is above the threshold of the high V_T inverter, the capacitor is charged by the power supply. The three memory cells are read one at a time and the capacitor is activated after every read operation. The capacitor is charged whenever the stored bit is '1' while its charge does not change if the stored bit is '0.' In addition to the control switches and inverter, the circuit uses some transistors to pull up or down certain nodes to ensure correct functioning. The *cap_reset* signal discharges the capacitor completely at the beginning of every logic execution phase.

Implementing `MAJ`, `OR`, and `AND`: The total voltage developed at the capacitor at the end determines the number of bits which are '1,' as shown in Figure 4b. In our experimental setup for 3-bit logic operations, the voltage levels 0.25V, 0.50V, and 0.72V correspond to the presence of one, two, and three logic high ('1') bits. So, a final voltage greater than or equal to 0.25V corresponds to `OR` being true. Similarly, a voltage greater than or equal to 0.50V and 0.72V translates to `MAJ` and `AND` being high respectively.

3.3 GPIM Architecture

Figure 5 shows the overview of the proposed GPIM. It consists of an array of memory cells. Each memory cell is connected to a wordline

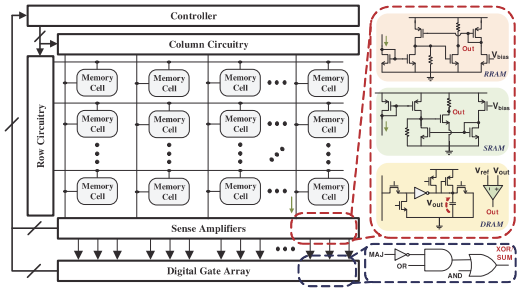

Figure 5: GPIM Overall Architecture

and a bitline, except a SRAM cell which is connected to 2 wordlines and 2 bitlines. The controller handles the column and row circuitry which control the bitline and wordline voltages, respectively. The sense amplifier is responsible for reading the data and processing it. Depending upon the memory technology, the sense amplifier will be one of the circuits proposed in the previous sections. Figure 5 shows the structure of these possible sense amplifiers. Each bitline has a GPIM circuit which outputs `OR`, `MAJ`, or `AND` depending upon the bias voltage. The output of the sense amplifier is supplied to a gate array, as shown in Figure 5, which is required to perform other operations like addition. This gate array just relies on the output of the GPIM sense amplifiers and not directly on the actual data. In some cases, intermediate results are written back to the memory for each bit and used to calculate the results for next bit. N-bit addition, implemented similar to [22], is a simple example of such a case where C_{out} of each 1-bit addition is written back to memory.

Advantages of GPIM: GPIM provides the following benefits:
Latency — The latency of implementing bitwise functions (`OR`, `MAJ` or `AND`) using the proposed amplifiers involves a read and logic execution. Both data read and logic execution being fast, are done together in a single cycle for SRAM and RRAM. For DRAM, the logic execution is slower than the DRAM write operation and has a latency approximately equal to 3 write cycles. Effectively, the circuit takes 2 or 4 cycles to execute 1-bit addition which is quite less than the time taken by previous PIM techniques to implement the same functionality.
Energy Consumption — GPIM benefits in energy consumption due to two major factors, (i) less dependency on device write operations and (ii) reduced number of memory operations required for implementing logic.
Device Switches in NVMs — PIM in NVMs is challenged by low endurance issues. Implementing logic using sense amplifiers reduces the number of read/write operations in memory, helping with the issue. For example, 32-bit addition in [22] involves around 400 device switches while GPIM uses only 31 device switches.
Memory Utilization — The proposed technique does not require a separate processing area in the memory block as in the case of previously proposed PIM designs. Previous designs require the use of a memory size much larger than the storage capacity to accommodate for processing memory cells. This is not required by our design since the processing happens in the sense amplifiers, increasing the effective utilization of the memory.

4 EXPERIMENTAL RESULTS

4.1 Experimental Setup

Performance and energy consumption of proposed hardware are obtained from circuit level simulations for a 45nm CMOS process technology using Cadence Virtuoso. We use VTEAM memristor

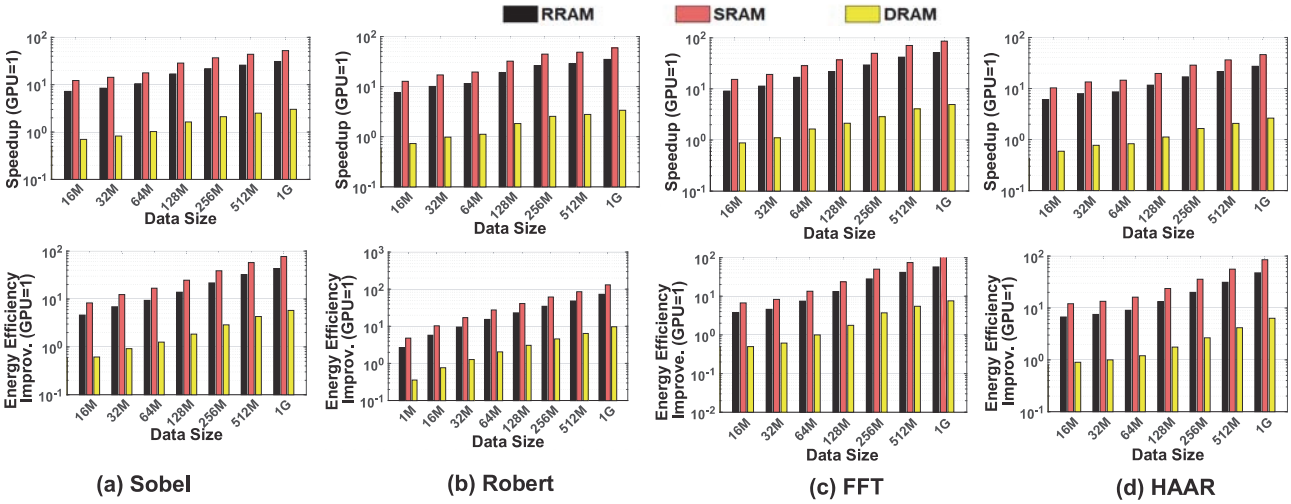

Figure 6: Speedup and energy efficiency improvement of preforming PIM operation over different applications.

Table 2: Comparison of 3-bit bitwise operations and 32-bit addition and multiplication in RRAM with prior work

Design	Metric	OR	MAJ	AND	XOR	32b ADD	32b MUL
GPIM	Latency (ns)	0.05	0.05	0.05	0.18	33.7	80.1
	Energy (fJ)	5.31	8.07	9.11	24.5	1.16×10^3	29.6×10^3
	Device Switches	0	0	0	0	31	$\sim 1.2 \times 10^3$
MAGIC [20, 22]	Latency (ns)	3.3	5.5	5.5	9.9	457.6	1090.3
	Energy (fJ)	48.1	96.4	96.2	192.6	9.2×10^3	181×10^3
	Device Switches	3	6	6	12	~400	$\sim 7.5 \times 10^3$

model [29] for our memory design simulation with R_{ON} and R_{OFF} of $10k\Omega$ and $10M\Omega$ respectively. We design the conventional 6T SRAM cell with standard transistor sizing using 45nm technology library with normal V_{TH} cells. In case of DRAM, the cell is designed in the same technology using the 1T-1C configuration with a storage capacitor of 30fF.

We compare the efficiency of the proposed GPIM with AMD Radeon R9 390 GPU, a recent GPU architecture, with 8GB memory. In order to avoid the disk communication in the comparison, all the data used in the experiments is preloaded into 64GB, 2.1GHz DDR4 DIMMs. We used Hioki 3334 power meter to measure the power consumption of GPU. We tested both GPIM and GPU over four OpenCL applications: *Sobel, Robert, Fast Fourier transform (FFT)* and *DwHaar1D*. For image processing applications, we used random images from *Caltech 101* [30] library, while for non-image processing applications inputs are generated randomly. Majority of these applications consists of additions and multiplications. The other common operations such as square root have been approximated by these two functions in OpenCL code.

4.2 Energy and Performance Comparison

We compare GPIM with the state-of-the-art prior work [20, 22]. The work in [20] is based on MAGIC logic family. It uses NOR computations to implement different functions in memory. The work in [22] used it to implement data-intensive additions and multiplications in memory. Table 2 compares the implementation of a 32-bit adder using GPIM for RRAM devices with those in [20, 22]. It shows that GPIM outperforms all the previous implementations. It performs at least 2.4× times better than the fastest adder. A major part of this improvement comes from the fact that GPIM avoids the huge latency associated with RRAM device switches by implementing logic using sense amplifiers. We evaluate GPIM for 32-bit multiplication. We compare it with [22] since it is the only switching based PIM supporting multiplication in memory. Our results show that in the case of RRAM, GPIM is 15× faster and consumes 6× less energy. This improvement can be credited to the lower latency and energy consumption of the MOSFETs as compared to RRAM devices.

4.3 GPIM Exploration over Technologies

We compare the three memory technologies, SRAM, DRAM, and RRAM, with traditional GPU core on the basis of performance and energy consumption. The comparison was done by running four different applications on all technologies. The experiments were conducted assuming that the memories were big enough to store the required data. Figure 6 shows that SRAM performs better than all other technologies, being on an average 29.8× faster than GPU while having 36.3× lower energy consumption. Here, while comparing performance, we take into consideration the absolute latency of execution. RRAM has latency and power consumption close to those of SRAM, on an average 17.6× and 20.3× respectively as compared to GPU. It also follows trends similar to SRAM over different applications. However, DRAM performs worse than even GPU in some cases. It also has the least performance and energy efficiency among the three memory technologies, being on an average 1.7× faster and 2.7× more energy efficient as compared to GPU.

SRAM: The results indicate that SRAM is the best solution for in-memory processing. However, this may not be a feasible option. The major drawback of SRAM is its cell area which goes as high as $140F^2$ in commercial SRAMs. Hence, having really huge SRAM blocks becomes too expensive. However, SRAMs can be utilized as PIM when the data size is small which still gives a significant improvement as compared to traditional GPU cores. The fast in-memory processing opportunities provided by SRAMs can also be exploited by compute caches [26] to reduce the data transfers across cache hierarchy.

RRAM: RRAM has performance and energy consumption metrics close to SRAM. Unlike SRAM, RRAMs are dense with a cell size of $4F^2$ when used in a crossbar configuration. Processing in RRAM devices is bottlenecked by the slower device switching speed as compared to SRAM. Moreover, RRAM is challenged by endurance issues. However, GPIM reduces the number of device switches required while processing data, which partially reduces these issues.

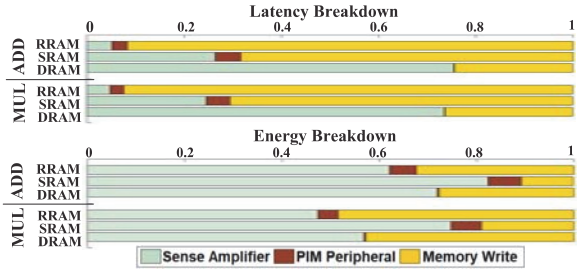

Figure 7: Latency and energy breakdown of different technologies while performing 32-bits addition and multiplication.

DRAM: DRAM performs poorly, even worse than GPU for small data sizes (< 64MB). This is due to the high latency and energy requirements of DRAM read and write operations which directly impact the efficiency of PIM in DRAM. However, as the data size increases, beyond 64MB in our experimental setup, the data transfer costs associated with the traditional GPU cores start dominating. Since PIM reduces these data transfers, GPIM for DRAM starts performing better than GPU with lower energy consumption. Hence, PIM is suitable for DRAM when the data size gets huge.

4.4 Performance and Energy Breakdown

Figure 7 shows the distribution of total latency and energy consumption among different stages of GPIM. The breakdown of latency for RRAM reveals that in the case of both addition and multiplication, the majority of the time (>90%) is spent in writing back the intermediate results to the memory. This is due to the high device switching latency of RRAM, which GPIM avoids as much as possible in logic execution. The latency for SRAM is distributed between the auxiliary circuit and memory write back in the ratio of about 2:5. This can be attributed to the reduced write latency of SRAM as compared to RRAM, while the logic execution latency for SRAM is more than that of RRAM. In the case of a DRAM, the distribution is quite different with logic execution taking the majority of the time. This happens because GPIM performs sequential reads of the data to be processed, which happens in parallel in RRAM and SRAM.

The breakdown of energy for addition reveals that the energy consumed by the logic circuit is greater than the device write energy for all the three memories. A major factor behind this behavior is the use of multiple copies of the logic circuit for different functions. In case of multiplication, the number of device writes is high, resulting in a greater contribution by the device write operations to the total energy consumption. The energy consumption for SRAM are highly skewed towards logic circuits because the device writes in SRAM are very efficient.

4.5 GPIM Area Overhead

The area overhead of GPIM depends upon the size of a memory bank. Our evaluations for a 4*M*b memory bank indicate that GPIM introduces overheads of 0.83%, 3.5%, and 4.1% for SRAM, DRAM, and RRAM respectively. This result follows the densities of different memories, with overhead being lesser for a memory with larger cell area. Moreover, for a fixed memory bank size, the overhead reduces with an increase in the number of rows in the bank.

5 CONCLUSION

In this paper, we proposed GPIM which enabled PIM in SRAM, NVMs, and DRAM. The circuits exploit the analog properties of the respective memories to implement a fast and energy efficient solution for PIM. The application of PIM in conventional memories allows us to avoid the unnecessary data transfers between memory and the processing core. Moreover, our implementations use fast switching speeds of transistors to achieve high performance in the conventional memories as well as NVMs. GPIM executes basic logic functions in memory and then uses them to implement arithmetic functions like addition and multiplication. Our evaluations show that SRAM, NVM, and DRAM are 29.8× (36.3×), 17.6× (20.3×) and 1.7× (2.7×) better in performance (energy consumption) as compared to AMD GPU.

ACKNOWLEDGEMENTS

This work was partially supported by CRISP, one of six centers in JUMP, an SRC program sponsored by DARPA, and also NSF grants #1730158 and #1527034.

REFERENCES

[1] C. Perera *et al.*, "Context aware computing for the internet of things: A survey," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 414–454, 2014.
[2] J. Gubbi *et al.*, "Internet of things (iot): A vision, architectural elements, and future directions," *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.
[3] M. Imani *et al.*, "A framework for collaborative learning in secure high-dimensional space," in *Cloud Computing (CLOUD)*, IEEE, 2019.
[4] H. Zhang *et al.*, "In-memory big data management and processing: A survey," *IEEE TKDE*, vol. 27, no. 7, pp. 1920–1948, 2015.
[5] A. Shafiee *et al.*, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *IEEE ISCA*, pp. 14–26, IEEE Press, 2016.
[6] M. Soltiz *et al.*, "Memristor-based neural logic blocks for nonlinearly separable functions," *IEEE TC*, vol. 62, no. 8, pp. 1597–1606, 2013.
[7] M. Imani *et al.*, "Rapidnn: In-memory deep neural network acceleration framework," *arXiv preprint arXiv:1806.05794*, 2018.
[8] N. Talati *et al.*, "Concept: A column oriented memory controller for efficient memory and pim operations in rram," *IEEE Micro*, 2019.
[9] S. Shirinzadeh *et al.*, "Fast logic synthesis for rram-based in-memory computing using majority-inverter graphs," in *IEEE/ACM DATE*, EDA Consortium, 2016.
[10] M. Soeken *et al.*, "An mig-based compiler for programmable logic-in-memory architectures," in *IEEE/ACM DAC*, pp. 1–6, Ieee, 2016.
[11] G. Piccolboni *et al.*, "Investigation of the potentialities of vertical resistive ram (vrram) for neuromorphic applications," in *IEEE IEDM*, pp. 17–2, IEEE, 2015.
[12] M. Imani *et al.*, "Floatpim: In-memory acceleration of deep neural network training with high precision," in *ISCA*, ACM, 2019.
[13] C. Merkel *et al.*, "Neuromemristive systems: Boosting efficiency through brain-inspired computing," *Computer*, vol. 49, no. 10, pp. 56–64, 2016.
[14] S. Gupta *et al.*, "Nnpim: A processing in-memory architecture for neural network acceleration," *IEEE Transactions on Computers*, pp. 1–1, 2019.
[15] M. Imani *et al.*, "Exploring hyperdimensional associative memory," in *HPCA*, pp. 445–456, IEEE, 2017.
[16] M. Zhou *et al.*, "Gas: A heterogeneous memory architecture for graph processing," in *ISLPED*, p. 27, ACM, 2018.
[17] M. Zhou *et al.*, "Gram: graph processing in a reram-based computational memory," in *ASP-DAC*, pp. 591–596, ACM, 2019.
[18] M. Imani *et al.*, "Mpim: Multi-purpose in-memory processing using configurable resistive memory," in *ASP-DAC*, pp. 757–763, IEEE, 2017.
[19] A. Siemon *et al.*, "A complementary resistive switch-based crossbar array adder," *IEEE JETCAS*, vol. 5, no. 1, pp. 64–74, 2015.
[20] N. Talati *et al.*, "Logic design within memristive memories using memristor-aided logic (magic)," *IEEE TNANO*, vol. 15, no. 4, pp. 635–650, 2016.
[21] S. Gupta *et al.*, "Felix: Fast and energy-efficient logic in memory," in *ICCAD*, pp. 1–7, IEEE, 2018.
[22] M. Imani *et al.*, "Ultra-efficient processing in-memory for data intensive applications," in *ACM/IEEE DAC*, p. 6, ACM, 2017.
[23] R. Nair *et al.*, "Active memory cube: A processing-in-memory architecture for exascale systems," *IBM Journal of Research and Development*, vol. 59, no. 2/3, pp. 17–1, 2015.
[24] Y. Kim *et al.*, "Orchard: Visual object recognition accelerator based on approximate in-memory processing," in *ICCAD*, pp. 25–32, IEEE, 2017.
[25] S. Kvatinsky *et al.*, "MAGIC–Memristor-aided logic," *IEEE TCAS II*, vol. 61, no. 11, pp. 895–899, 2014.
[26] S. Aga *et al.*, "Compute caches," in *IEEE HPCA*, pp. 481–492, IEEE, 2017.
[27] J. Zhang *et al.*, "In-memory computation of a machine-learning classifier in a standard 6t sram array," *IEEE JSSC*, vol. 52, no. 4, pp. 915–924, 2017.
[28] S. Jeloka *et al.*, "A 28 nm configurable memory (tcam/bcam/sram) using push-rule 6t bit cell enabling logic-in-memory," *IEEE JSSC*, vol. 51, no. 4, 2016.
[29] S. Kvatinsky *et al.*, "Vteam: a general model for voltage-controlled memristors," *IEEE TCAS II*, vol. 62, no. 8, pp. 786–790, 2015.
[30] "Caltech Library." http://www.vision.caltech.edu/Image_Datasets/Caltech101/.