ABSTRACT

Title of Dissertation: RESISTIVE RAM BASED MAIN-MEMORY SYSTEMS: UNDERSTANDING THE OPPORTUNITIES, LIMITATIONS, AND TRADEOFFS

Meenatchi Jagasivamani

Doctor of Philosophy, 2020

Dissertation directed by: Professor Bruce Jacob

Department of Electrical & Computer Engineering

As DRAM faces scaling issues as a high-density memory, emerging technologies are being explored as alternatives. One promising candidate is Resistive Memories (ReRAM), which is scalable, vertically stackable, and because of the possibility of integration with standard logic process, can deliver higher density as a main-memory solution. The key differentiator with this approach involves a ReRAM memory array that integrates directly with a logic processor underneath.

In this research work, I explore ReRAM as a main-memory alternative at three levels of detail – at the device level, the physical-design level, and finally at the architecture level. I begin with an overview of ReRAM and compare with alternate technologies. I look at the physical design of the solution and present the results of area studies on integrating a VSCALE processor at the 45nm technology node with a ReRAM bit-cell array. The area study was performed based on parameters specified by my collaborators at Crossbar Inc. The results showed that the optimum operating point is at 50% array efficiency with a VSCALE processor, and that this configuration incurs an area penalty of 18%.

Two of the key challenges for ReRAM with respect to DRAM performance include the higher write latency requirement (typically on the order of 1us) and the lower write endurance (typically less than 10^8 cycles). This compares with DRAM write-latency times of less than 30ns (depending on technology node and generation) and write endurance of more than 10^15 write cycles. In this research work, I explore the possibility of utilizing the ReRAM cell in an intermediate state between non-volatile state and threshold state, where I intentionally tradeoff the write energy for a much lower data retention. This allows the chip to more easily replace existing DRAM-like main memory applications, without requiring higher write programming current or accommodating for a longer write latency. I performed this evaluation both at the device-level and at the architecture level.

At the device-level, I used UMD's Nano-fab lab to construct a Metal-Oxide based ReRAM bitcells on which I characterized the relationship between data-retention and write current applied. My fabricated ReRAM was composed of Titanium-Oxide and Aluminum Oxide. I also confirmed the behavior of a mixed-volatility state where a formed filament relaxes over time to move to a high-resistance level. Based on my experimental measurements, operating in the mixed volatile state would reduce write energy by 10 to 100x, and thereby improve the write endurance.

Finally, at the architecture-level, I used the Structural Simulation Toolkit (SST) to characterize a ReRAM-based main-memory system and compare with a DRAM-based one using our research group's DRAMSIM3 tool. I also characterized the sensitivity of various architectural parameters (core-to-memory controller ratio, queue depth, NoC topology) on system performance on stream and gups-based graph benchmarks which indicated that the torus topology will provide reasonable performance. Impact of the number of parallel processors indicated that at low processor counts, DRAM outperforms ReRAM due to its faster memory latency. However, at high processor counts, ReRAM with its higher number of parallel connections is able to deliver higher system performance than DRAM.

RESISTIVE RAM BASED MAIN-MEMORY SYSTEMS: UNDERSTANDING THE OPPORTUNITIES, LIMITATIONS, AND TRADEOFFS

by

Meenatchi Jagasivamani

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2020

To my parents, Vadivel and Rani Jagasivamani;
my husband, Guru Thuduppathy;
and my children, Gugan and Kayal Thuduppathy.

Acknowledgements

I would like to thank my family for supporting me in this journey and encouraging me to pursue the PhD after working in the industry. I am grateful to my mother for being my number one supporter, my father for instilling a deep sense of curiosity and appreciation for engineering, my husband for always believing in me to reach beyond my limits, and my children for filling me with joy each day.

I owe a great deal to Professor Jacob for being a wonderful research advisor. He was patient and encouraging throughout my PhD research, while having a long-term vision for the research work being explored. His suggestions in providing specific research questions to explore, while allowing me the intellectual freedom to discuss and pursue research areas of interest was paramount to the depth and breadth of my research study.

I would also like to specially thank Professor Yeung, for his valuable technical feedback and offering suggestions for improvement during the entire course of my PhD. Thank you to all of my committee members who provided me with their time and energy to further consider certain aspects during my defense.

Finally, I would like to thank my friends and colleagues at the University of Maryland, Shang, Brendan, Candace, Devesh, Luyi, and Daniel for their wish and support all these years.

Table of Contents

Table of Figures

Table of Tables

1 Introduction

1.1 Motivation and Problem Description

The memory bus is a major limiting factor to overall system performance. Current system performance is limited between the processing power's data needs and the data rate received by the memory system, with CPU request rate typically 3-4x faster than the data rate received from the overall memory system. System architects have come to accept the limitation due to the memory bandwidth wall and have focused on modifying memory access patterns and increasing parallelism in the computation layer in order to increase instruction throughput.

There are several mitigation strategies that are currently employed to address this problem. Hardware techniques include employing multiple levels of cache memory blocks. This relies on memory access requests being either temporally or spatially related, allowing for access requests to be serviced using data present in the cache blocks. Software techniques include prefetch to load specific data for an application or managing the access patterns by locating data in a predictable pattern in the memory. Finally, system-level techniques include introducing multiple memory controllers for bandwidth and incorporating high-bandwidth memories. While all of these techniques mitigate some of the issues, as the computational system becomes increasingly parallel, the memory parallelism imposes an upper limit on the overall system performance. Figure 1-1 illustrates a conventional system and depicts the problem.

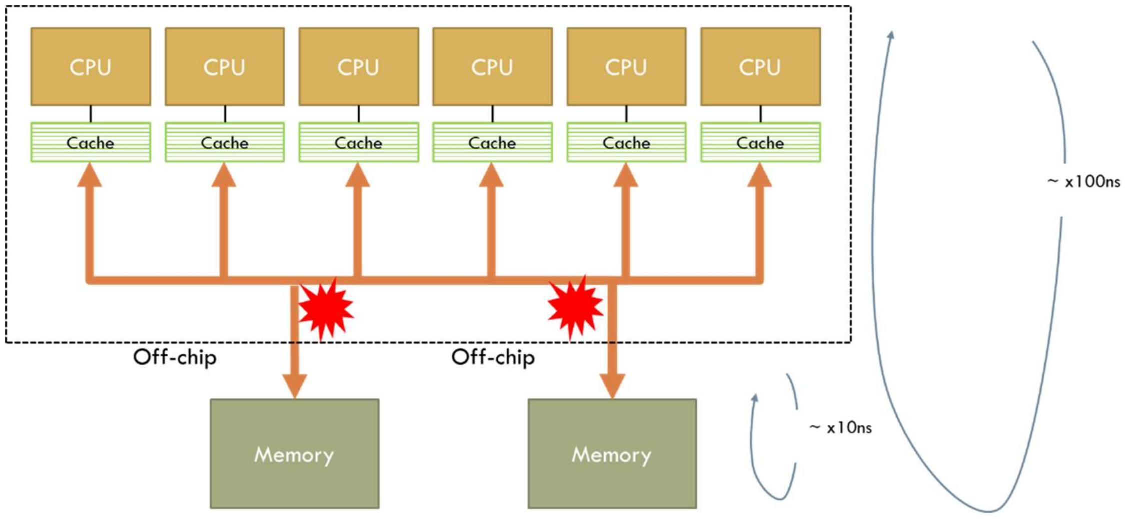


Figure 1-1 Motivation: Memory Bandwidth Wall

This figure shows multiple CPU processors that are embedded within a single chip, to perform the computations. These multiple CPUs generate several memory requests in parallel, often independent of each other. Each CPU is connected to multiple levels of cache memory blocks, often with the final level (Last Level Cache) being a shared memory block. The cache blocks attempt to service the data requirements of the CPU if the requested address is within the confines of the data contained with the cache. Any misses in the requested data would necessitate an access to an external main memory, typically DRAM, to fetch the data and fill the cache block. As can be seen in the figure, the external main memory is often located off-chip and are accessed through a few memory controller circuits embedded on chip. The memory controller itself has the ability to queue pending incoming memory requests, while the external memory is servicing the requests.

Figure 1-1 denotes six such CPU units, however modern systems could make use of close to 100 such CPU units. As the number of independent CPU or processor units increase, so does the number of independent memory access requests and the likelihood for a bottleneck at the memory controller. This causes memory requests from the CPU to be stalled while pending requests are serviced by the main memory. The result is that while DRAM device-level memory latency is on the order of 10s of nanoseconds, due to this bottleneck of memory requests, from the CPU's point of view, the perceived memory access latency ends up being much higher, on the order of 100s of nanoseconds for large parallel systems.

Thus, we can observe that the system is fundamentally limited by the number of wires that connect the processor and the memory chip. This bandwidth wall stems from the limited number of memory access points that exist in current systems. Due to the number of pins required to make a connection to an external DRAM subsystem (ex: DIMM), the DRAM memory controllers on-chip are often limited to six or eight per chip. Our proposed approach seeks to alleviate this bandwidth wall problem directly by utilizing a memory technology, ReRAM, that allows for higher numbers of access connections between the processor and the memory subsystems.

1.2 Proposed Approach

Emerging memory technologies are currently being explored by industry and academia to address both scalability concerns with conventional solutions and improved power-performance capabilities [1]. One promising memory technology is Resistive Memories which utilize creation of a high or low resistance state in a device to correspond to a digital value of 0 or 1. The resistance states are modified by creating a conductive filament in a dielectric material. The filaments could be either oxide-based (OxRAM) or metal ion based (CBRAM) and are controlled by applying specific high voltage or current pulse(s) of a specific shape [2,3,4]. In comparison with DRAM, ReRAM promises nonvolatility combined with better scalability, CMOS back-end-of-line (BEOL) compatibility, reasonable switching speeds for read, and higher density when stacked. Integrated Logic and ReRAM Integrated Circuits open the doorway for enabling more intuitive implementation of addressing the memory bandwidth wall problem without requiring complete redesign of long-standing software to hardware design techniques.

Our proposed solution for addressing the memory bandwidth wall described earlier involves using ReRAM as a main-memory replacement for DRAM and integrating it to the CPU logic on the same chip. This is different from 3D stacked-die types of approaches that make use physical integration of discrete dies, as shown in Figure 1-2. Our solution, which we call Monolithic Computer, involves the ReRAM cells residing in metal layers which are fabricated on the same die. This ReRAM technology has been demonstrated and fabricated in products from Crossbar, Inc who our research group is in collaboration with for part of this research work, as well as others in industry, such as Intel, Micron, and Rambus. Additionally, this approach enables extremely high parallel connections to the CPU and directly addresses the Memory Bandwidth Wall problem.

Current studies and research work focus on a specific material composition, with characterizations pertaining only to that area. A broad understanding of the technology, implications on how one parameter affects another, and the various tradeoffs involved is missing. Such an understanding allows wider adoption of this technology by computer architects to leverage the advantages into their design.

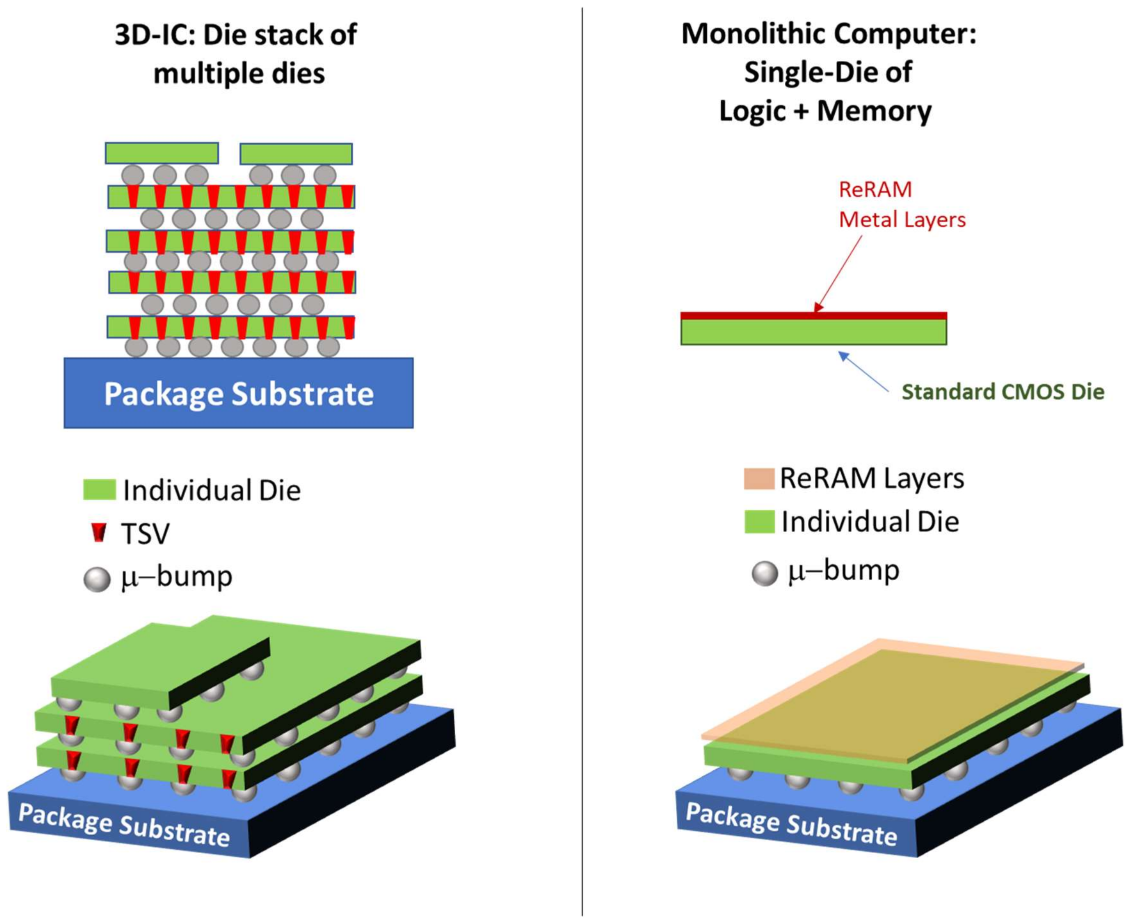


Figure 1-2 System Connection in Proposed Approach (Side and Corner View)

1.3 Contribution and Significance

In this dissertation, for background, I pull together research work done from different groups, both in industry and academia to extract the broad trends that emerge for this technology and draw together the various implementations of resistive memory to reveal design insights and architectural impacts. This is a literature survey of existing research on all variations of resistive memory technology, known by different names, such as ReRAM, PCM (Phase Change Memory), memristor.

For the experimental component, I begin with my results of design experiments performed using a collaboration with Crossbar Inc. Crossbar ReRAM utilizes a novel fabrication technology that provides integration capabilities with logic. Exploration studies on a specific ReRAM instance from Crossbar have been performed to understand the impacts on area, power, and bandwidth of integrating with a RISC-V processor. I have successfully established a methodology for physical floor-planning of a Resistive Memory layer on top of existing logic and present the area impact of a memory-processor architecture.

I also directly seek to address the high write latency and low write-endurance problem associated with ReRAM by characterizing the impact of write energy on the data-retention of the cell. My research thrust to support this goal involved fabricating ReRAM bitcells as test-cells using UMD's Nano-fab lab. I collected characterization data on these cells and characterized the relationship between data retention and write energy.

My final research thrust involved architectural simulations to quantify the impact of ReRAM write latency on various parallel simulations and evaluate the impact of additional memory hierarchies and non-regular NoC Topologies. To support this effort, I utilize Structural Simulation Toolkit (SST) to model and simulate different architectural configurations. My simulations indicated that despite the longer access time latencies of the ReRAM array, due to the much higher number of connections to the CPU logic, the ReRAM architecture is able to exceed the performance when compared to DRAM. A high enough number of memory access requests were needed where this advantage comes into play, with the crossover point for my simulation being 64 cores. My first order NoC topology comparison showed that typically torus and fat-tree configurations performed the best when compared with a mesh topology, with torus being 39% better and fat-tree being 70% better at the lower link bandwidths where the topology counts. Due to its ease of implementation, torus might be preferable over the other topologies as the link bandwidth increases, or as the number of cores increases.

1.4 Organization of Dissertation

The dissertation will begin with an overview of emerging memory technologies and a comparison of them. Here, I present the resistive memory cell operation and relationship between related memories such memristor and PCM. I present the key tradeoff pertinent to this technology in terms of area, program bandwidth, read performance, power consumption, long-term data-retention and reliability effects, and multi-level cell implementations. In the first part of my report, I present the detailed implementation of my area study, including the CAD flow to perform the study, and the results from my study. In the second part of my report, I present the premise of leveraging the non-volatile/volatile switching behavior of the cell, the device fabrication and characterization work, and present some of the preliminary results from my SST simulation. To sum, the thesis spans a broad range of topics and research techniques from physical design to device and circuit level, and to the architectural level.

High-level summary of the chapters are as follows:

- Chapter 2: Literature Survey and overview of Non-volatile memory technologies. Additional focus is given for the different implementation of ReRAM and alternate application space for this technology.
- Chapter 3: My motivation for using ReRAM as a replacement for DRAM as the main-memory, a more in-depth overview of the cell operation, and some of the device level challenges as a main-memory.
- Chapter 4: Floorplanning study of the die area impact of ReRAM integration with CPUs using Cadence and Synopsys design tools to perform the synthesis and digital implementation.
- Chapter 5: Device based characterization of a test ReRAM cell investigating cell behavior with lower program current and its effect on the data retention of the resistance state.
- Chapter 6: Performance studies (C and C++ based performance modeling using SST) comparing conventional DRAM based memory systems against ReRAM based main-memory system. Additional studies on the impact of parallelism are also presented. Finally, I

calculate the area required for some of the sub-blocks in the central ReRAM IP that I propose and provide a floorplan for the design.

- Chapter 7: Expanded architectural simulation work looking at different NoC topology and system configurations and the impact on performance. I also present effect of the number of DRAM memory controllers on the system performance in support of a Hybrid ReRAM-DRAM solution.
- Chapter 8: I talk about utilizing ReRAM in the volatile state to limit the data persistence and its possible application as a trusted on-chip main memory to improve overall system security.
- Chapter 9: Conclusion of the dissertation and research work
- Chapter 10: Bibliography of the Technical literature and references.
- Appendix A: Command File used in the Auto-Place-Route Physical Design study.
- Appendix B: Javascript code to perform architectural sizing calculations to estimate number of processors, and ReRAM blocks within a given chip size.

2 Emerging Memory Technologies

2.1 SRAM, DRAM (HMC, HBM, eDRAM), STT-MRAM, ReRAM

In this section, I provide a high-level comparison of current state-of-art and emerging memory technologies' capabilities. I begin with a brief overview of each of the technologies. Table 2-1 presents a summary of key parameters for the different memory technologies. I go over each of the memory technologies in detail.

SRAM: Static Random-Access Memory (SRAM) consists of a six-transistor (6T) bitcell with a back-to-back inverter pair tied to pass-transistors that allow access to the cell, as shown in Figure 2-1. The bitcells continuously maintain the data injected into the storage node. Data is statically maintained as long as power is supplied to the circuit. SRAM has one of the fastest access time at the expense of area overhead and typically serve as cache blocks on a chip.

	SRAM	**STT-MRAM**	**DRAM (HMC)**	**DRAM (HBM)**	**eDRAM**	**ReRAM**
Read Latency	1-10 ns	1-10 ns	~30ns	~30ns	100ns	200-800 ns
Write Latency	1-10 ns	10-50 ns	~30ns	~30ns	100ns	1-10 us
High Write Voltage Requirement (charge pump)	None	Yes, dependent on retention requirement (3v to 6v)	None	None	None	6v
Write-Endurance	1e16	1e13	1e16	1e16	1e16	1e6
Area	200 F^2	32 F^2	8 F^2	6-8 F^2	35 F^2	1-4F^2 (dependent on # of stacks)
Process	CMOS	CMOS + MTJ layer	CMOS	CMOS	CMOS (+Cap)	CMOS + ReRAM
Energy Efficiency	Moderate	Moderate	Moderate	Low	Low	20-30x lower than flash
Non-Volatile?	No	Yes, possible	No, refresh every ~10-100 ms	No, refresh every ~10-100 ms	No, requires refresh every < 100us	Yes

Table 2-1 Comparison of key parameters of Memory Technologies

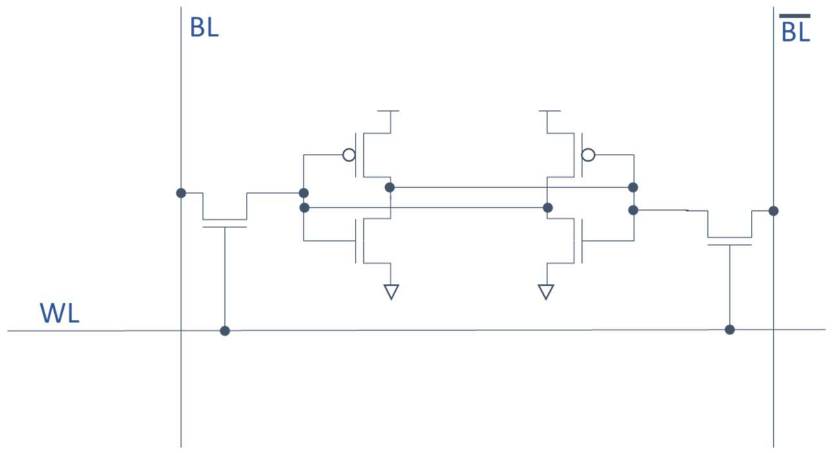


Figure 2-1 Conventional 6T SRAM cell

DRAM: Dynamic Random Access Memory (DRAM) bitcell is comprised of a capacitor whose charge is altered to store a data value of 0 or 1, as shown in Figure 2-2. Because the charge on the capacitor dissipates over time, a periodic write is performed to refresh the data on all bit-cells. An access transistor provides the mechanism to read and write the capacitor. DRAM provides fast read and write access times, but since it's typically located off-chip, it has limitations in achieving very high memory bandwidth and density at the same time. Also, DRAM technology based on the current implementations are projected to run into scaling issues at advanced process nodes.

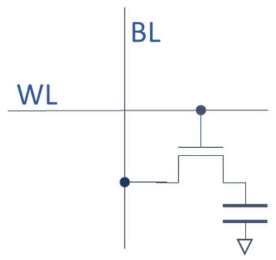


Figure 2-2 DRAM Bit-cell

Typically, DRAM memory is implemented as a separate stand-alone discrete die. Embedded DRAM (eDRAM) versions of the bitcells allow for the DRAM memory to be integrated on the same die as CPU but requires more expensive processing and take up silicon area. New DRAM architectures provide increased density by stacking several DRAM memory layers in a single chip. The two most common ones are Hybrid Memory Cube (HMC) by Micron and High Bandwidth Memory (HBM). HMC is developed by Micron to provide a discrete high-density DRAM memory chip consisting of 3D-integrated stacks of DRAM Memory dies. HBM is an open-standard high-bandwidth DRAM memory stack that requires a silicon interposer to connect the DRAM to a CPU/GPU die.

STT-MRAM: Spin-Transfer Torque (STT) Magnetic RAM (MRAM) bitcell is comprised of a magnetic tunnel junction (MTJ), where the direction of magnetic moments and the spin direction of the electrons determine the state of the bitcell, as shown in Figure 2-3.

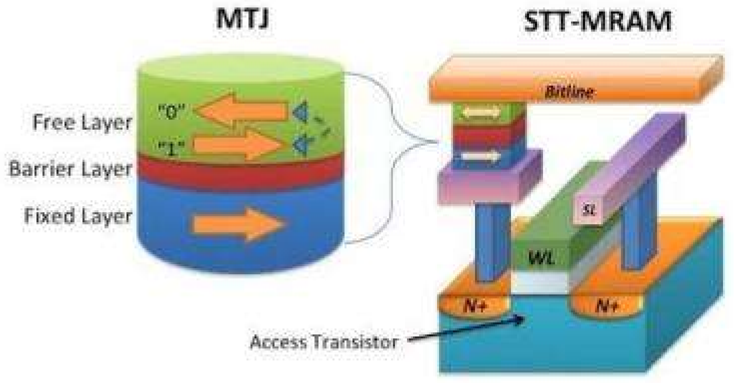


Figure 2-3 STT-MRAM Bitcell ***Figure source:*** **(MRAM-info, 2016)**

Because of its CMOS compatibility, this bitcell could be integrated into standard manufacturing process and could deliver high density with non-volatile data retention.

2.2 ReRAM Implementation Variations

There are several variations on the exact resistive creation mechanism based on the materials used [7,8,9]. The three major versions are:

(1) **CBRAM**: Conductive Bridging RAM which relies on the creation of microscopic conductive filaments through metal-ion migration;

(2) **OxRAM**: Creating Metal-oxide physical defects which results in conductive paths of varying resistances in a layer of oxide material by causing a valence change.

(3) **PCM**: Phase Change Memory which changes the crystal structure of a chalcogenide glass from amorphous to crystalline, thus altering the resistance of the material.

PCM is constructed using a heater material, such as tungsten (W), which has a high resistivity and emits heat to its surrounding. The chalcogenide material is placed on top of the heater and a current is passed through structure to apply a high temperature (close to 600K) to melt the material. By lowering the programming current slowly, we anneal the material to cool slowly and settle into a crystalline structure which has a lower resistivity. Alternatively, by abruptly bringing down the program current, we quench the material and the resulting structure is amorphous and highly resistive in nature. Thus, the resistivity of the material is altered, and the data state is represented as the resistance value. A select device is needed in conjunction with the PCM cell so that a single cell can be "selected" among an array of cells.

Up until recently, most PCM implementations used either a MOS transistor or a buried PNP-BJT as the selector. This prevented PCM array itself from being stacked vertically. Additionally, PCM cells were observed to have a drift phenomenon, where the natural state of the material eventually drifted towards a crystalline structure (low-resistance), which is especially problematic for multi-level cell behavior. Device engineering work, along with a new selector that can reside in the metal layers are being investigated to circumvent this problem. In comparison to PCM, ReRAM have not been reported to be prone to data disturb from signal lines adjacent or underneath to the memory bitcell. The work described in this thesis covers OxRAM and CBRAM implementations, both of which work on creating a conducting filament and altering the overall resistance of the material.

To provide an overview of existing ReRAM implementations, I performed a survey of reported specifications of different Resistive Memory implementations based on published data. Table 2-2 summarizes the performance metrics. Several of the implementations are partnerships between design companies working closely with a semiconductor manufacturing fab to realize high-volume implementations of ReRAM cells.

Organization	Capacity	Process (nm)	Structure	Area (mm^2)	Density (Gb/mm^2)	Cell Size (F^2)	Read Latency (uS)	Write Latency (uS)
Intel/Micron 3D Xpoint	16Gb	20	PCM+OTS between Metal 4 & Metal 5	206.5	0.62	4.4	8	30
SanDisk/Toshiba ReRAM	32Gb	24	OxRAM Metal-Oxide based with Diode selector	130.7	1.958684	7	40	230
Micron/Sony ReRAM	16Gb	27	CBRAM based CuTe Alloy+Buried MOS selector	168	0.761905	6	2	10
Crossbar ReRAM stacked	4Mb	40	1TnR	test chip		5.6	0.5	10
Crossbar ReRAM 1T1R	16Mb	40	1T1R, 9 metal	test chip			0.02	10
Adesto EEPROM	512kb	40	CBRAM	test chip		118	1.2	60
IBM/Macronix PCRAM	-	90	PCM based with MOS selector	test chip	-	20	0.0375	0.13125

Table 2-2 Key performance metrics of various ReRAM implementations

Figure 2-4 below plots the bitcell size comparison of the different implementation against the process node. SRAM and DRAM metrics are also provided for comparison. The cell size is reported in feature-squared (F^2), which denotes the multiplication of the smallest feature size achievable in that particular process node. The value of F is a critical technology parameter defined as the minimum polygon that can be fabricated in that process node and is typically limited by the lithography of the process. It is often used as the minimum achieved gate length of the transistors. The figure highlights the bitcells based on ReRAM technology.

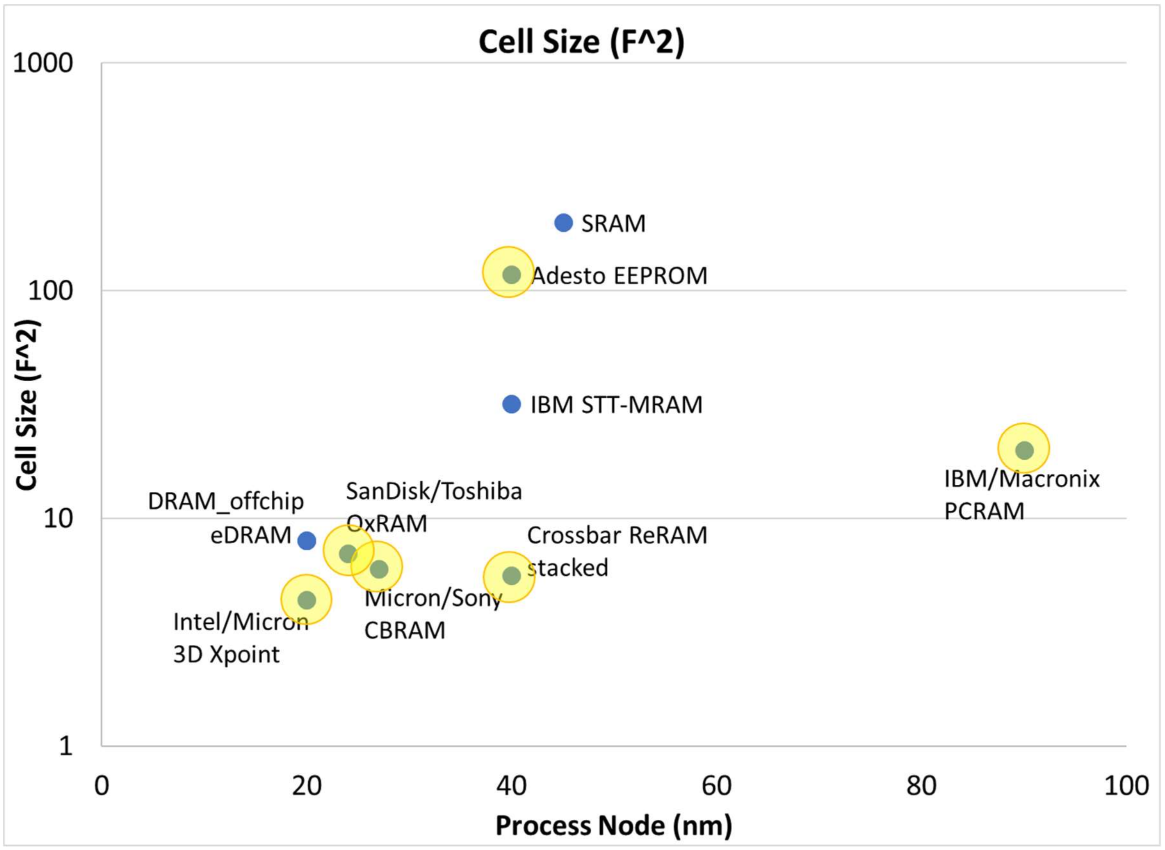


Figure 2-4 Cell-Size Comparison for different Memory Technologies

From the plot, we can observe improvement in cell size as we scale to advanced process node, largely through innovations vertical stacking. One exception to this is the Adesto EEPROM product which uses a PCM bitcell with a MOS selector and is not stackable. This product targets low-power IoT applications and the high-area is sufficient for the low-volume product. For the other implementation of ReRAM cells, I see the feature size to be lower than DRAM bitcell. Note that the Crossbar ReRAM bitcell is based on a two-layer stack but is expected to be vertically scalable to up to 8 stacks, which would further reduce the bitcell size. STT-MRAM occupies higher area compared to most ReRAM implementations, but it’s read and write latencies, which are on the order of SRAM latencies, are much lower than ReRAM. This memory technology could be a competitive alternative for on-chip cache application to replace much the higher area cost of SRAM cells.

Read latency comparison among ReRAM implementation is presented in Figure 2-5. ReRAM bitcells have higher read latencies when compared with the other technologies. The exception here is the Crossbar 1T1R ReRAM, which reported a read latency of 20ns and was targeting a high-speed embedded memory application. This product was implemented in 40nm 9-metal process and had a total capacity of 16Mb. This implementation used a transistor as the selector device, and therefore would not be stackable. From the plot, we can observe a slight increase in read latency with advanced process nodes, however, as can be seen in summary Table 2-2, this is more due to the capacity of the memory rather than advances in technology. As the technology matures, we can observe that ReRAM transitions from EEPROM type of memories that require lower capacity to Intel’s 3D Xpoint memory with higher memory needs. The higher capacity is supported by larger arrays, which often requires higher latency times. I discuss this phenomenon in more detail in the device tradeoff section 3.3.

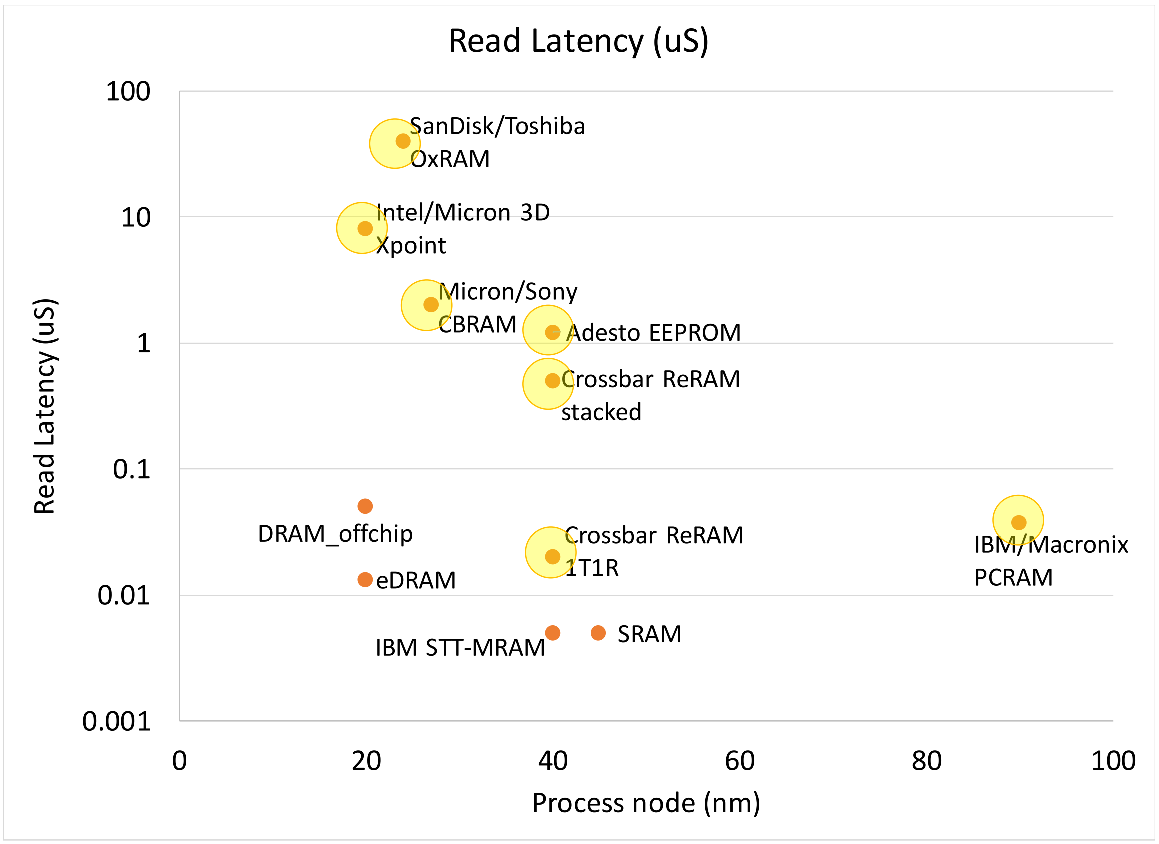


Figure 2-5 Read-Latency Comparison for different Memory Technologies

Write latency comparison among ReRAM implementation is presented in Figure 2-6. ReRAM bitcells, being non-volatile memory, have higher write energy requirements, which also translates into higher write latencies. The IBM/Macronix PCM reported a lower write latency of 131ns on a test-chip product. Although not reported in this chapter, the overall write energy also tends to be higher and leads to lower write endurance when compared to volatile memory technologies. In chapter 5, I discuss the device-level challenges in adopting ReRAM as a main-memory replacement for current computer architectures. ReRAM is a Non-Volatile memory with higher write energy and latency requirements than DRAM cells. ReRAM targeted for main memory applications needs to be engineered to support shorter latencies and higher write endurances.

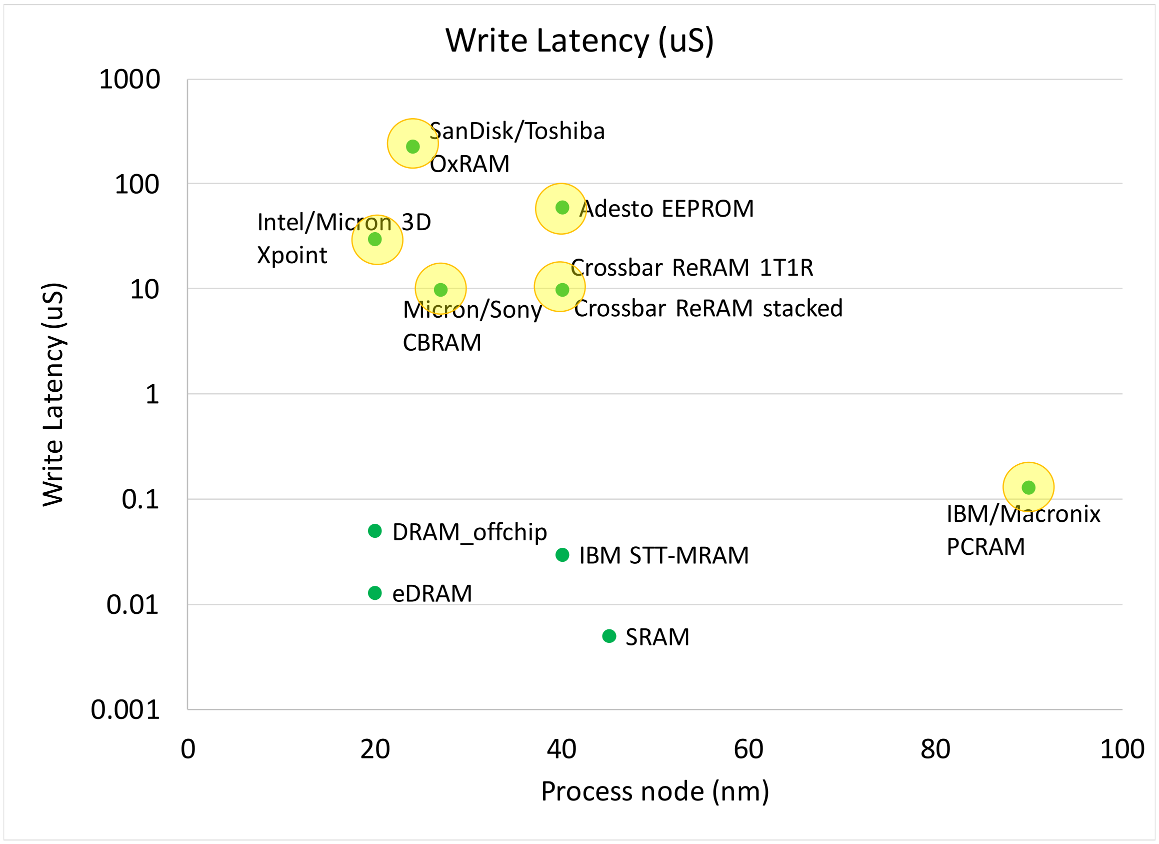


Figure 2-6 Write-Latency Comparison for different Memory Technologies

2.3 Applications for ReRAM technology

The ease of integration and low-bandwidth characteristics of ReRAM readily lends itself to be used in applications that require high parallelism with fine access granularity. Parallel multi-processor architectures meet this criterion and could be implemented using the mesh architecture topology that I mentioned in the previous section. Such parallel multi-processors are most suited for computation intensive programs that can be expressed as SIMD (Single Instruction Multiple Data), MODS (Monolithic Operations, Distributed Data), or DODS (Distributed Operations Distributed Storage). These architectures consist of an array of tiles, with each consisting of a modest processing core, supporting ReRAM memory, and NOC switch to support inter-tile communication. Each unit should be capable of functioning as autonomous processing unit, with individual processors having modest computation power. Collectively the mesh architecture could provide higher power efficiency on computation intensive programs (SIMD, MODS, DODS).

2.3.1 ReRAM with Support Logic Circuits

Resistive Memories is an emerging technology that has huge promises in terms of scalability, integration with logic, and helping address the memory wall problem. However, it has limitations in stream bandwidth and write endurance making an augmenting memory component a suitable transition, rather than a replacement for currently existing SRAM or DRAM cache needs. One of the biggest advantages with ReRAM comes from the fact that ReRAM has the potential to support computation-in-memory because you can fit in much more complex logic underneath the memory layer and it can still be at near sense-amp pitch. In this section, I look at applications that take advantage of the proximity to the memory that the higher BEOL allows in enabling the creation of more powerful ReRAM that allows us to build custom sense-amp pitched logic to support certain data-intensive applications. Some potential circuits to integrate could be buffer circuits to increase bandwidth and embedded hardware accelerators to create processor-in-memory like features. These accelerators would be supporting floating point operations.

I present two augmenting logic to tailor ReRAM for an application that researchers are looking at. To overcome the bandwidth limitation for streaming intensive applications, I can place register banks that shift in data from ReRAM and provide a single wide data output. For example, suppose a ReRAM memory array has read bandwidth of 8 bits per read, which implies 8 Sense-Amplifier columns. If my target bandwidth is one 128bits per read, and if the read time is dominated by partly by wordline selection time, then selecting one row and reading a 128-bit "page" at a time to accumulate into "shift register" would optimize some of the read-time overhead. This register bank buffer could be placed directly underneath the ReRAM memory resulting in no additional area requirements.

Figure 2-7(a) illustrates this using a simple augmenting logic using an output buffer to increase the perceived data bandwidth. This approach is similar to DRAM stream read access, where a single read outputs 128-bit granularity by switching the column multiplexer and selecting subsequent columns in a single row. Another simple augmenting logic to be considered includes floating point computation logic that supports several signal processing and matrix computation applications. As shown in Figure 2-7(b), this would involve pipelined floating-point computation logic (Multiply-Multiply-Add) folded directly underneath the ReRAM memory. A read request from one or more arrays would feed a pipelined floating logic block to perform the computations as successive reads are performed in parallel. This approach is similar to the FPGA pipelined approach used to embed accelerators into the FPGA fabric.

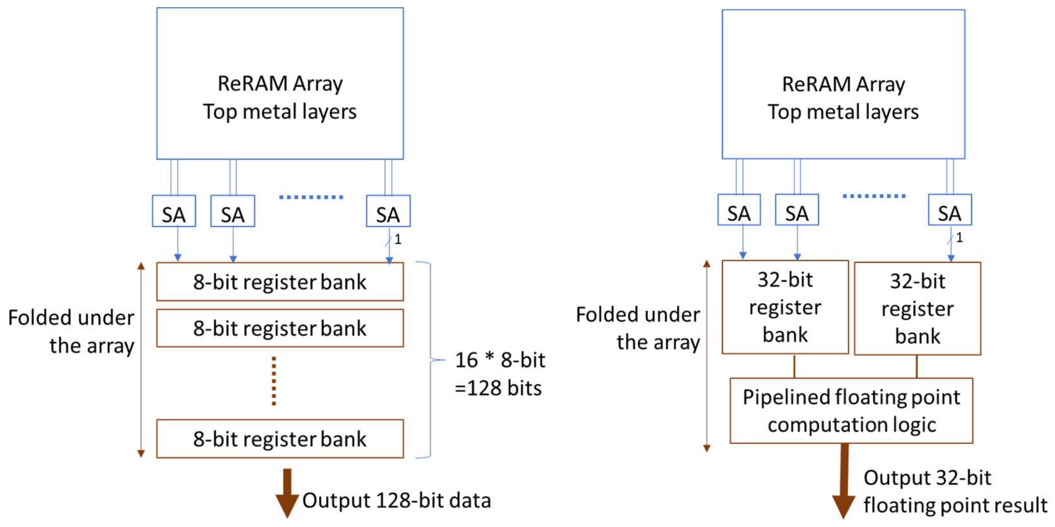


Figure 2-7 Augmenting Logic to enable ReRAM adaption into key applications (a) output buffer to increase data bandwidth (b) pipelined floating point logic to enable computation

Finally, non-volatile logics are a group of circuits that make use of the non-volatility in ReRAM to preserve the state of logic when a chip goes to deep power-down modes. Because of the ease of integration that ReRAM allows, logic states of key circuits could easily be preserved in ReRAM memory residing above. Due to the intermittent power availability of IoT (Internet on Things) devices, non-volatile logics are being explored as an application for ReRAM technology.

2.3.2 ReRAM for Super Conducting applications

ReRAM could have an application for Super Conducting circuits due to their non-volatility. Super conducting circuits make use of a RSFQ (Rapid single flux quantum) type of logic, as opposed to traditional CMOS logic. This type of circuit relies heavily on capturing spikes that propagate through the system to achieve the different logic functions. Figure 2-8 shows an example RFSQ circuit where signal from BLK1 is transmitted to BLK2 as a spike. The input signal, i, appears before the clk signal spike in order to latch in the input signal. A Joseph Junction device, depicted as X in the figure, is used to maintain the signal until it is consumed by the BLK2. However, these RFSQ gates consume the input token/charge, and a lot of care is taken to path-balance and synchronize the arrival of all inputs. In order to combat this, conventional techniques [24], implement special Non-Destructive-Read-Out (NDO) circuits that implement persistence of the input signal.

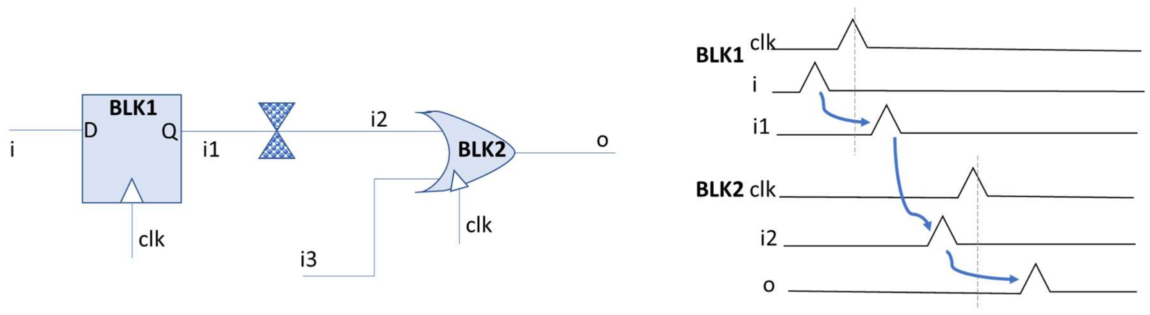


Figure 2-8 – RFSQ Circuit

Although MRAM type of memory technologies are being studied as a possible application for this, ReRAM could be a better alternative due to its integration with traditional fabrication technologies. However, it's higher write energy and write latency make it a device challenge in adopting in this application. One interesting study to explore is the possibility of making use of ReRAM, to implement the persistence instead. By using a ReRAM with a very low data retention time, it could translate to a lower write-energy requirement, as long as the written spike need not be maintained for very long time (not non-volatile behavior). In chapter 5, I explore using ReRAM in this mixed volatility state to tradeoff data retention with write energy and write endurance. One challenge in this solution might be that the temperature range that these circuits would operate in might limit the material composition of ReRAM to be used. Further exploration is needed to evaluate the feasibility of this solution.

3 ReRAM Background

3.1 ReRAM as DRAM alternative

There is currently one DRAM limitation that system architects have to work around. This is the tradeoff between bandwidth and capacity, illustrated in Figure 3-1. The x-axis in the graph is the peak bandwidth rated for the device, measured in GB/sec. The y-axis is the typical total memory capacity available for that particular implementation.

Conventional DDR4 implementations are capable of high storage capacity, close to 400GB. Their bandwidths on stream triad benchmarks are reported as below 100GB/sec. The stacked DRAM implementation, on the other hand, has a high bandwidth close to 500GB/sec measured with the stream triad benchmark. However, their capacity maybe quite low, on the order of 16GB in total. While these bandwidths are for sequential dense access patterns, the effective bandwidth drops dramatically for sparse access patterns to below 100GB/s.

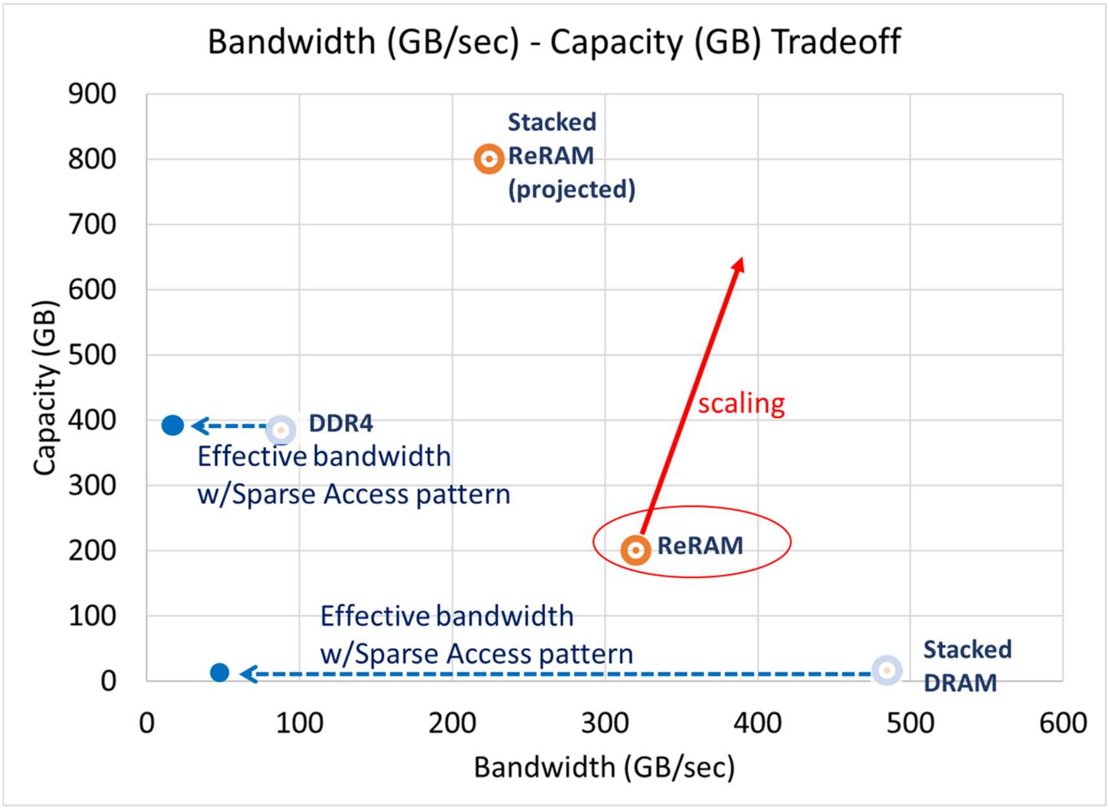


Figure 3-1 DRAM Bandwidth-Capacity Tradeoff

While off-chip DRAM can provide significant capacity, the bandwidth is low. The option of increasing aggregate bandwidth using additional chips has a high-power penalty (~2-4W per DIMM). Implementations such as Stacked DRAM provide high-bandwidth, but with low-capacity. DRAM bandwidth also is targeted for dense-type of memory access patterns and degrades severely in sparse type of access patterns.

ReRAM, on the other hand, can provide a much higher bandwidth at higher density. ReRAM has a low access granularity of 8B and can sustain the bandwidth for both dense and sparse memory access patterns. The graph shows the projected

ReRAM bandwidth of 320GB/sec at a capacity of 200GB, based on current capabilities. Additionally, I project that by stacking the ReRAM devices vertically in a 3D-IC could increase the capacity dramatically with a small impact of the overall bandwidth.

In addition to the bandwidth-capacity tradeoff, DRAM is also reported as facing scaling issues and being vulnerable to failure at advanced technology nodes. ReRAM on the other hand has been fabricated at 28nm technology and shows no issues of being scalable beyond 7nm. This allows the ReRAM memory to scale with advancements in the processor and logic technology and could further improve the capacity and bandwidth of the memory.

3.2 Overview of Resistive Memory and Cell operation

ReRAM stands for resistive Random Access Memory, where the resistance of a material is varied by applying different voltage/current across the material, and the resistance is used to indicate a data value of 0 or 1. In this section, I present a brief overview of ReRAM characteristics. Resistive memories have two main components: the selector, and the resistive storage element. The ReRAM bitcell's basic storage mechanism of operation involves the use of dielectric materials which normally don't conduct current. A dielectric breakdown is induced by subjecting the material to a high enough current or voltage, which typically causes permanent damage to the device in other dielectric devices, such as diodes and capacitors. The ReRAM materials are engineered in such a way so that this dielectric breakdown does not cause permanent damage and is reversible.

Figure 3-2(a) shows the cross section of the 1S1R (1 selector per 1 resistive element). During read, the voltages are expected to operate in the nominal range for the technology, while write voltages are expected to be pumped to a higher voltage level. This Crossbar ReRAM implementation does not utilize a separate access transistor for selection, but the selector device is integrated with the resistive element to form the switching medium (SM) layer for the bitcell as shown in Figure 3-2.

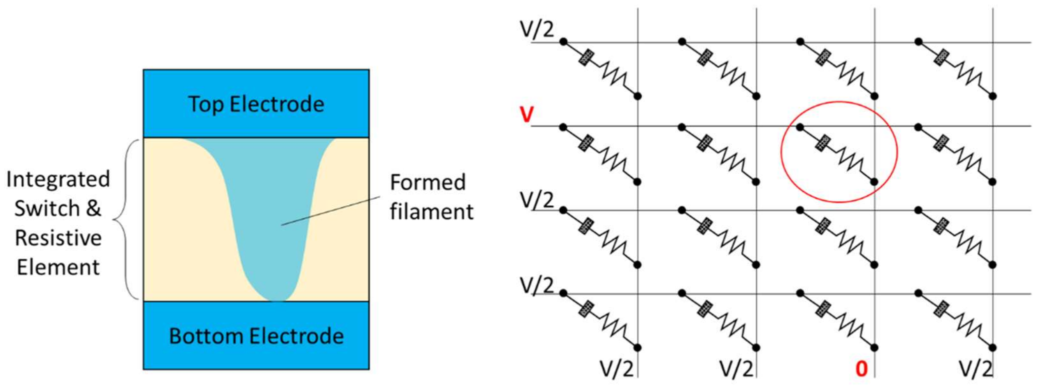


Figure 3-2 ReRAM Bitcell Details (a) ReRAM bitcell cross-section (b) Crossbar 1S1R array bias scheme, with selected cell circled

The SM is sandwiched between the bottom electrode (BE) and the top electrode (TE). A voltage above a threshold (> V_{TH}) is required to select the cell to perform a read or write operation. For the program operation, a much higher voltage (>V_{PRG}) is applied to enable the formation or resetting of the conductive filaments. Figure 3-2(b) shows the bias scheme of the crossbar memory array for selection. All wordlines and bitlines are held at V/2, while the selected cell's wordline and bitline are biased to have a difference of V across it. The selector device is engineered so that the ratio between the ON-resistance, defined as when the bitcell has a high voltage bias (V) across it, and the OFF-resistance (voltage bias of V/2 in this example), is very high. This high selectivity ensures minimal sneak path current on unselected cells on the same bitline, which have a potential of V/2 across their cells.

3.3 ReRAM Read and Write Performance Tradeoffs

The read latency of a ReRAM array is dependent on the overall array size, as shown in the graph in Figure 3-3. The x-axis in the graph is the sub-array size of the memory array, which is the product of the number of rows and columns with an array. The graph has two y-axis – overall die area required to meet a certain memory storage capacity measured in sq mm and read latency delay measured in micro-seconds.

For the purpose of area efficiency, it is desirable to have as high an array size as possible. This is because having several smaller arrays would increase the overhead to the surrounding peripheral circuits, such as the row and column decoders. Although the overall sizing of the individual drivers could be smaller for the smaller array, the overall area needed would be higher since there would more of the decoder logic.

Additionally, by separating the arrays into small sub-arrays, certain duplication of control and sensing circuits becomes necessary, adding to the overall overhead area.

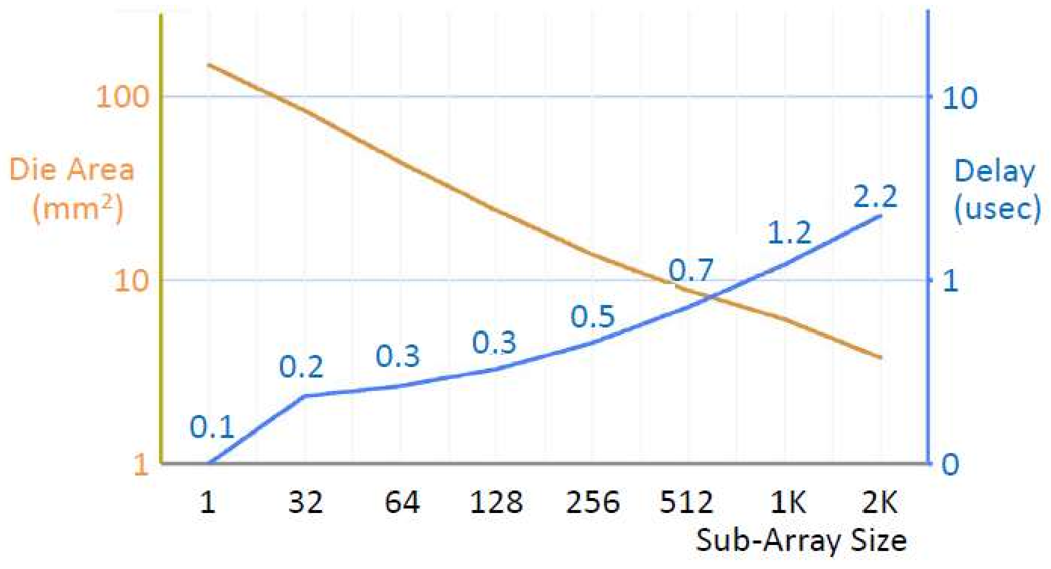


Figure 3-3 ReRAM Array Size vs Read Latencies

The graph shows that read-latency delay (marked by the latency numbers), increases as the size of the sub-array increases. A very small array of a single bit (1), can have an expected delay of 0.1uS, or 100ns, while a very large sub-array of 2000 bitcells can have an expected delay of 2.2uS. From the die-area point of view, the small sub-array of a single bit would incur a high die-area of 100mm^2, while the large sub-array of 2K would have a die-area of 3mm^2. Thus, there exists a strong tradeoff between array performance and the area. This high dependency is due to the latency timings largely being dominated by the parasitic elements (Resistance, Capacitance) of the wordline and the bitline. A shorter array reduces the length of these lines, and therefore directly helps to reduce the latency of the memory.

To consume shorter latency, the array size needs to be kept small, which results in lower area efficiency. In order to match read latencies close to DRAM main-memory, this tradeoff between array-size and area efficiency could require a smaller-bank based architecture to increase the read bandwidth, at the expense of die-area. On the write-latency side, these are much longer when compared to DRAM write-latencies due to the non-volatile state change of the bitcell. For write-latencies, a separate write-back cache could be used as a solution to buffer write operations for certain applications.

3.4 ReRAM Write Endurance Challenge

Conventional ReRAM bitcell write endurances are on the range of 10^5 to 10^8 cycles, while typical DRAM write endurance is greater than 10^{15} cycles. Figure 3-4 compares the write endurance ranges of DRAM against ReRAM. Write endurance reflects the durability of the bitcell for write operation and is measured in the number of write cycles. ReRAM bitcell, as it is, is over 7 orders of magnitude lower than DRAM.The large difference in write endurance limits between ReRAM and DRAM is a critical device challenge for ReRAM. If the write endurance limits are limited to 10^5

cycles (100,000 cycles), then typical applications that make use of main-memory would not be supportable using ReRAM as a direct replacement for DRAM.

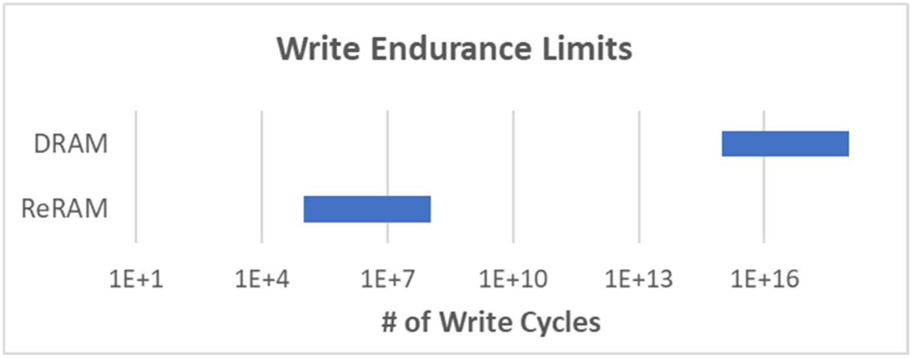


Figure 3-4 Write Endurance Ranges for DRAM vs ReRAM

DRAM write energy on average is around 19pJ/bit, while ReRAM write energy is quoted as 65pJ/bit. I expect that write endurance has a tradeoff with data Retention that could be leveraged for main-memory applications. Additionally, this tradeoff could have benefits in lowering the write energy requirements, which is the second device challenge I mentioned.

4 Area Exploration studies

One of the most common versions of ReRAM memory involves a “crossbar” structure of two orthogonal strips of wordlines and bitlines, the intersection of which produces both the resistive storage element and the selector device. Figure 4-1 shows the Crossbar’s version of the ReRAM bitcell being comprised of the selector device and the memory cell, both of which are sandwiched between the orthogonal bitlines and wordline signal lines. This pattern can be continued to provide vertical stack-ability of the memory, thereby increasing the effective density.

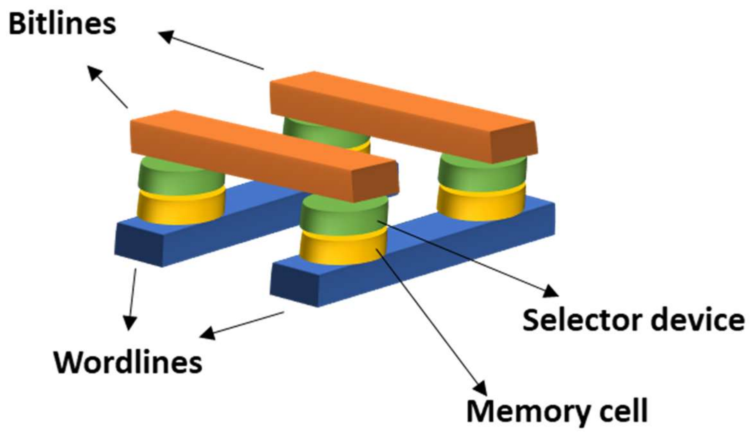


Figure 4-1 Cross-Section ReRAM bitcell

These memory layers are fabricated on BEOL metal layers and can be stacked to provide increased memory capacity and density. Additionally, these can be integrated onto traditional CMOS processes, allowing for logic or ASIC circuits to be placed in certain regions under the memory. In comparison to separate vertical high-density memories, this technology helps manufacturers circumvent some of the challenges with existing 3D ICs, including higher development costs, and reliability with the TSV fabrication. An IC with integrated Logic and Memory layers (see Figure 4-2) increases the function per unit volume/area while reducing power consumption significantly. The crossbar version of ReRAM memories is stackable and allows for logic to be placed under the memory layer.

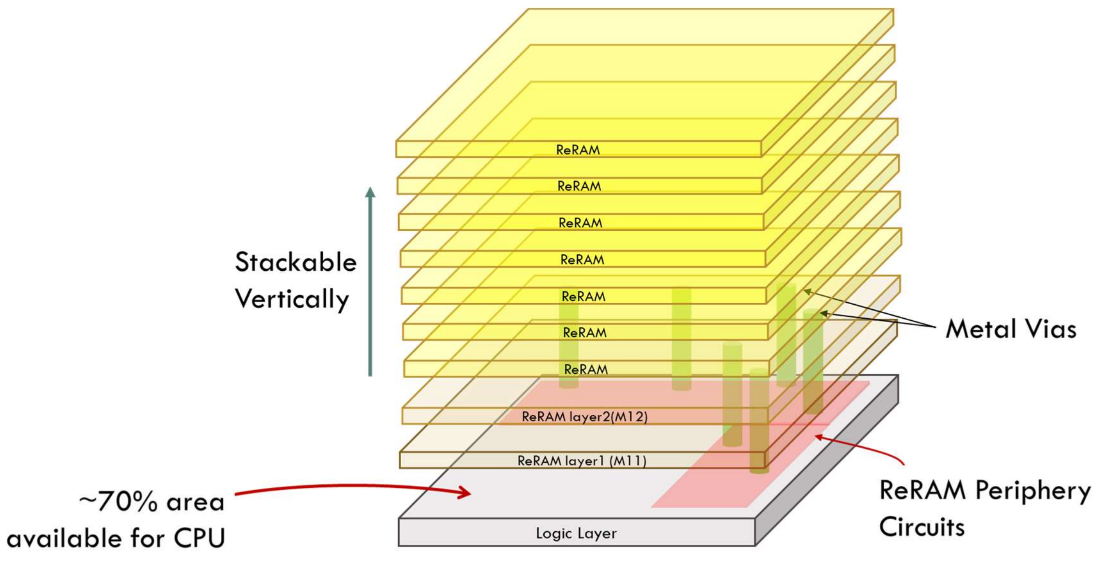


Figure 4-2 ReRAM Physical Integration.

As shown in Figure 4-2, while the actual memory cells are in a BEOL metal layers, the peripheral circuits – such as, the word line decoder, column multiplexer, and sense amplifier, would need to take up space in the substrate and lower metal layers and forming blockage regions for logic circuits. However, this still leaves a majority of unused space under ReRAM Memory Stack (more than 70% for a two-layer stack). I propose using the unused space under ReRAM metal stack for CPU or other Logic elements.

My aim for the physical design feasibility study was to explore a monolithic processor core that can be physically integrated with a ReRAM memory on the same chip. In this section, I attempt to integrate a standard-cell based synthesized RISC processor circuit with a ReRAM crossbar memory circuit and analyze the area and routing congestion that results from such an integration. I first present some of the ReRAM integration constraints and the CAD methodology used to study the area impact. I consider three different ReRAM integration and summarize the measured results. Two different integration ReRAM-Processor configurations are presented in this section with the area impact results obtained. Finally, I consider the physical integration of a SRAM memory placed underneath the ReRAM array layout.

4.1 Crossbar ReRAM Integration Constraints

My initial study is based on the Crossbar implementation of ReRAM memory which is CMOS compatible and back end of line (BEOL) stackable. CMOS compatibility ensures that the exotic materials used for the ReRAM stack can be deposited on top of standard CMOS fabrication techniques. One method of physically realizing this type of integrated circuit involves a two-step process, where the CMOS circuits are fabricated at a standard process foundry and then taken to a ReRAM fabrication facility for the specialized ReRAM layers to be deposited on top, in a split-fabrication like approach. Figure 4-3 presents the physical implementation of the ReRAM bitcell into a standard CMOS process.

Crossbar ReRAM uses a Select device embedded with the Resistive cell (1S1R) and the cells lie at the cross point of orthogonal metal layers, as shown in Figure 4-3 (a). The Figure 4-3 (b) shows the split-fabrication like approach described earlier, where the specific ReRAM layers can be embedded on-top of, or even in the middle of standard fabrication processes.

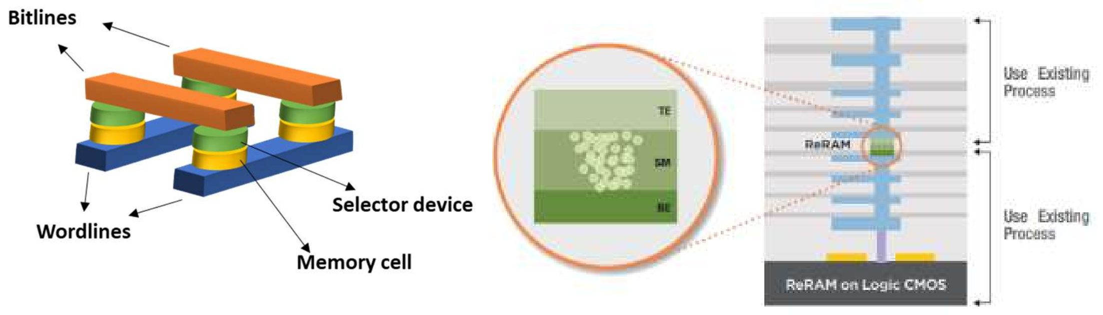


Figure 4-3 Crossbar ReRAM Bitcell (a) Orthogonal Bitcell Layout (b) ReRAM integration with CMOS Process *Figure source* (Crossbar Inc., 2018)

Table 4-1 summarizes the key performance metrics of Crossbar's ReRAM array. I will briefly go over each of these characteristics and compare with DRAM performance where applicable. The bitcell area is competitive with a DRAM bitcell and has the potential to achieve higher density with increased vertical scaling. Also, as noted in the table, the bandwidth per array is 4-8 bits. Therefore, to provide sufficient bandwidth to a single core, I envision several arrays that are distributed across the full-chip and are accessed in ganged mode, in a Distributed Shared Memory-like architecture.

Key Parameter	**Performance**
Area	4-16 F^2
Bandwidth per array	4-8 bits
Read Latency	200-700 ns
Write Latency	1 us
Cell Leakage	0.1 nA/cell
Program Energy	10-100 pJ/cell
Endurance	$> 10^5 – 10^8$ cycles
Retention	> 7-10 years
Scaling Potential	< 10 nm
Ron/Roff ratio	100
Selectivity (ΔI @V_R, $V_{R/2}$)	$> 10^6$ - 10^{10}

Table 4-1 Crossbar 1S1R ReRAM Parameters

Some of the critical parameters that pose a device challenge for ReRAM replacing DRAM as a main-memory are the latency and write endurance limits. Both read and write latency times are much higher than typical DRAM times, with write latency being especially much higher. The Ron/Roff ratio in the table is a characteristic of the selector device engineered by Crossbar, Inc. The crossbar ReRAM bitcell has a high Ron/Roff ratio over 100 to reduce sneak path currents from unselected cells and a low cell leakage current.

The program energy per bit is also significantly higher than DRAM, and consequently, the write endurance for ReRAM is expected to be around 10^5 – 10^8 cycles, which is much lower than that of DRAM, which is quoted to be above 10^15 write cycles. This is a critical device challenge to be overcome in order to replace DRAM for typical applications. The flash memory bitcell, however, has a much lower write endurance of 10,000 to 100,000 cycles. This low write-endurance is is managed by wear-leveling techniques to minimize the number of write operations to any particular cell, along with flash memory's application consisting largely of read operations. ReRAM has high scaling potential, however, and is expected to be scalable below 10nm.

Crossbar ReRAM technology integrates with standard logic processes, is stackable vertically for increased density, and has a 1-4F^2 cell size, depending on the number of stacks. Although not all ReRAM variations in development allow for this assumption, the general direction of ReRAM is moving towards increased density by utilizing vertical scaling and integration with logic-process compatibility. This type of ReRAM is organized so that the bitcells are stacked on higher metal layers which are shown in Figure 4-2 as M11, M12, as an example. The bitcell layout of the ReRAM could be simplified as a cross-section of adjacent metal lines, whose intersection determines the location of the resistive storage element. Figure 4-4shows the bitcell layout in 45nm technology used for my area study. The ReRAM bitcell dimension I am using for the array is $1.4(2*\lambda)^2$ which is 106nm x 106nm at 45nm technology.

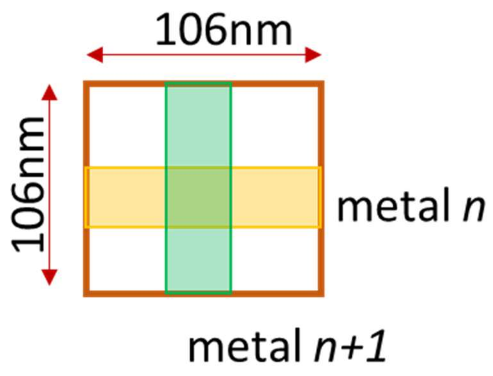


Figure 4-4 45nm ReRAM bitcell

The peripheral support circuitry for the ReRAM to perform the address decode, row and column selection, and sense amplifier read and verify circuits would be implemented in the substrate using standard CMOS layers, such as the diffusion, polysilicon, and some of the lower metal layers. Embedding these peripheral circuits into a processor circuit would have an area cost and is one of the focus of my area study. Processor circuits are implemented using an Auto-Place-and-Route (APR) tool. This tool takes a high-level design description netlist, such as VHDL or Verilog synthesized netlist, and places the standard cells in order to meet timing and minimum area goals. For my area study, I assess the impact of embedding the ReRAM peripheral circuits into a processor logic. In traditional digital implementation flow, I model these peripheral circuits with a blockage layer to indicate to the APR tool that standard cells may not be placed in this region.

Figure 4-5 shows the memory array organization that can be formed to group together multiple arrays and provide sufficient data bandwidth. A single ReRAM array, shown on the left in the figure, consists of bitcells arranged in several rows and columns. A single horizontal row, referred to as wordline, is selected during a read or write access by wordline (WL) decoders. Multiple columns, also called bitlines, are sensed through a column multiplexer (MUX) which is often placed below the array. A sense amplifier (SA) compares the current sensed on the selected bitlines against a reference current to decide on the data read out. This is done for both read and write operations, as write-operations often involve a verify step to ensure that the write pulse was able to successfully place the cell to the desired state. As can be seen in the diagram, these peripheral circuits form a L-shape on the side and bottom of the array.

The physical layout of four single arrays is grouped into a mat, shown in Figure 4-5 on the right. The four arrays are rotated so that their peripheral circuits are placed next to each other. This organization allows for sharing of control signals between the arrays during an access. The peripheral circuits make use front-end-of-line (FEOL) layers, such as the ones needed to create the transistors (diffusion, polysilicon, contact), as well as the lower metal layers to connect the CMOS logic together. The ReRAM array itself only uses BEOL layers, and the area underneath is available for the CPU logic, as I mentioned before.

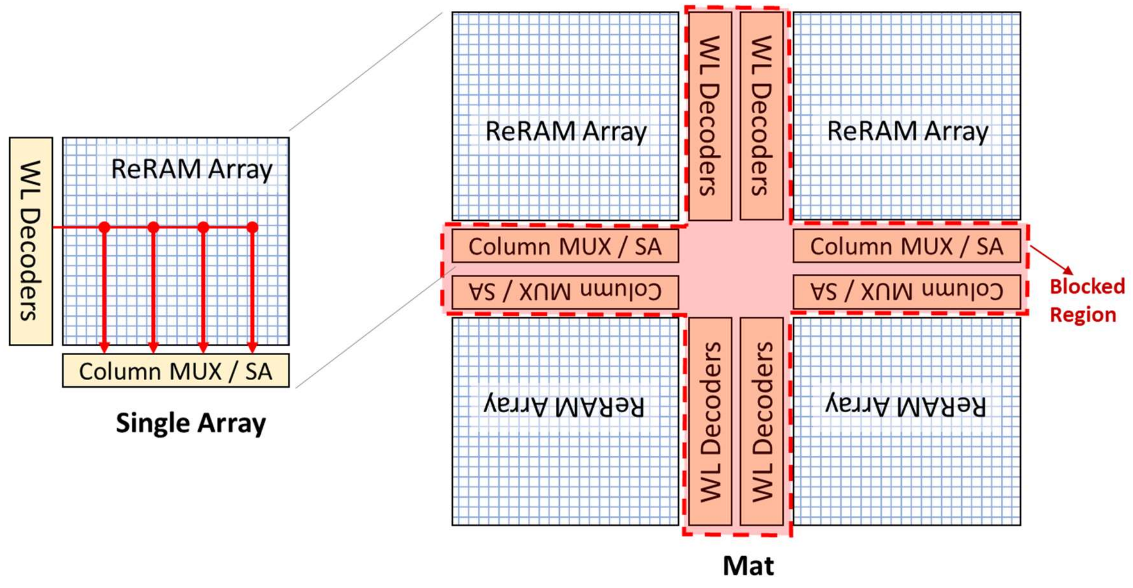


Figure 4-5 Memory Organization

The ReRAM peripheral circuits are the blocked regions during the APR digital implementation and form a “cross” shape of blocked region, and are indicated in Figure 4-5. Any CPU logic blocks need to either fit under one of the ReRAM arrays or need to have a method for connecting between two ReRAM array locations. While the blocked region specifies that no standard cells may be placed in that location, there can be limited restriction on the interconnect routing over these blocked regions. The specific metal layers that are blocked have significant impact on the routability of the overall integrated design. Completely blocking all metal routing over the blockage region necessitates any routing connections to go around the blocked regions results in significant additional routing area overhead with an integrated design.

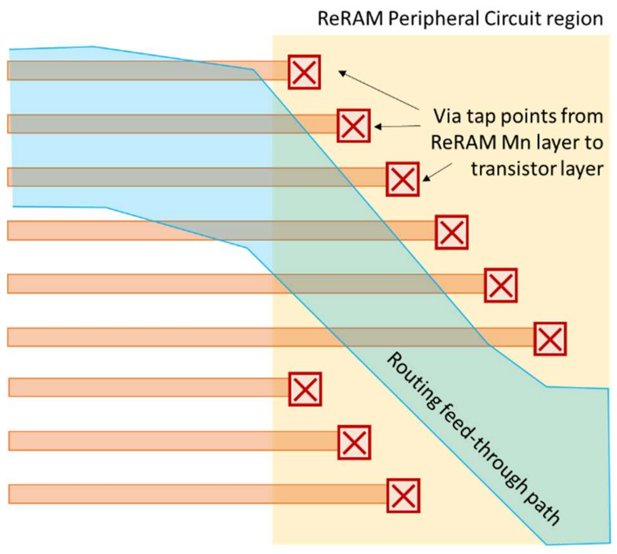

ReRAM Peripheral Circuit region
Via tap points from ReRAM Mn layer to transistor layer
Routing feed-through path

Figure 4-6 Via tap points from ReRAM metal layer to periphery circuits

Figure 4-6 describes the routing approach I assumed for my area study. The figure shows the close up of the physical interconnection between wordlines and the wordline decoder in the peripheral circuit region. The horizontal bars on the figure are wordlines coming from the ReRAM array to connect to an individual wordline driver, which is often the connected to a drain node on one or more transistors. Therefore, this connection needs to be able to route the high-metal line of the wordline (for example, from metal-layer 11) to the diffusion node of a transistor. This means that this connection has to go through multiple metal-via taps to descent to metal-1, and then connect to a diffusion contact. Having the CPU logic interconnection lines through this region poses a potential conflict with this transition. Therefore, I have identified a way in which an uninterrupted feed-through path could be allocated for the CPU logic interconnections. This feed-through path allows for global signals to route between standard-cell logic groups of the CPU logic circuit. The top-down view shows staggered via tap points that allows for a routing channel for signals to feedthrough across blocked region. This approach is scalable as the number of ReRAM stack increases. With higher stacking, there would be more via tap connections that would be needed. The blocked region could expand to accommodate a larger staggered connection from the higher memory metal layers to the base transistors below.

4.2 CAD Methodology

In this section, I go over the tool flow methodology I followed to perform my area assessment. Standard EDA tools are used for performing the area analysis of a co-located ReRAM with a processor, as shown in

Figure 4-7. The tool-flow starts with a design netlist to be synthesized. In the figure, this is indicated as RISC-V processor netlist in Verilog (.v) format, since my study involved a RISC-V processor. This behavioral Verilog netist is synthesized by Synopsys Design Compiler into physically realizable individual standard-cells selected from a design library. The process design kit (PDK) I used for my study is based on 45nm process node and makes use of design library from Nangate. The synthesized netlist (_syn.v) output from the synthesis step is input to Cadence Encounter is used for the APR step of the flow to produce the final GDSII layout. This is used in conjunction with specific limitations on the blockage to embed the ReRAM peripheral logic within the processor layout.

I used the design collateral files from North Carolina State University's (NCSU) 45nm process design kit (PDK). I also needed standard cell design libraries at this technology node. I initially looked at using one provided by Oklahoma State University (OSU). I chose the open-source Berkeley RISC-V VSCALE processor as the core for studying the processor-memory area impacts. The synthesizable Verilog netlist of the core is called VSCALE and uses a 32-bit instruction set with a single-issue in-order 3-stage architecture. The resulting layout was 59,672 sq um and operated at a maximum frequency of 150MHz.

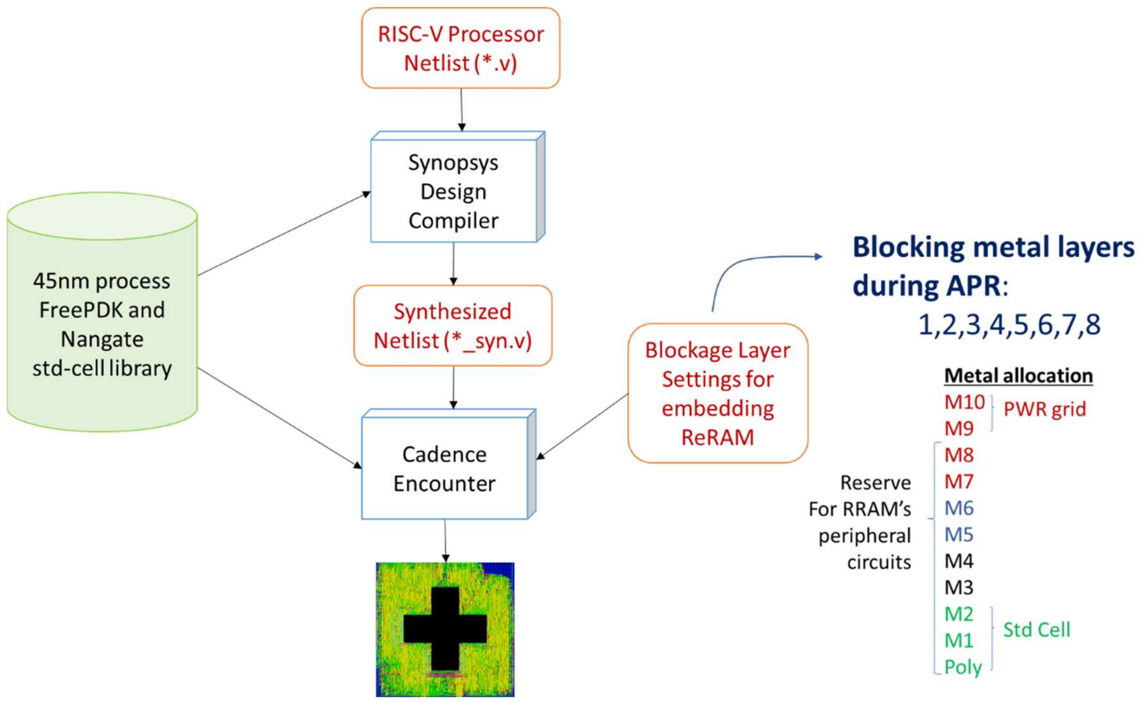

RISC-V Processor Netlist (*.v)
Synopsys Design Compiler
45nm process FreePDK and Nangate std-cell library
Synthesized Netlist (*_syn.v)
Cadence Encounter
Blockage Layer Settings for embedding ReRAM
Blocking metal layers during APR:
1,2,3,4,5,6,7,8
Metal allocation
M10
M9
PWR grid
M8
M7
M6
M5
M4
M3
M2
M1
Poly
Reserve For RRAM's peripheral circuits
Std Cell

Figure 4-7 Digital Implementation Tool Flow of an integrated ReRAM RISC-V Processor tile.

The OSU library for the 45nm process only provided 32 standard-cells, which may not provide sufficient diversity for optimum choice of standard-cells in terms of area and performance. This could cause the digital implementation to be overly pessimistic in terms of area and power, and not be representative of real PDKs available when manufacturing. As a result, I explored utilizing an alternative open-source PDK from Nangate based on the same 45nm PDK but containing a larger number (134) of standard cells. I repeated the Synthesis and APR step on the VSCALE processor to obtain an overall physical layout area of 30,373 sq um at 250MHz clock frequency, which was over 50% area reduction observed with this design kit. I attribute this area reduction to be due to sufficient diversity in the standard-cells available, which enabled the digital implementation to select an optimum standard-cell instance to minimize area and delay. I used this Nangate PDK to perform relative area comparison studies.

To mimic the integration constraints listed in the previous section, two types of blockage layers are indicated in the Cadence Encounter setting. The first is for the placement blockage to prevent standard cells from being placed, and the second is routing blockage for the specific metal layers to limit routing. Based on our discussion with Crossbar, prior ReRAM area measurements indicated that a 25% memory to periphery area ratio is a reasonable approximation for the two-layer memory stack. I used this guideline for allocation of the blockage area. For this second type of constraint, I mimic the restricted metal routing described in the previous section by blocking metal layers 1-8 and allowing for the APR tool to route through the blocked region using metal 9 and 10. The ReRAM memory layers are assumed to be in metal layers 11 and 12 above the standard CMOS layers. Rather than mimicking routing feedthrough channels, this allows for global interconnection signals that need to connect across the blocked region limited routing options. A summary of the blockage settings and metal allocation for my design is also provided in Figure 4-7.

4.3 Single ReRAM Cluster Integration

My first objective was to integrate the VSCALE processor with a ReRAM memory to create a processor-memory tile that could be laid out in an array, based on application needs. To begin with, I measure the stand-alone area of the VSCALE processor core alone. The synthesized netlist targeted an operating frequency of 150MHz with a total of 59,672 standard cells at the 45nm process technology (nominal process, 1v, 27c). I used Cadence Encounter to perform the APR and generate the layout for the core alone. My approach measures minimum feasible area by iteratively reducing the floorplan dimension and checking for congestion, Design Rule Check (DRC), and connectivity violations. If the floorplan area provided to perform the APR step is too small, then the tool will not be able to place all the standard-cells, make necessary connections, and meet the timing constraints imposed for the design. DRC is a check that ensures that the physical layers are drawn to meet the lithography rules of the process. Figure 4-8 shows the generated layout of the standalone core with power-rings around the core and a power-strap in the center.

The VSCALE core with the 45nm PDK, the core area consumed 30,373 sq. um. The dimensions of the floorplan are 172um x 172um. The generated layout includes the necessary standard-cells to implement the function described, as well as the interconnections in metal to make the connections. This PDK allows for 10 metal layers and the standard-cells are covered almost entirely by the metal signal lines. The APR tool typically uses even-odd metal routing, meaning that even metal layers are used for one direction, for example vertical, and odd metal-layers are used for horizontal location. This allows for efficient packing of a high number of metal interconnects. The floorplan also includes the power-rings in metal 9 and metal-10, and a metal-strap in the center of the core to allow for sufficient power supply bias.

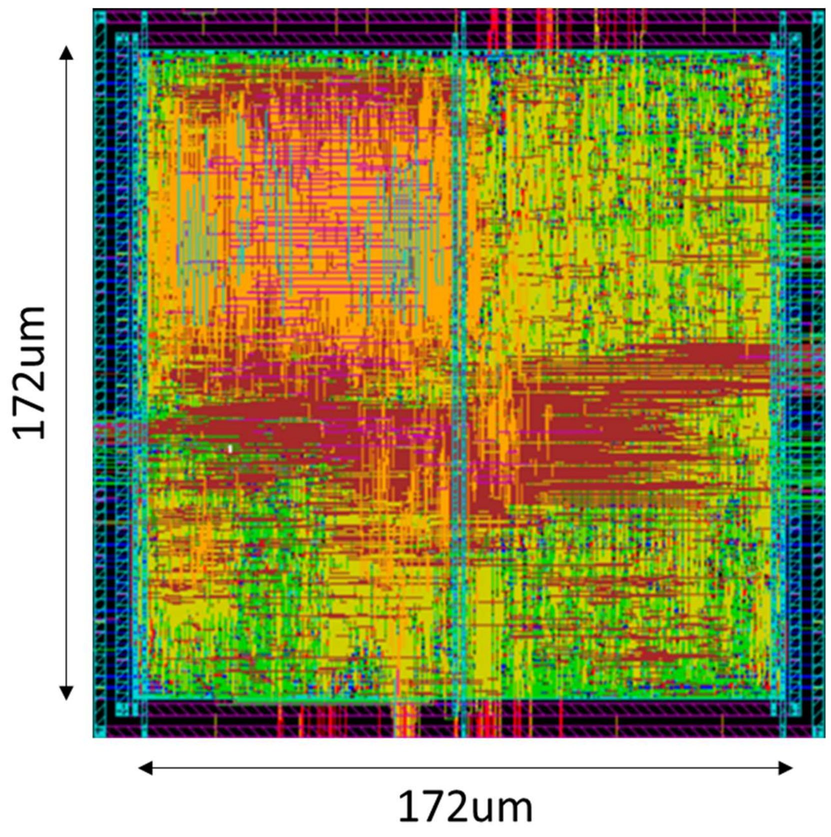


Figure 4-8 Layout of VSCALE Processor Core

Next, I talk about how embedding a ReRAM memory within the standalone core could be accomplished. As I indicated in Figure 4-5, a single array will require an L-shaped peripheral region surrounding it and is expected to have a relatively low bandwidth of 4-8 bits per array. In order to deliver reasonable bandwidth, I expect these arrays to be grouped together, in a mat, to form banks of arrays to meet the data bandwidth requirement in parallel. Physically, I chose these to be placed back-to-back in order to form one contiguous blockage region for higher area utilization.

I created a physical layout of the integrated ReRAM peripheral circuit with the VSCALE core using the above physical constraints as inputs to the Cadence Encounter tool. For this experiment, I used 4 ReRAM arrays, each of size 75um x 75um, which corresponds to a memory capacity of about 0.5MB for a 2-layer ReRAM stack. Note that crossbar has demonstrated feasibility of scaling to 8-layers for the ReRAM stack. Since the peripheral region takes 25% of the ReRAM area, this amounted to a total blocked region of 5600 sq. um for this configuration. The minimum feasible area was measured by iteratively creating a floorplan of smaller dimensions until the design is successfully placed and routed without any DRC or connectivity violations.

Figure 4-9 shows the generated layout of the ReRAM peripheral circuits embedded into a single VSCALE core. This layout only shows the standard-cell and blockage region information. The center cross (in red) denotes the blockage region, we've described to the APR to keep out the standard cell placements. The rows of standard cells (in blue) surround the blockage region complete. As mentioned before, the blockage region is specified for four of the L-shaped peripheral circuits arranged in a Cross configuration for my physical design study. This configuration has the advantage of allowing for I/O connectivity between the ReRAM memory and the processor, as well as allowing for connection between the overall tile which would need to communicate with other blocks.

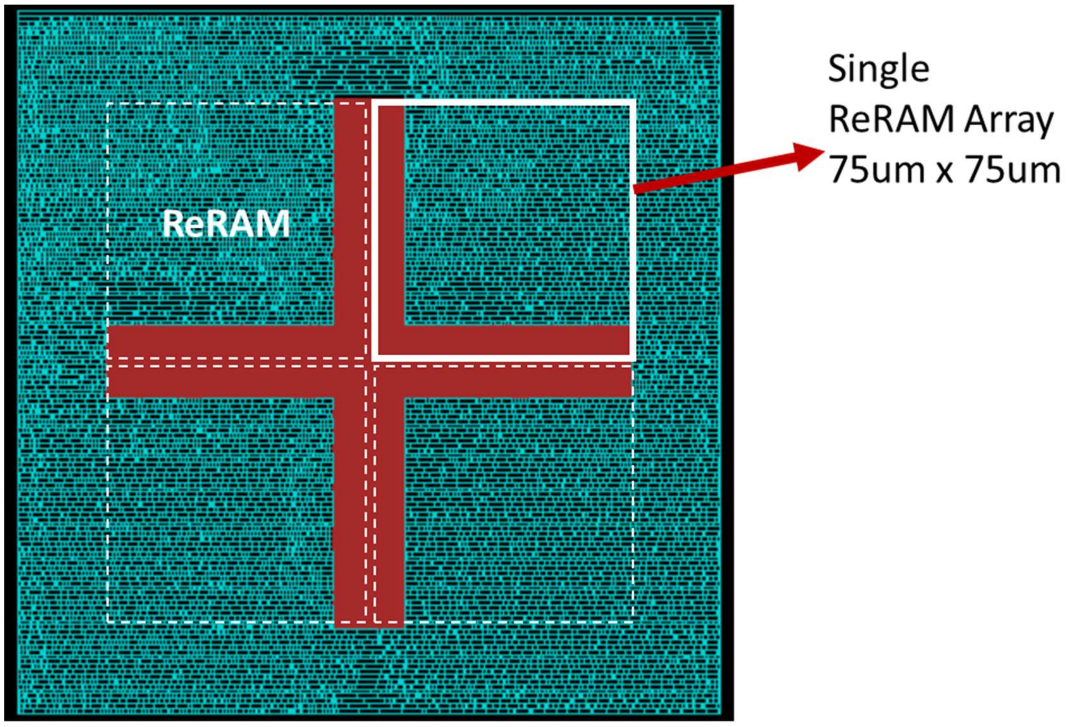


Figure 4-9 Blockage Region for ReRAM Peripheral Circuits

Figure 4-10 shows the final integrated ReRAM-Processor layout with all of the metal layers up to metal-8, excluding metal-9 and metal-10. Each of the red-square represents a single ReRAM array. The standard-cells and metal lines surround the L-shaped peripheral region, which is on the corner of each of the ReRAM arrays. The ReRAM arrays themselves will use higher metal layers, above metal-10 in this process node. This generated layout required the minimum floorplan area to meet the design and performance constraints without violating the DRC and connectivity rules. The dimensions of the layout are 200um x 200um, with the individual ReRAM arrays being of size 75um x 75um.

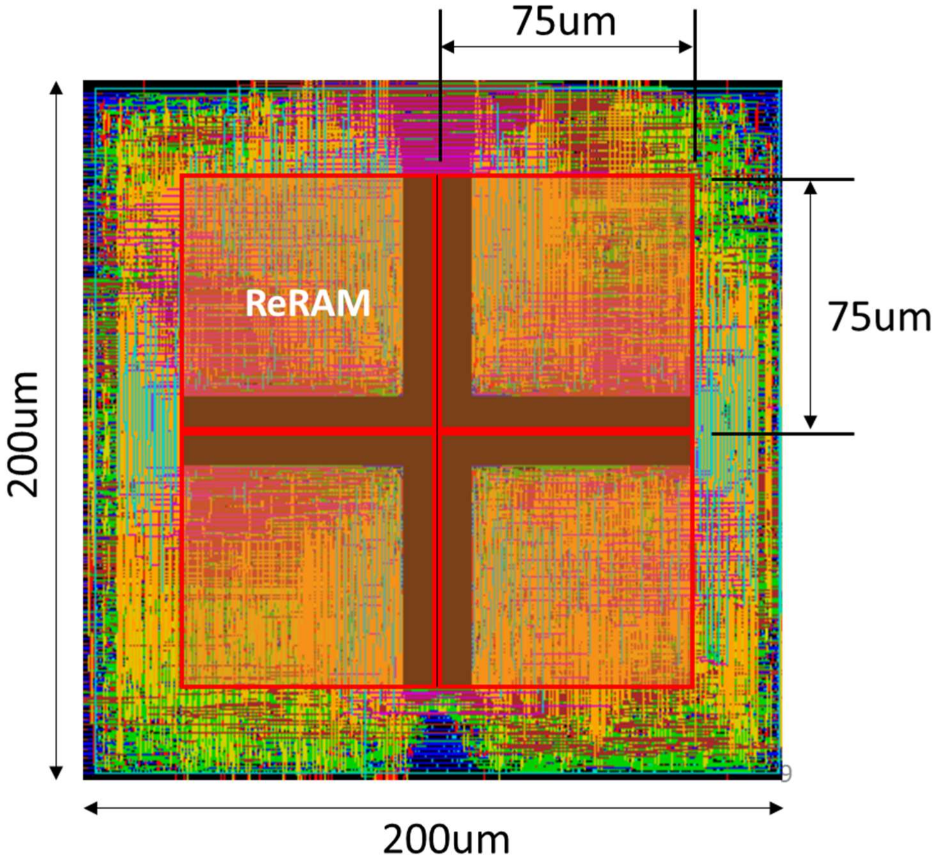


Figure 4-10 Layout of an integrated ReRAM RISC-V Processor tile.

Table 4-2 summarizes the measured area and the impact penalty of integrating a single ReRAM cluster with a RISC-V processor. The total area of this integrated design was 40,026 sq. um. Each ReRAM array's area was 75um x 75um, with the total ReRAM dimension being 150um x 150um. This allows for a total data storage for all four ReRAM arrays of 244kB, assuming a $5.6\lambda^2$ ReRAM cell per layer. For a 2-level stack, this translates to 488kB of total storage. The total blocked region blocked for the ReRAM's peripheral circuit was 5600 sq. um, which is 25% of the total ReRAM area of 22,500 sq um, in line with the expected overhead for a 2-layer ReRAM stack.

	NanGate 45nm Digital Library based on NCSU FreePDK45 design kit
Standard-cell area	22,088 sq μm
Target Frequency	250 MHz
Minimum Core Area for Processor Alone	172um x 172um = 30,373 sq μm Core Density = 98.9% Total Wire Length = 358mm
Minimum Area for Integrated Version with Blocked area: 5,600 sq μm	200um x 200um = 40,026 sq μm Core Density = 84.8% Total Wire Length = 278mm
Additional Area Impact due to Integration	48391/(30373+5600) = 1.1287 → 11.3% penalty

Table 4-2 Integration Results

After accounting for the peripheral blockage area and the actual standard-cell logic area of the processor, the total integrated layout incurs an additional overhead of ~11.3% in the 45nm process. The area penalty from the integration is measured as the difference between the total area of the integrated design and the sum of the VSCALE processor area and the ReRAM blocked region. This area penalty is mainly attributed to additional area needed for the routing of signals due to the blocked area in the center, around which there would be a higher incidence of routing congestion. There is also minor contribution due to additional filler cells incurred due to the larger overall area of the block. Filler cells are needed periodically to provide tap connections to the n-well and p-substrate from the power supply. This ensures that the body node of the transistors is well-biased. A larger area, therefore, requires more of these tap connections, increasing the overall area needed as well. This overhead is the area penalty due to additional area required for routing and standard-cell placement inefficiencies caused by noncontiguous regions available for the processor.

4.4 Multiple ReRAM Cluster Integration

Due to the small number of bits that each array outputs, about 4-8 bits/array, I expect many ReRAM arrays are tiled across the chip to form mats. These mats are accessed in a ganged mode to provide sufficient bandwidth. To study the area impact of such an approach, I studied the impact of multiple ReRAM arrays integrated into a single core, as illustrated in Figure 4-11.

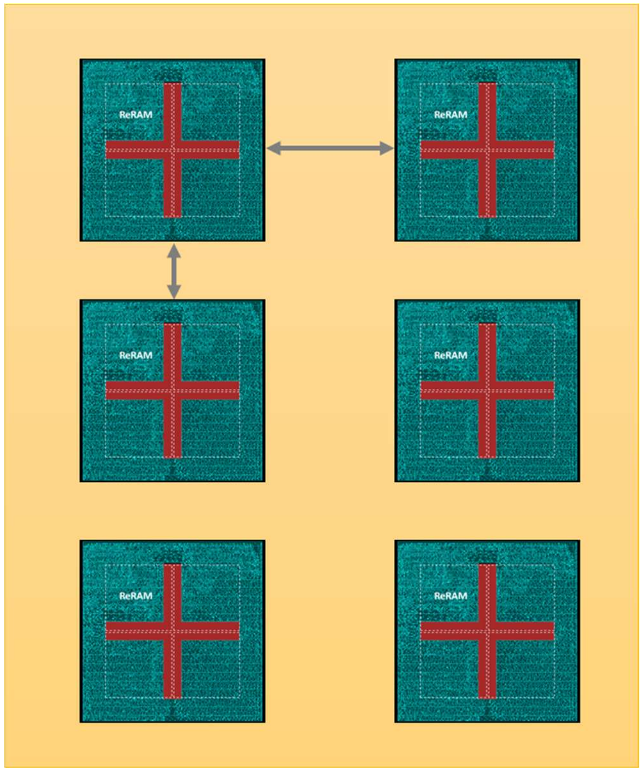


Figure 4-11 Embedding Multiple ReRAM Mat Clusters within a Larger Processor

The previous area study used a single ReRAM array to fit within the VSCALE core. VSCALE is a 32-bit integer core and is not representative of realistic cores which tend to be larger and more complex. To correspondingly increase the core size, I scaled the VSCALE processor's data path from 32-bit to 256-bit. Figure 4-12 below shows the scaled 256-bit VSCALE processor without any embedded ReRAM. The minimum generated layout had a floorplan dimension of 533um by 533um and an area of 284,077 sq. um at the 45nm technology node using the FreePDK based Nangate standard cell library. This larger core allows us to integrate multiple ReRAM mats into the VSCALE design.

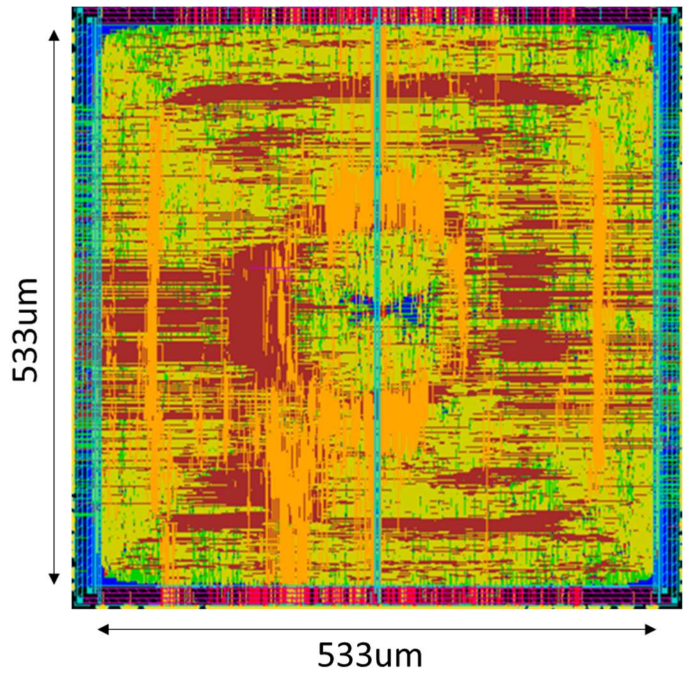
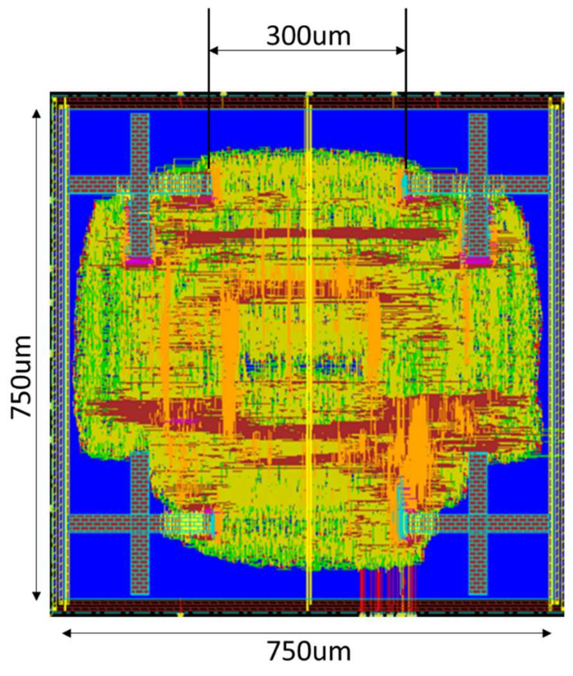


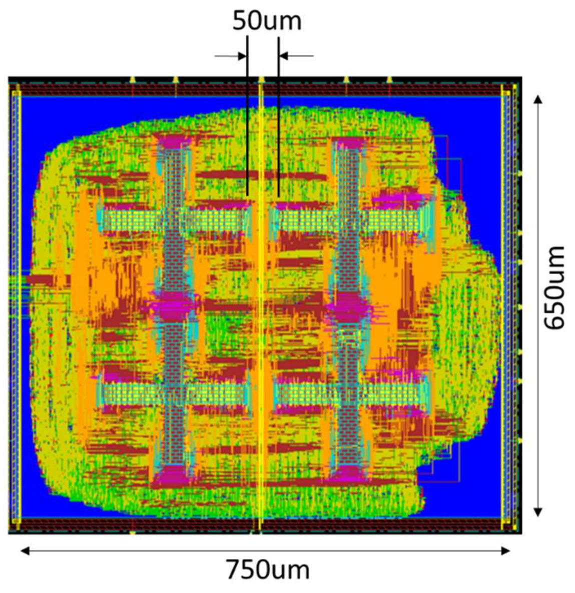
Figure 4-12 Scaled 256-bit VSCALE Processor Layout

Using the larger 256-bit VSCALE processor, I studied the impact of embedding four of the mat clusters within them in a 2x2 tile pattern. For the ReRAM array size, I aimed for an array of 1000 x 1000 matrix, and therefore used an array size of 109umx109um, making the mat size to be 218um x 218um. This size allowed us to tile the 2x2 mat within the 256-bit VSCALE processor for the purpose of my study. I iteratively varied the inter-tile cluster spacing to obtain the optimum spacing for minimum overall area for a range of tile spacings from 50um to 400um. The minimum area for each spacing parameter was found by iteratively reducing the floorplan dimension to check for feasibility.

Figure 4-13. below illustrates the floor planning result at the extremes of the inter-tile spacing when embedding ReRAM clusters within a larger circuit. If there is not sufficient spacing between the ReRAM peripheral circuit's blocked regions, then network congestion occurs when the processor blocks are being placed between them which cannot be resolved by the APR. On the other hand, if the spacing is too far apart, the entire generated layout can fit between the tiles resulting in large unused spaces. This can be seen Figure 4-13(a), which has an inter-crossbar spacing of 300um, and an overall chip dimension of 750um x 750um. Figure 4-13(b) shows the minimum area configuration for an inter-crossbar distance of 50um and a specified floorplan dimension of 750um x 650um (width x height). Note that it might be possible to optimize this layout manually and utilizing the areas in the corner to overcome this, however manual layout is beyond the scope of my initial area study.



(a) Spacing too large



(b) Mat spacing too close

Figure 4-13 Inter-Mat ReRAM Array Spacing causing Inefficient Layout

Figure 4-14 shows one of the generated layouts with four clusters of ReRAM arrays tiled and embedded within a 256-bit scaled VSCALE version. This layout shows the minimum area possible for an inter-mat spacing of 200um. There are four ReRAM mats, with each mat consisting of 4 arrays themselves. The total number of ReRAM arrays in this layout is 16, each of which follows the dimensions in the previous section. The design was obtained by iteratively reducing the overall floorplan size until the APR generated the layout successfully for this specific inter-tile spacing. The APR tool itself attempts 10 iterations by default to optimize the signal routing to meet the timing spec in the minimum possible area. Once the connectivity is verified, the DRC checks are performed to ensure that none of the physical design rules are violated.

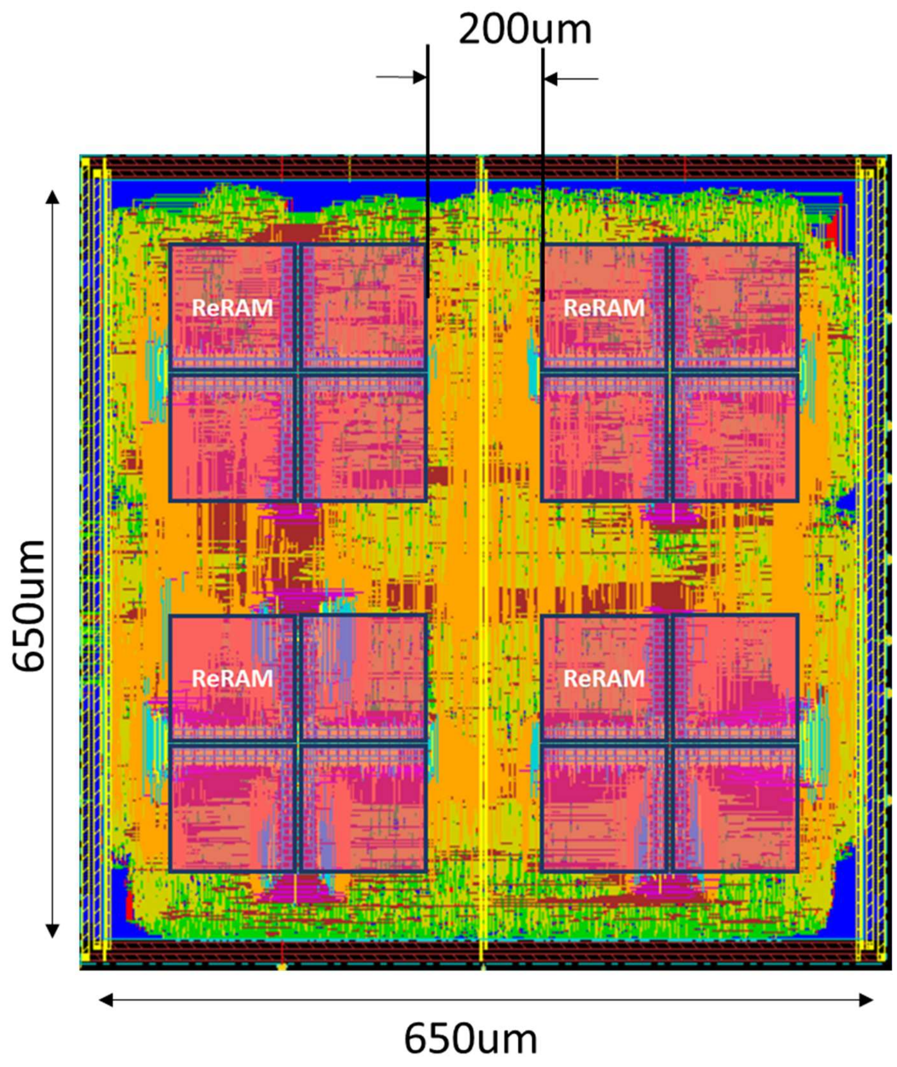


Figure 4-14 Multiple ReRAM clusters integrated with a 256-bit RISC-V Processor

The iterative process of finding the minimum area was repeated for a range of inter-mat spacing from 50um to 400um and the results are presented in Table 4-3. The minimum width represents the width of the minimum design, considering the width of the combined ReRAM MAT widths, and the width of the spacing. The ReRAM mat dimension is 218um x 218um, as I mentioned earlier. For example, at an inter-mat spacing of 25um, the minimum width would be 2*218um+25um=461um. The minimum area therefore would be square of 461, or 212,521 sq um. This type of floorplan would have no spacing on the outer edge of the array and therefore is not a feasible design. The chip area denotes the actual minimum floorplan area to realize the processor-ReRAM integrated design.

Inter tile spacing (um)	**25**	**50**	**100**	**150**	**200**	**300**	**400**
Min width (um)	461	486	536	586	636	736	836
min area (sq um)	212521	236196	287296	343396	404496	541696	698896
chip area (sq um)	562500	487500	390000	390000	422500	562500	722500
stdcells only (sq um)	223365	221981	221697	222471	221997	221465	221632
stdcell area (sq um)	516402	441178	343797	343797	376257	516256	677012
stdcell efficiency	39.71%	45.53%	56.85%	57.04%	52.54%	39.37%	30.68%
array area	190096	190096	190096	190096	190096	190096	190096
array efficiency	33.79%	38.99%	48.74%	48.74%	44.99%	33.79%	26.31%
% penalty	0.71	0.49	0.19	0.19	0.29	0.71	1.20

Table 4-3 Summary of Inter-Mat Spacing on Area and Efficiency

The stdcells-only row lists the raw standard-cells area reported from the synthesis tool, while the stdcell-area row lists the measured std-cell area from the APR tool, to include the additional area needed for routing. As can be seen on the results, this is often double of the raw standard-cell area. The stdcell efficiency reports the area occupied by the std-cell over the overall floorplan area and is intended to be a metric of how much usable space was devoted for the processor logic. At the optimum spacing of 100um or 150um, I see the standard cell efficiency being close to 60%.

The array area row denotes the total ReRAM array area that are used within the floorplan. Note that most of this array area makes use of higher-metal lines that don't coincide with the lower layers. As a result, there can be a high amount of overlap between the stdcell and the ReRAM array area. This is reflected in the results that show that at the optimum inter-mat spacing of 100um, the array efficiency is close to 50%. The final parameter, % penalty, denotes the additional area incurred from the integrated ReRAM-Processor system. For the four MAT, the total area of the peripheral region incurred 44,172 sq um. This is in-line with the 25% guideline that I followed for the array-to-peripheral area ratio. The table results indicate that the optimum configuration has a penalty of 19%.

The plots presented below in Figure 4-15 show the results of the optimal spacing and minimum area as a function of the inter-crossbar spacing. The x-axis lists the inter-tile spacing of the ReRAM mat blocks varying from 25um to 400um. Note that this spacing is uniformly applied between all of the MAT blocks. The y-axis in the first figure shows the area in sq mm.

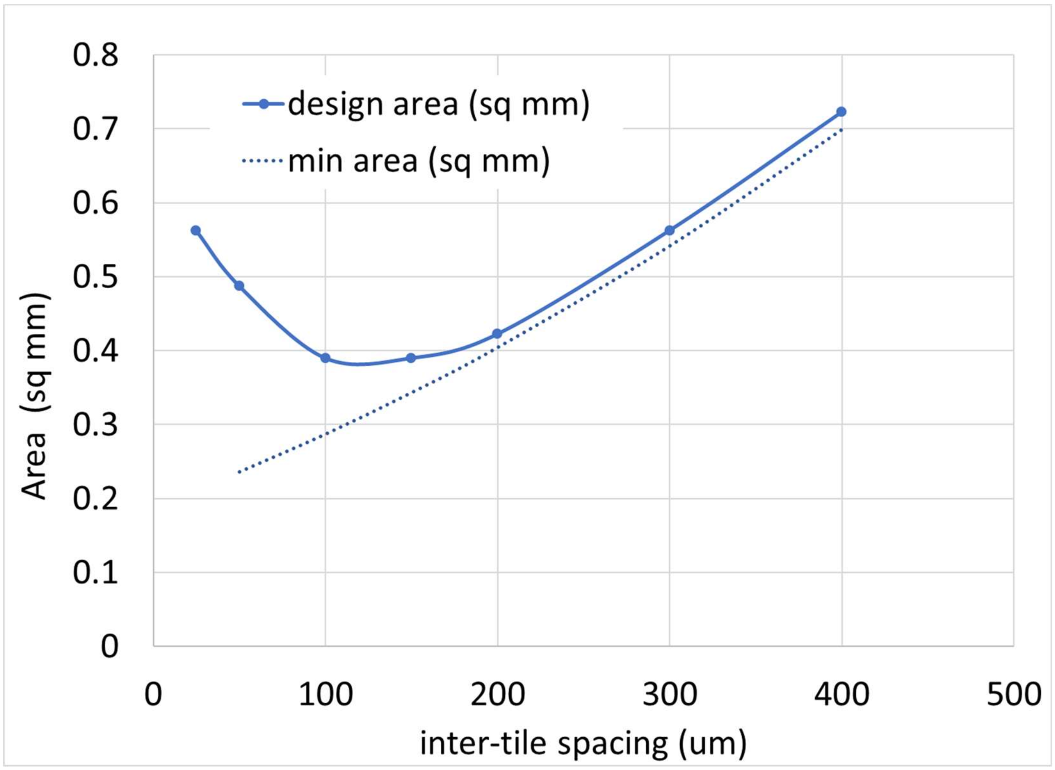


Figure 4-15 Impact of Inter-MAT ReRAM cross spacing on Area

The min-area, as described earlier, lists the theoretical limit on the minimum feasible floorplan, considering the spacing between the MAT blocks and the sizes of the MAT blocks themselves. The chip area line denotes the minimum successfully generated layout by the APR tool, given the timing constraints. I see that at large inter-tile spacing for the MAT blocks, the realized design is close to the theoretical limit. This is because the spacing between the MAT blocks is so large, that the entire design is able to fit within the inside of the MAT arrays, similar to the design indicated in Figure 3 12(a).

Figure 4-16 shows the impact of inter-mat spacing of the ReRAM blocks on the array and standard cell efficiency. The x-axis varies the inter-mat spacing and the y-axis reports the efficiency and penalty numbers as a percentage.

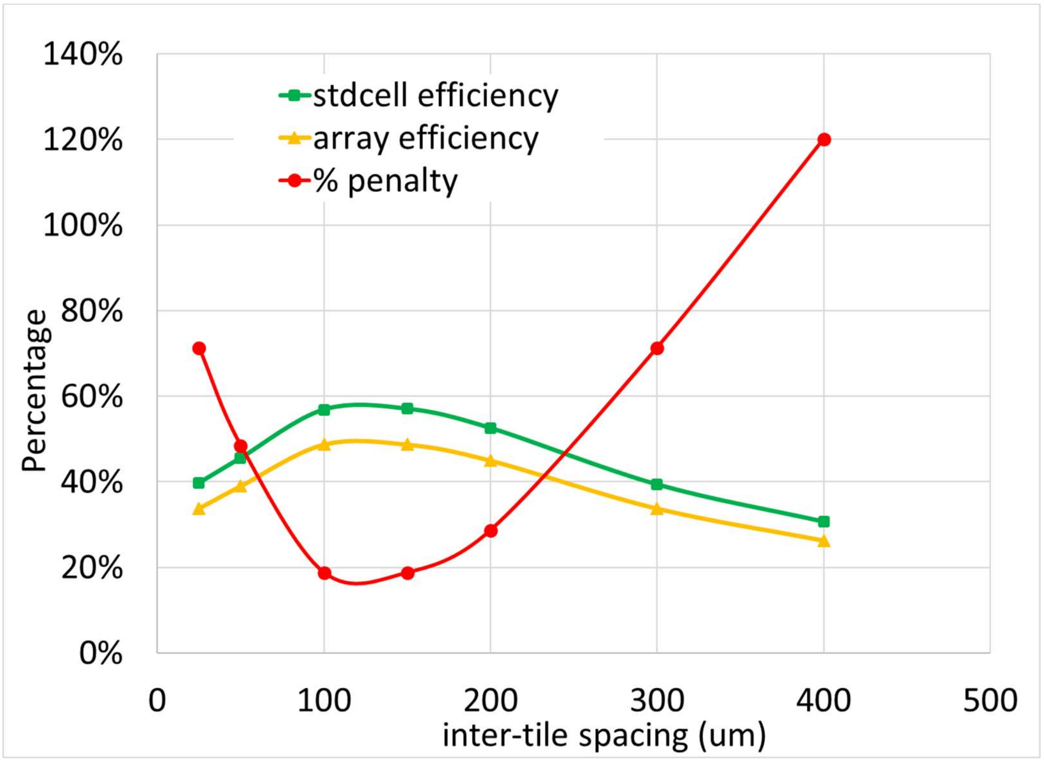


Figure 4-16 Impact of Inter-MAT ReRAM cross spacing on Efficiency

The standard-cell efficiency peaks at 57% at the optimum spacing of 150um. The ReRAM peripheral logic area occupies a total of 44,172 sq um, which accounts for 11.3% of the total design area of 390,000 sq um. The power rings surrounding the floorplan also consume some area, approximately 13%. The remaining area numbers could be accounted for filler cells, and unused standard-cells at the corner of the floorplans. Since the area study is done in increments of 50um, a finer step might indicate a lower feasible design than identified.

On the ReRAM array side, the efficiency peaks at 49%. The limitation on the ReRAM side preventing the array from completely covering the provided area is the peripheral circuits that align with each row and column. Completely covering the array would mean that these peripheral circuits extend all the way to the end of the floorplan, severely limiting the signal interconnections across the blocked regions. While the inter-tile spacing dictates the spacing between the mat blocks, the overall floorplan dimension dictates the spacing from the boundary of the design to the edge of the peripheral region.

The third curve in fig shows the area penalty of integrating the two design blocks. The area penalty is calculated by subtracting total design area from the individual processor and ReRAM peripheral block area. This penalty accounts for the cost of disrupting the processor floorplan area with a ReRAM peripheral block, largely due to additional routing for signals between standard cell groups, with possible routing around blockage regions.

To summarize, the results show that an optimum inter-ReRAM spacing exists to maximize area efficiency at close to 50%. At 45nm, with my design configuration, the optimum inter- spacing is 100um to 150um. Larger spacing (> 200um) leads to inefficiency from unused synthesized areas (empty space) while smaller spacing (<100um) leads to inefficiency from routing congestion between standard cell groups. The optimum spacing produced a peak array efficiency of 50%, with around 20% area overhead penalty for this configuration. For alternative configurations, the specific optimal point could be affected by the relative size of the processor and the blocked region due to the ReRAM array and would be worth investigating this relationship in a future study.

The 256-bit VSCALE extrapolation only scales the data path portion of the processor and will not model impacts of the control path of a more complex, realistic processor. However, I am only interested in the impact of routing congestion from a larger processor. For this purpose, extrapolating the data path is likely to have a higher impact on the generated layout rather than from a more complex control path. This is because I believe while complex control circuits might require more interconnects, these connections would be spatially local. On the other hand, data-path connections typically tend to span over longer distances to connect between subblocks. Therefore, I believe the area impact results would be a conservative indication of more realistic processor.

The total consumed area for the optimum layout was 0.4 sq. mm, with a total ReRAM data storage of 4MB for a 2-level stack, for all four clusters combined. The implementation shows a 2x2 array of ReRAM crosses integrated with a 256-bit integer RISC-V processor. Using a ReRAM array of size 109um x 109um, the total ReRAM data storage realized would be 4MB at 45nm process node.

Extrapolating these results to an 8-layer stack would create a 16MB ReRAM memory integrated into the ReRAM-CPU tile with an area of 0.4 mm^2. For a 400 mm^2 die size, the above ReRAM array could be tiled 1000 times across the chip, to produce a total storage capacity of 16GB ReRAM. Because an 8-layer stack would require additional peripheral circuits to decode the wordline per stack and/or higher current driving transistors, the number of cores will be scaled down. At the 16nm process, assuming a 10x reduction in area, a 400 mm^2 chip should be capable of delivering 160GB ReRAM storage with logic underneath assuming a 50% area efficiency.

Appendix A: Cadence Encounter Command File contains the final Cadence Encounter command file used to specify the blockage settings and perform the APR to generate the layout.

4.5 SRAM-ReRAM Integrations

One other configuration of interest is integrating an SRAM memory array underneath the ReRAM memory. The motivation for this is an SRAM array that would function as a write-back cache to an ReRAM main-memory so that the impact of ReRAM write latency, which is on the order of 1us, could be minimized. For my study, I have selected an open-source academic memory compiler, called OpenRAM [6], created by UC Santa Cruz and OSU. This tool includes SRAM leaf cells for the 45nm process using the same FreePDK45 design kit used by my standard-cell logic.

The SRAM bitcell used by the OpenRAM library at the 45nm node is shown in Figure 4-17 and compared with a 45nm ReRAM cell, which is close to 100 times smaller. The left side of the figure shows the ReRAM bitcell layout modeled as a cross-section of two metal layers, with a bitcell size of 5.6*Feature^2. At 45nm technology, this translates to 106nm x 106nm per bit. The right side of the figure shows the bitcell leaf-cell from the OpenRAM library provided by UC Santa Cruz and Oklahoma State University at 45nm. The bitcell dimensions are 0.707um x 1.344um and is composed of the conventional 6-transistor design. Since the academic version of the SRAM bitcell can be 2.5x larger than commercial version, I can expect the difference between the ReRAM and SRAM bitcells to be closer to 35x larger. Industry SRAM bitcells are optimized for the specific process they are to be fabricated in, and therefore have special

SRAM DRC rules that allow the pitch of the metal and base layers to be drawn closer than for regular logic, due to the regularity of the lithography pattern.

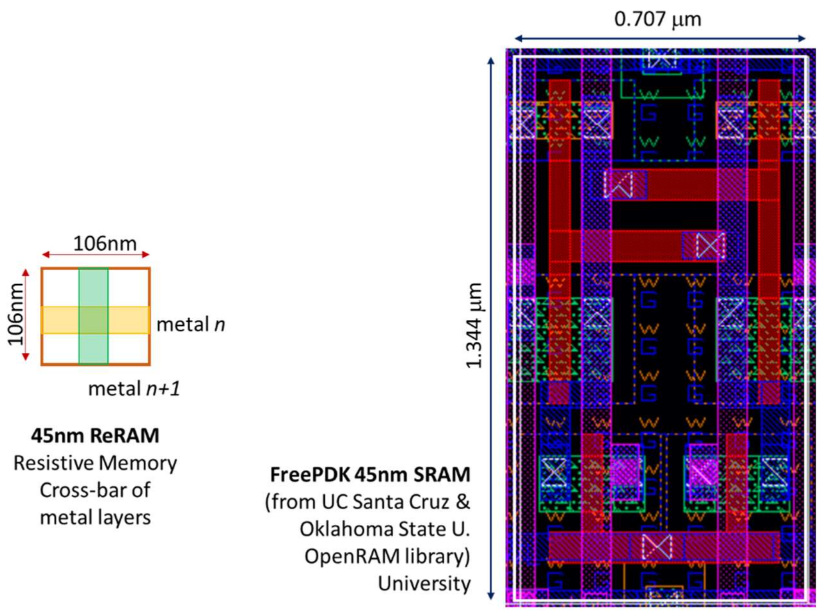
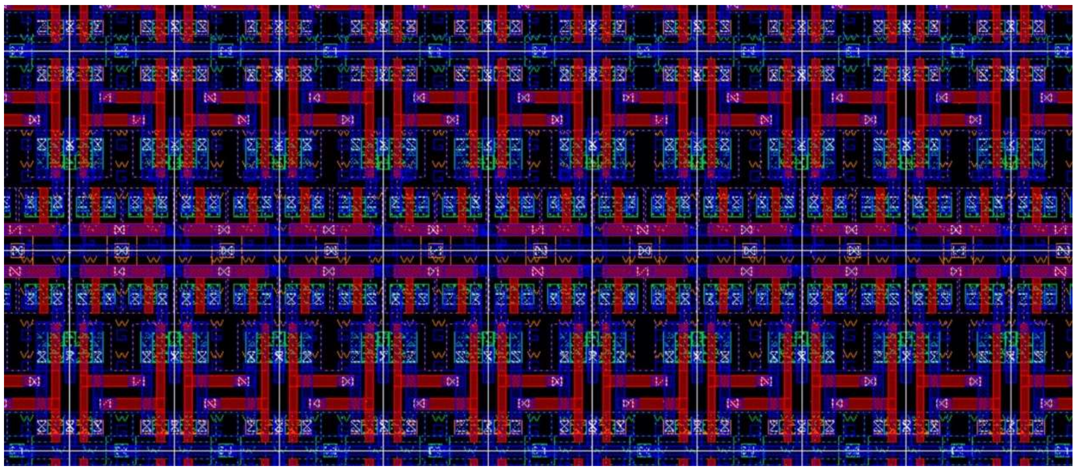


Figure 4-17 – Bitcell Relative Sizes at 45nm

Figure 4-18 shows the generated memory array bitcells (a) and the complete generated memory (b) in the 45nm technology. Figure 4-18 (a) shows the regular structure of two SRAM bitcell rows. The generated memory contains 128 rows and 256 columns and has a storage capacity of 4kB. The total area for the SRAM memory is 194.1um by 207.86um.

(a) 

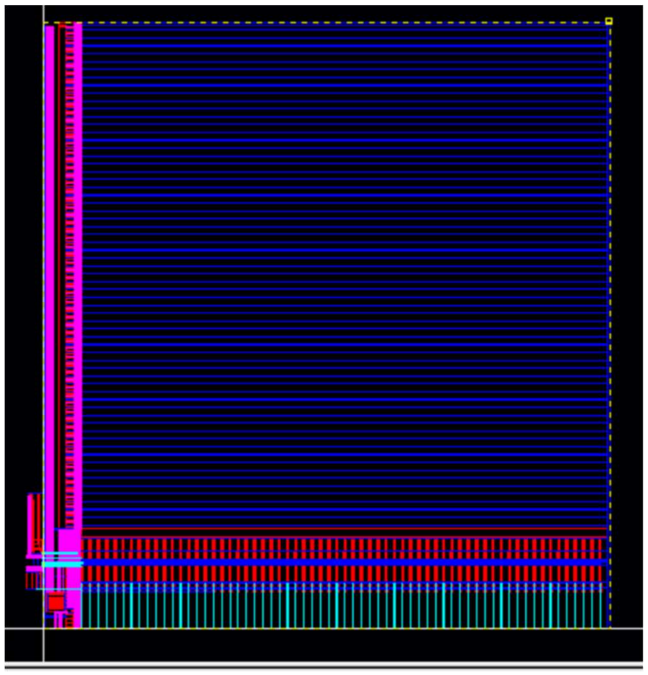
(b) 

Figure 4-18 - OpenRAM 45nm (a) Generated Bitcell Array (b) SRAM

Figure 4-19 shows four SRAM arrays placed together with four ReRAM array on top. The SRAM arrays are rotated to allow for the I/O ports of the SRAM to be accessed externally and not conflict with the central control region of the ReRAM array.

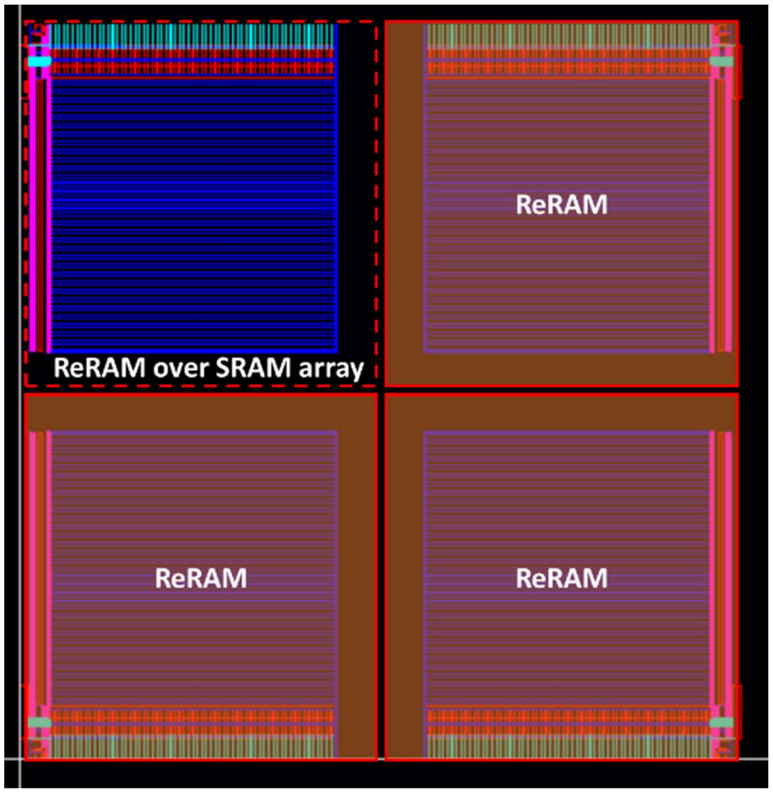


Figure 4-19 ReRAM Integrated with SRAM memory

The total SRAM capacity in this instance is 16kB (4kB each SRAM) with a total layout area of 211,725 sq. mm. I have drawn the SRAM arrays rotated to allow for their I/O ports on outside of tiles. The four ReRAM arrays each are drawn as a 115um x 115um array, with a total data storage of 1.1 MB for a 2-level ReRAM stack, with potential to scale to multiple layers based on fabrication capability.

Compared to the ReRAM-CPU layout, SRAM's array region would largely be limited to the lower metal layers (below metal-4). Therefore, ReRAM I/O connections can be made on the higher regions without difficulty. Also, because the four SRAM arrays are independent blocks, there is no need for the signal feedthroughs on the peripheral blockage regions, which makes this a more straightforward implementation.

4.6 Memory Architecture Calculator (MAC)

My next plan with regards to the physical design study was to use the area overhead numbers obtained to create a rough estimator on the die size, while being integrated with different processor types. Since I are considering a tile-based architecture, I looked at existing commercial and academic processors that have multiple-cores that could be adopted in such a way.

I considered four processor types for the study:

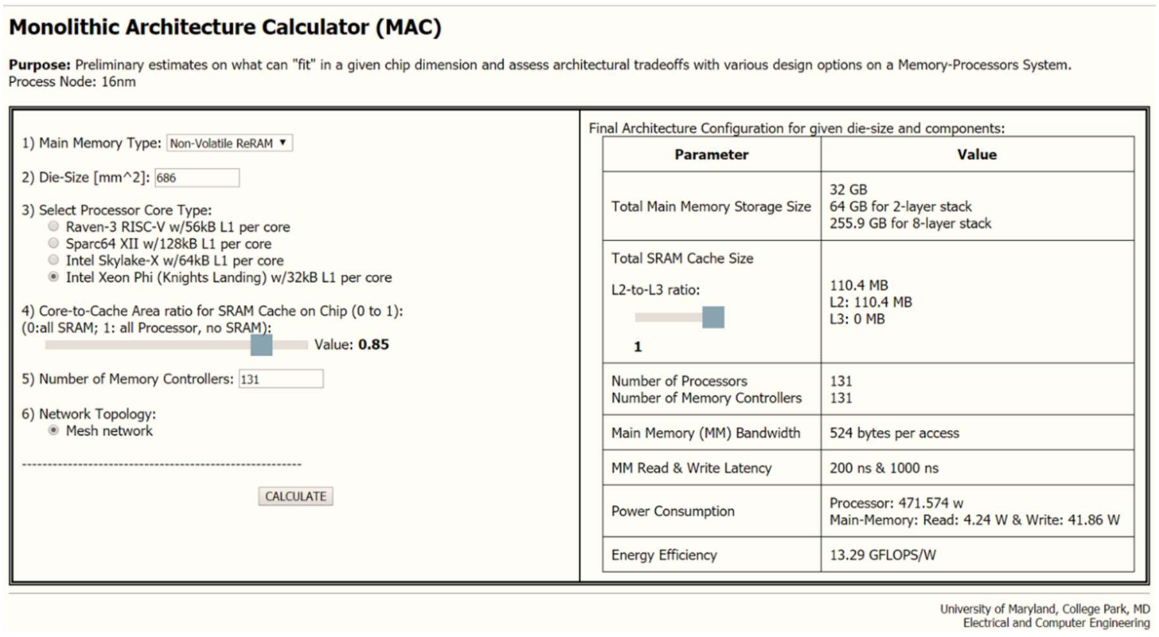
1. Raven-3 RISC-V processor with 56kB L1 cache per core
2. Fujitsu Sparc64 XII processor with 128kB L1 cache per core
3. Intel Skylake-X processor with 64kB L1 cache per core
4. Intel Xeon Phi (Knights Landing) with 32kB L1 cache per core

My target process node for my in-house calculator was 16nm. I extrapolated the area per core based on die-size measurements to estimate the per-core area for each of the processors at 16nm. They are listed in Table 4-4. Each of the different provide different functionality targeting their specific application, and consequently the area per core varies based on the complexity. This is reflected in the peak performance results listed for each processor.

Processor	Area per core [mm²]	Avg Power per core [W]	Peak Performance [GFLOPS]	Power Efficiency [GFLOPS/W]
RISC-V	0.55	0.17	6	34
Sparc64	5.02	24.5	448	1.14
Intel Skylake	16.9	9.17	1152	6.98
Intel Xeon Phi	3.13	3.61	3456	13.29

Table 4-4 Area, Power, and Performance comparison of Processors

I created a web based Monolithic Architecture Calculator (MAC) using JavaScript to provide rough estimates on what can "fit" in each chip dimension. Figure 4-20 has a screen-capture of the MAC interface.



Monolithic Architecture Calculator (MAC)

Purpose: Preliminary estimates on what can "fit" in a given chip dimension and assess architectural tradeoffs with various design options on a Memory-Processors System.
Process Node: 16nm

1) Main Memory Type: Non-Volatile ReRAM

2) Die-Size [mm^2]: 686

3) Select Processor Core Type:
- Raven-3 RISC-V w/56kB L1 per core
- Sparc64 XII w/128kB L1 per core
- Intel Skylake-X w/64kB L1 per core
- [✓ Intel Xeon Phi (Knights Landing) w/32kB L1 per core]

4) Core-to-Cache Area ratio for SRAM Cache on Chip (0 to 1):
(0:all SRAM; 1: all Processor, no SRAM):
Value: **0.85**

5) Number of Memory Controllers: 131

6) Network Topology:
- [✓ Mesh network]

Final Architecture Configuration for given die-size and components:

Parameter	Value
Total Main Memory Storage Size	32 GB 64 GB for 2-layer stack 255.9 GB for 8-layer stack
Total SRAM Cache Size L2-to-L3 ratio: 1	110.4 MB L2: 110.4 MB L3: 0 MB
Number of Processors Number of Memory Controllers	131 131
Main Memory (MM) Bandwidth	524 bytes per access
MM Read & Write Latency	200 ns & 1000 ns
Power Consumption	Processor: 471.574 w Main-Memory: Read: 4.24 W & Write: 41.86 W
Energy Efficiency	13.29 GFLOPS/W

CALCULATE
University of Maryland, College Park, MD
Electrical and Computer Engineering

Figure 4-20 MAC JavaScript Architectural Area Estimator

This can be used to assess architectural tradeoffs with various design options on a Monolithic Memory-Processors System and have the specific instance count of the different processor and memory controller. Users can specify cache size and number of memory controllers on a 2D mesh NoC topology. The left-side of the frame is the user-input, and the right-side summarizes the resulting characteristic of the chip based on the parameters shown, when the user clicks on "CALCULATE". User selects the type of main-memory (ReRAM or DRAM), the die-size, the processor type. The user also can select the ratio of area allocated between core and cache. The default value shown of 0.85 specifies 85% allocated for the processor area with 15% reserved for the SRAM cache area. The user can also specify the number of memory controllers, which can be an iterative process based on the number of processors that can fit. The example shown has a core processor to memory controller ratio of 1:1.

Appendix B: MAC Javascript Source Code has the complete JavaScript source code for the MAC.

4.7 Alternative Floorplan arrangements (L, Crossbar, Fractal design)

In this chapter, I analyze 3D floor planning options on how to partition the different blocks and I/O placement to minimize routing congestion and performance.

The previous experiment showed that integrating with a cross like connection in the middle of a processor logic limits the overall array efficiency of the chip. Here I am trading off the ability to connect to several discrete ReRAM memories locally to processor tiles to provide high bandwidth. As an alternate, if memory capacity is of prime importance, there is a way to approach near 100% array efficiency by utilizing an L-shape for the overall memory.

The floorplan shown in Figure 4-21 shows a 3-instance grouping of VSCALE processors (Single issue 3-stage in-order 32-bit integer RISC-V processor) underneath a 1MB 2-layer stack ReRAM memory in the 45nm process, as the previous section. The APR layout area APR area without ReRAM came out to be 304um x 304um = 92,712 sq. um, while adding this L-shaped ReRAM floorplan increased the area to 320um x 320um = 102,400 sq. um. This shows a negligible area overhead penalty from incorporating ReRAM in this way: 102.4k/ (92.4k + 11k) = ~1, i.e., no increase in area. I attribute this to the fact that since the available area for performing the APR is contiguous, no additional routing area needed.

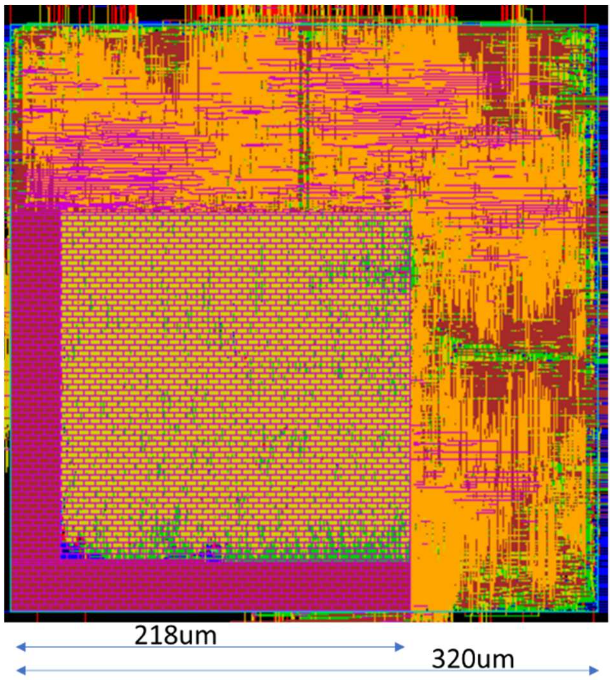


Figure 4-21 ReRAM with 3-core VSCALE processor

Depending on the array and processor size, each tile could be a self-contained core along with a memory, as shown in Figure 4-22. A small region between the tiles could be used for inter-tile routing channels and for network-on-chip (NoC) signals. However, there is a limitation in the ReRAM array size being too large, as this increases the read and write latency of the memory.

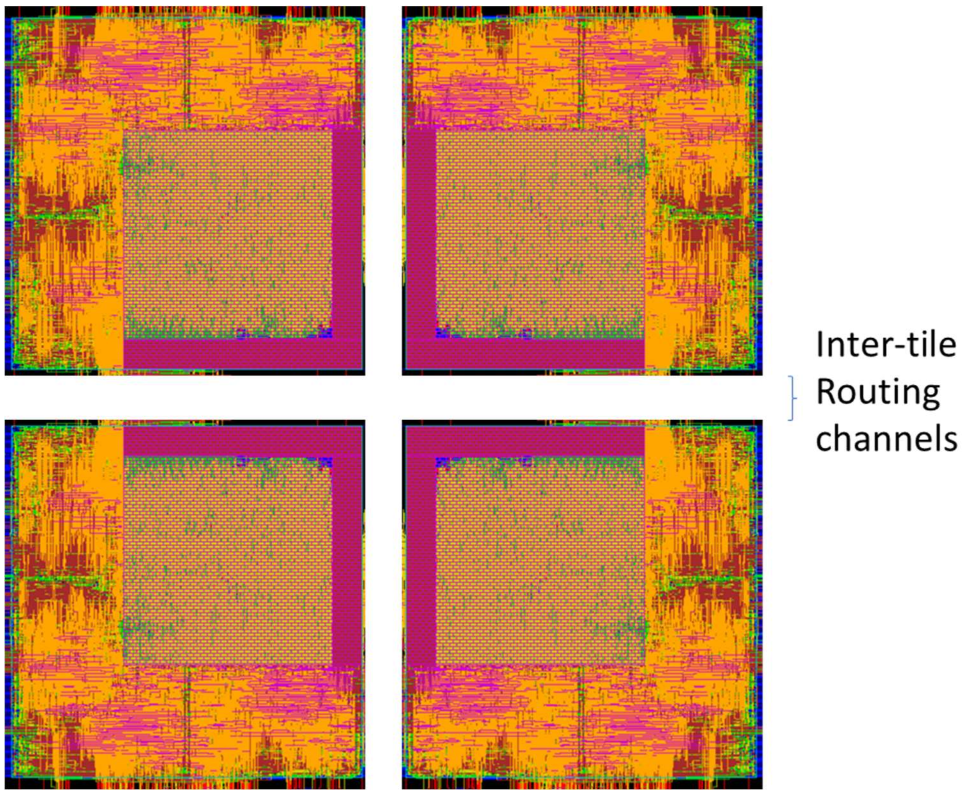


Figure 4-22 Independent Core with ReRAM block

Therefore, for designs that can tolerate a single interface point, it is possible to achieve a much higher array efficiency for the chip by placing the memory peripheral circuit alongside two edges of the chip. This ensures that maximum contiguous area is available for the APR tool.

With the motivation of having ReRAM integrate with a tiled processor, there are two floorplan options available based on the communication needs. In the case of a star network topology, the fractal design shown in Figure 4-23(a) allows for every ReRAM + Processor tile to be connected through the central node to any other tile. By not closing off the fourth tile, interconnection congestion would be prevented. This type of topology would typically be used in a server-client type of system with a need of central network connection.

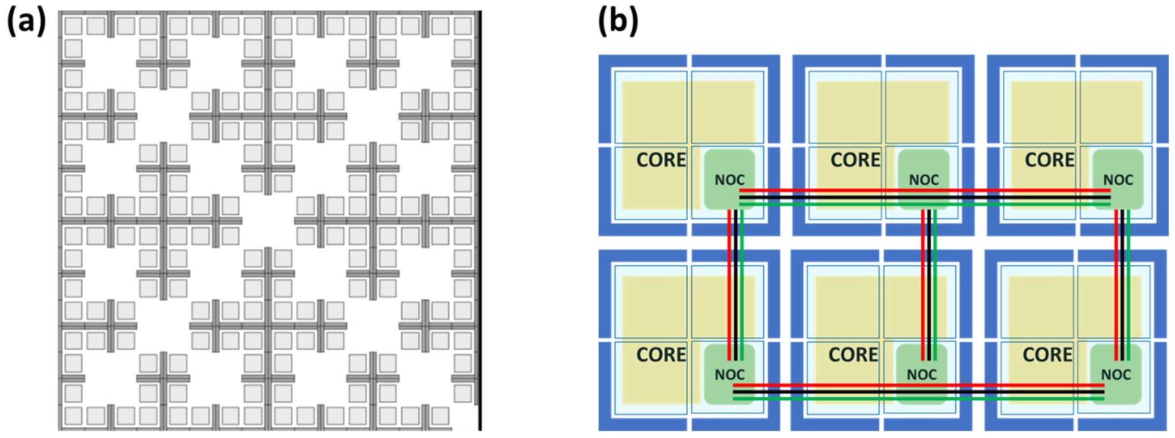


Figure 4-23 Alternative ReRAM-Processor integration floorplans showing (a) Fractal approach for Star topologies and (b) Mesh approach for mini-core parallel architectures

As an alternate, consider the massively parallel multi-processor approach where each individual tile consists of a modest processor coupled with local memory to provide higher power efficiency for certain tasks. These typically adopt a mesh-architecture topology where the interconnect communication is handled by a separate NoC (network-on-chip) control circuit. Figure 4-23(b) shows a possible approach of how this type of chip could be implemented with the ReRAM tiles.

4.8 Conclusion

Three observations are of note with the results obtained so far. First, I have shown that by making minor modifications to established standard tool flows, it is feasible to create a hybrid chip utilizing ReRAM, logic, and embedded SRAM blocks. Second, ReRAM density with respect to SRAM is quite favorable, especially using the 2-layer stack implementation. In the case of the core, floorplan results indicate that I can integrate the peripheral logic with minimal area penalty, while gaining the ability to create an integrated processor-memory system. Finally, I gave an overview of alternate floorplans arrangements that maybe suitable for specific applications that align with the memory access pattern.

5 ReRAM Device-Level Research Study

Based on my previous study, we believe that emerging memory technologies such as ReRAM that can be integrated onto standard CMOS processes have a significant advantage in replacing conventional DRAM as main-memory systems. These memory systems provide highly parallel, low granularity memory systems that support graph algorithms that are critical for machine learning and data science applications. In this section, I cover the research study that addresses the challenges at the device-level.

5.1 Motivation

ReRAM's high write energy and write latency requirements, along with lower write endurance, are key device-level challenges to be overcome when compared with existing DRAM solutions. The higher write energy requirement for ReRAM (when compared with DRAM) comes from the need to induce a physical change for storing Non-Volatile data. The data-retention time typically targeted for Non-Volatile ReRAM is typically around 10 years. However, ReRAM for Main-Memory applications do not necessarily require non-volatility of data. Current DRAM solutions store the memory for a few milliseconds before a refresh operation rewrites the data to preserve them indefinitely, as long as the power supply to the chip is supplied. I propose that if I reduce the data-retention requirement from 10 years to a much shorter time scale (for example: 100 seconds), it can be possible to use a lower write energy during the program operation. This lower data retention bitcell could be augmented with a periodic refresh so that the data would be rewritten.

Several prior work on ReRAM for neuromorphic applications, have demonstrated the switching between volatile and non-volatile states of these materials to mimic Spike-Timing Dependent Plasticity (STDP) [17-21]. For example, previous work by Shi, etc. [21] using Hexagonal Boron Nitride (h-BN) stacks-based ReRAM has shown switching behavior between volatile and non-volatile states. There was an observed “Self-Recovered region” which was an intermediate region between High and Low electrical stress which induced a time delay before “resetting” of carbon filaments once the stress was removed. I am not aware of anyone who is intentionally using the plasticity of ReRAM as a temporary memory storage device in order to exploit it for Main-Memory or DRAM replacement uses.

Based on reported characterization data for neuromorphic applications, there exists an intermediate region between high and low electrical stress where the memory retains the data for a much shorter time, but also has a lower electrical stress requirement. This translates to a lower electrical voltage or current applied to the cell, and/or for a shorter time. My approach is to use these materials in an intermediate region between volatile and non-volatile state where the data is retained for a much shorter time than is typically expected for non-volatile memory. In this intermediate region, based on the electrical stress applied, the conductive filaments remain for a shorter period, after which, the metal ions migrate back to the electrodes, relaxing the cell's state. Also, because this intermediate region requires less electrical stress than the non-volatile state, this translates to a lower write-latency, and/or lower program current/voltage to write to the cell. Additionally, this would also alleviate the requirement for a higher-voltage supply and corresponding charge pump circuitry to be included on the chip. This would make ReRAM-Processor integration more feasible for general applications, and not just read-heavy applications.

5.2 Fabrication Approach

Figure 5-1 shows an example of a Resistive Memory stack with two layers of metal-oxide region for the resistive-switching. This figure represents a cross-section of a ReRAM bitcell and exposes the material composition used to form the bitcell stack.

At the bottom of the stack, is a metal electrode formed with Platinum (Pt). The resistive switching element is composed of two materials – an Aluminum Oxide (Al_2O_3) and Titanium Oxide (TiO_2). Closing out at the top of the ReRAM stack are two metal electrodes – a Titanium (Ti) layer, and a Platinum (Pt) top electrode layer. This entire stack could be fabricated on top of substrate or on top of metal, depending on the process flow.

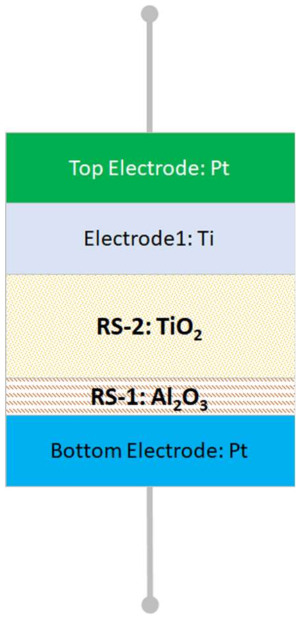


Figure 5-1 ReRAM Metal Stack

This ReRAM stack shown in the figure is one possible implementation of ReRAM that prior literature has shown to display the short-term plasticity. Using such a device, one possible scenario is that the data could be loaded into ReRAM from storage and allow for the computations to take place on the data for a set duration. After this set period, ReRAM data would be reloaded back from storage or refreshed from

ReRAM itself periodically, similar to what is done for DRAM memory. Alternate operation modes could also be introduced that allow for varying levels of persistence of memory depending on the level of write energy applied.

Oxide based ReRAM is attractive as the underlying metal insulator–metal structure is simple, compact and CMOS-compatible. Also, these materials have been observed to provide multi-level behavior and results in bipolar, asymmetric structure which follows the ionic migration model of the STDP behavior. Based on my literature survey, the following were identified as possible candidates for the ReRAM stack:

1. HfOx-based RRAM: TiN/HfO_2/Ti/TiN, TiN/HfO_2/Mg/W
2. Pt/Ta_2O_5-x/W
3. Ta/TaOx/TiO_2/Ti
4. Ti/AlOx/ TiN
5. Au/Ti/h-BN/Cu
6. Pt/Al_2O_3/TiO_2/Ti/Pt

All of the above have been observed to provide multi-level behavior and results in bipolar, asymmetric structure which follows the ionic migration model of the STDP behavior. Based on discussions with the UMD Nanofab lab, our universities' in-house fabrication and device testing facility, the final ReRAM stack combination **Pt/Al_2O_3/TiO_2/Ti/Pt** was feasible option and I chose to fabricate this stack, as shown in Figure 5-1.

The process outline for fabricating the ReRAM device for the Pt/Al_2O_3/TiO_2/Ti/Pt metal stack is as follows:

1. Start with 4” silicon wafers covered by 200nm of Thermal SiO2
2. Perform Standard Clean and Rinse
3. Form Bottom Electrode: Physical Vapor Deposition (PVD) of Platinum=Pt (60nm)
4. PVD of 5-nm Al2O3
5. PVD of 30-nm TiO2
6. PVD of 15nm Ti
7. Complete with Top Electrode: PVD of 60nm Pt

I used the following shadow-mask configuration as my initial fabrication, which allows us to create the masks manually, without requiring an external mask supplier. Figure 5-2 shows the initial ReRAM devices that I planned to fabricate. The devices will be on the range of 6mm for proof of concept. The figure on the left shows the top-down view of a 4” wafer with six devices of varying sizes, each having two probe points for the top and bottom electrode. Note that these devices are the resistive switching element alone and does not include the select device needed in an array to control unselected cells’ leakage current. The figure on the right shows a 3D view of the ReRAM stack and the connection to the bottom electrode plate in Platinum. The marked spots denote the location of the probe landings for my characterization measurements. The exact dimensions of the devices are provided in the next section which goes over the fabrication approach in detail. Based on my understanding of the resistive filament creation, I believe that the filament width will be localized and limited based on the current and electrical stress applied. Therefore, the shortest path through the oxide layers will limit the width of the filament.

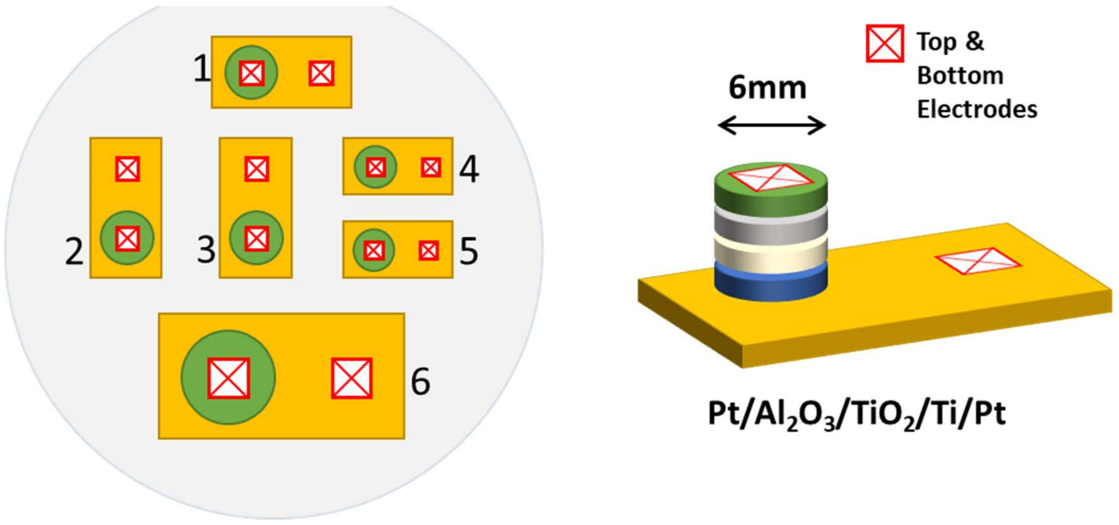


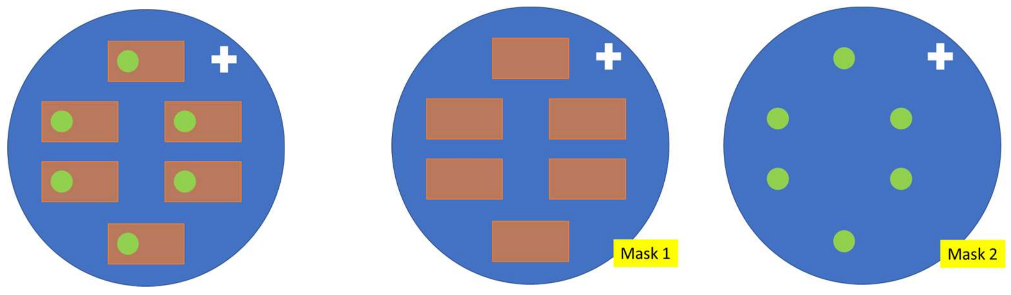
Figure 5-2 UMD ReRAM Device Fabrication

The configuration in Figure 5-2 shows six discrete ReRAM devices that will be fabricated. The top and bottom electrodes will be connected to test-probes to apply the stress and measure the resistance of the path. My characterization plan is to study the relationship between the resistance state of the device, data-retention time, and the electrical stress applied. The nature of the electrical stress is a combination of many parameters:

- Current Limitation (CL) – The program or write pulse can be controlled to not exceed above a set current-limitation point. This prevents the cell and write-path circuits from being exposed to excessively high amounts of current and being damaged.
- Pulse Height (voltage) – The program operation involves applying a voltage at a certain amplitude.
- Pulse Length (time) – The duration or the width of the write pulse applied.
- Pulse Period (time between pulses) – Certain write operations involve applying multiple write pulses in succession to move the placement of the resistance state. This induces the filament to be formed gradually and helps in avoiding over-setting the bitcell.

5.3 Mask Generation

For the test devices, I use two masks to create the pattern needed, as shown in Figure 5-3 below. The figure shows the top-down view of the masks used for the fabrication. The combined overlay of the two masks is shown in Figure 5-3 (a). One rectangular mask is used for the bottom electrode (Pt), which is Mask-1 in Figure 5-3 (b). Mask 2 in the figure is composed of a circular opening for the oxide layers and the top electrodes (Al2O3/TiO2/Ti/Pt). A small alignment mark is placed on the top-right corner to help with the positioning of the second mask.

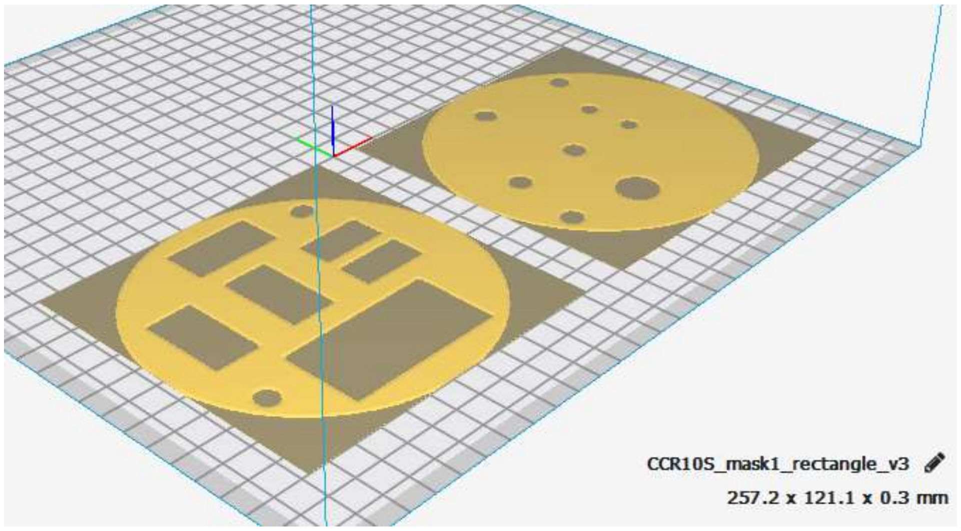


(a) Overlay of Masks **(b) Two masks used for device fabrication**

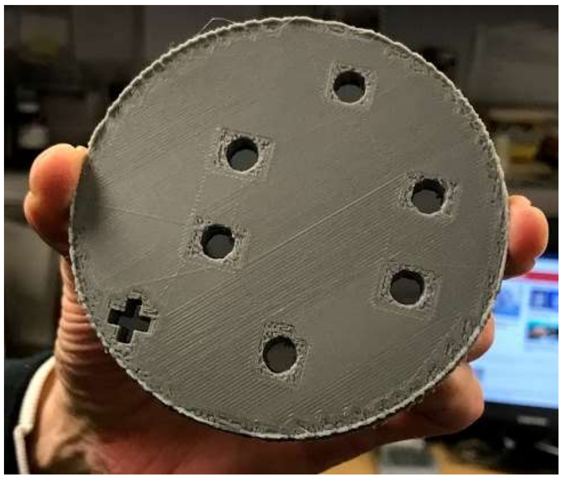
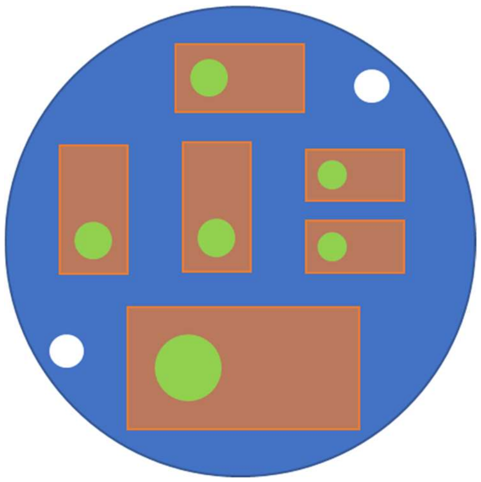
Figure 5-3 Mask Configuration

I ordered 4" (100mm) Si wafers with the <100> orientation from University Wafers to create my test ReRAM device structures on using Physical Vapor Deposition. I initially used a 3D-printer to print a polymer mask to check for alignment and confirm with UMD's fab-lab staff, shown below on the left. Ultimaker Cura software (v 3.6.0) was used to create the stereolithography file (.STL) CAD descriptions for the two masks. The masks were printed on a Creality 3D CR-10s printer using PLA, with a mask thickness of 0.5mm. Figure 5-4 shows the Cura generated mask file (a) and the prototype 3d-printed mask (b).

For the final mask, I decided to create the devices with different sizes to study the impact, and also changed the alignment marks to a circle (from a cross) to make it easier to create the mask. Figure 5-4 shows the final mask configuration I used to create the hard metal mask. Due to the higher temperature to which the mask would be exposed during the PVD process and the low resolution of the features (order of mm), I decided to use an Aluminum Shadow Mask for the features. I ordered 6061 Aluminum sheets (0.063” thick) from McMaster Carr.



(a) Cura 3D-Model of Mask Prototype

(b) 3D printed PLA mask prototype (c) Final Mask Configuration

Figure 5-4 Mask Prototype Creation

University of Maryland has an iReap Machine Lab with a ProtoTRAK SMX milling station which I used to cut the features and create the mask. The dimensions input for the first and second mask are given below. There are six devices of varying dimensions that were fabricated. The milling tool has the option to cut geometric shapes with specified location and dimensions.

For my mask generation, I used the circle and rectangle pattern to input the features to be drawn shown in Figure 5-5. For both masks the lower-left (LL) and the upper-right (UR) alignment marks were drawn as circles, which is an easier geometry to draw. The Xo and Yo indicates the origin of the feature, which is the center for a circle and the lower-left and upper-right coordinates for a rectangle. Mask 2 specifies the actual location and dimension of the ReRAM stack. The bitcell diameters used are two devices of 5.94mm, three devices of 7.56mm, and one device of 14.04mm.

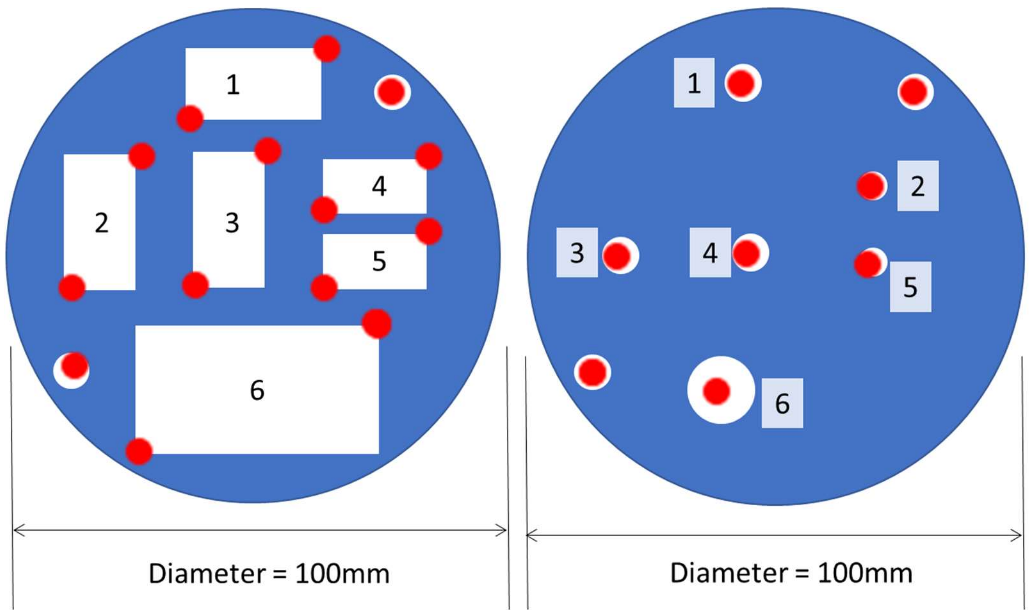


Figure 5-5 Final Mask Configurations for Mask 1 (left) and Mask 2 (right)

Table 5-1 lists the dimensions and coordinates used to specify the location of the features that were input into the milling tool for the two masks. With respect to the center point of the mask, the respective X and Y coordinates for the different features are explicitly specified. The dimensions for the circle that would be encompass the ReRAM stack are also specified in the diameter parameter.

Xo	Yo	Mask 1 Features
-38	21	LL alignment mark, Circle diameter = 7.56
26	-34	UR alignment mark, Circle diameter = 7.56
-14	-28	LL coord Rectangle 1
13	-42	UR coord Rectangle 1
-38	6	LL coord Rectangle 2
-24	-21	UR coord Rectangle 2
-13	5	LL coord Rectangle 3
1	-22	UR coord Rectangle 3
13	-10	LL coord Rectangle 4
34	-21	UR coord Rectangle 4
13	6	LL coord Rectangle 5
34	-6	UR coord Rectangle 5
-25	39	LL coord Rectangle 6
23	13	UR coord Rectangle 6

Xo	Yo	Mask 2 Features
-38	21	LL alignment mark, Circle diameter = 7.56
26	-34	UR alignment mark, Circle diameter = 7.56
-7	-36	Circle 1, diameter = 7.56
19	-15	Circle 2, diameter = 5.94
-32	-1	Circle 3, diameter = 7.56
-6	-2	Circle 4, diameter = 7.56
18	0	Circle 5, diameter = 5.94
-12	26	Circle 6, diameter = 14.04

Table 5-1 Mask Feature Specifications

The pictures below in Figure 5-6 (a) show the ProtoTrak SMX milling station which allows the features' coordinates and dimensions to be input. Figure 5-6 (b) shows the features being cut into the aluminum sheet. Figure 5-6 (c) shows the final two masks after the cut, with them overlaid on top of each other using the alignment marks in Figure 5-6 (d). The edges of the masks were deburred to smooth them out.

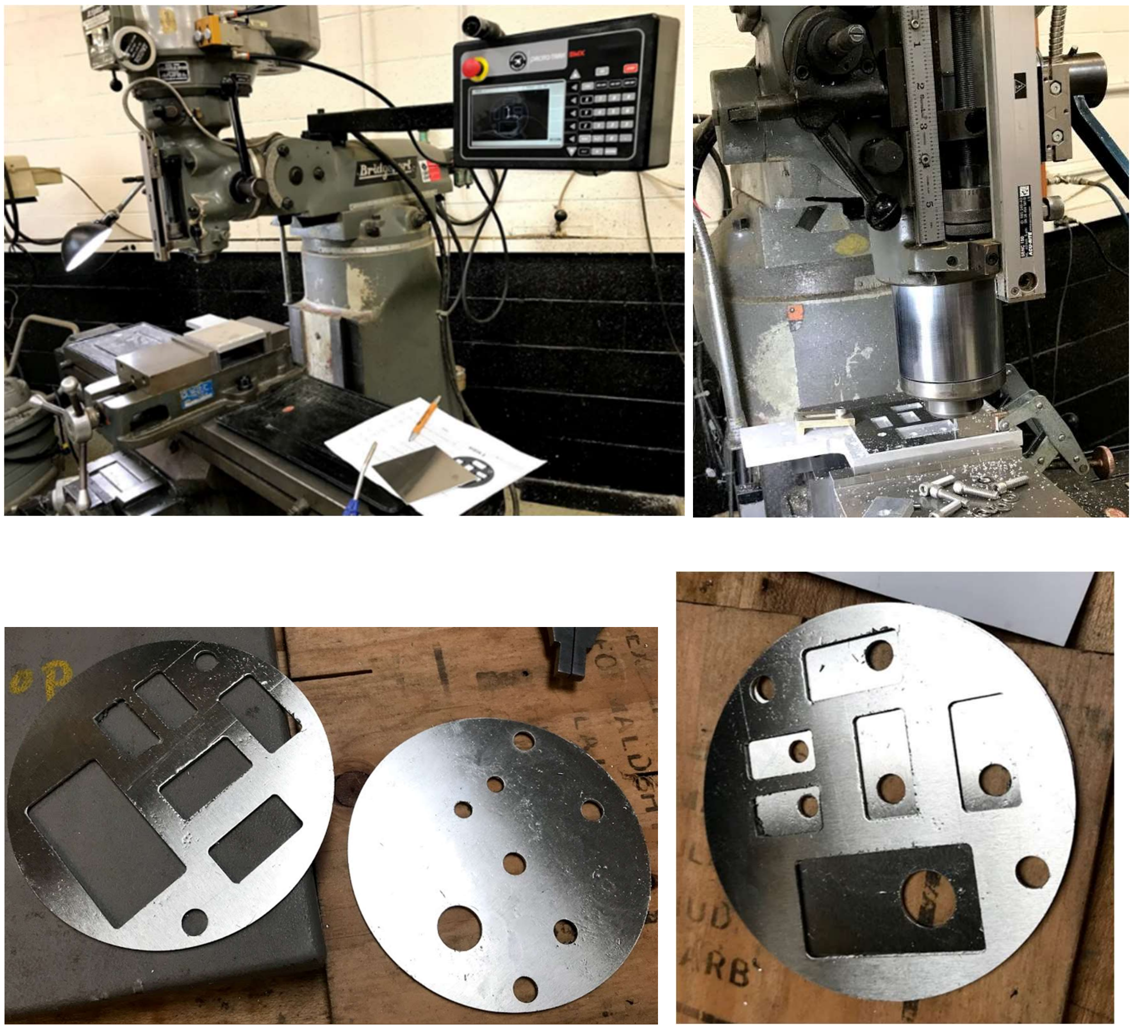


Figure 5-6 Mask Fabrication *clockwise from top:* (a) ProtoTRAK SMX Milling Station (b) Sheet Mask being cut (c) Finished mask set (d) Finished Mask Set overlaid

These two masks fabricated create the rectangular bottom electrode and the circular metal-oxide ReRAM stack along with the top-electrode. My process used Platinum and Titanium for the metal electrodes and Aluminum-Oxide and Titanium-Oxide for the metal-oxide stack. I used UMD's Physical-Vapor-Deposition chamber to sputter the materials to the areas, which is a feasible approach due to the larger dimensions of these devices.

5.4 Device Fabrication

I worked with UMD's nanofab lab to fabricate the Pt/Al_2O_3/TiO_2/Ti/Pt ReRAM devices on the 4" (100mm) wafer using the aluminum masks I had previously milled. As mentioned before, my motivation is to study the use of ReRAM devices in an intermediate region between volatile and non-volatile state where the data is retained for a much shorter time than is typically expected for non-volatile memory. In this intermediate region, based on the electrical stress applied, the conductive filaments remain for a shorter period, after which, the metal ions migrate back to the fill the oxygen vacancies in the filament, relaxing the cell's state. Also, because this intermediate region requires less electrical stress than the non-volatile state, this translates to a lower write-latency, and/or lower program current/voltage to write to the cell.

The process outline for my device fabrication is shown in Figure 5-7. Figure 5-7(a) shows the initial SiO_2 deposited onto the silicon substrate. I used 4" (100mm) Si wafers covered by 200nm of Thermal SiO_2. Fabrication of the devices was performed using Physical Vapor Deposition (PVD). First, the bottom electrode was formed of 60nm Platinum using the first mask (rectangular base), as shown in Figure 5-7(b). Then, the metal-oxide layers and the top-electrodes were deposited using the second mask (circular). The thickness used were 5nm Al_2O3, 30nm TiO2, 15nm Ti, and 60nm Pt for the top electrode, as shown in Figure 5-7(c).

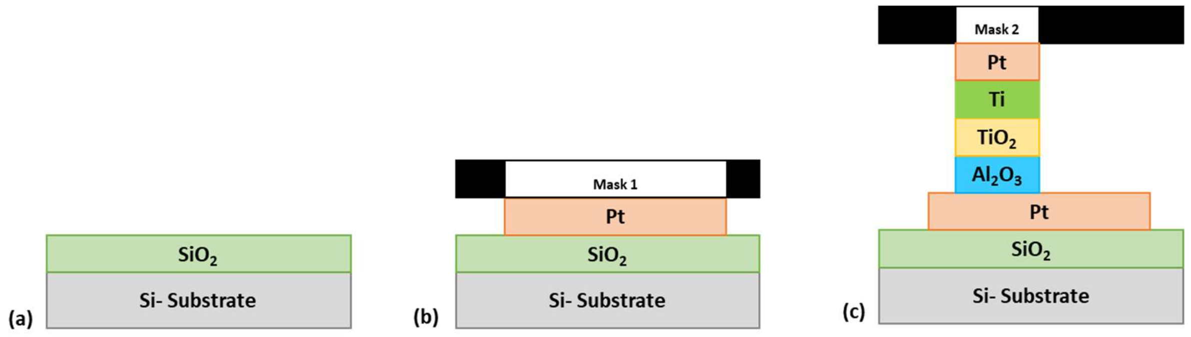
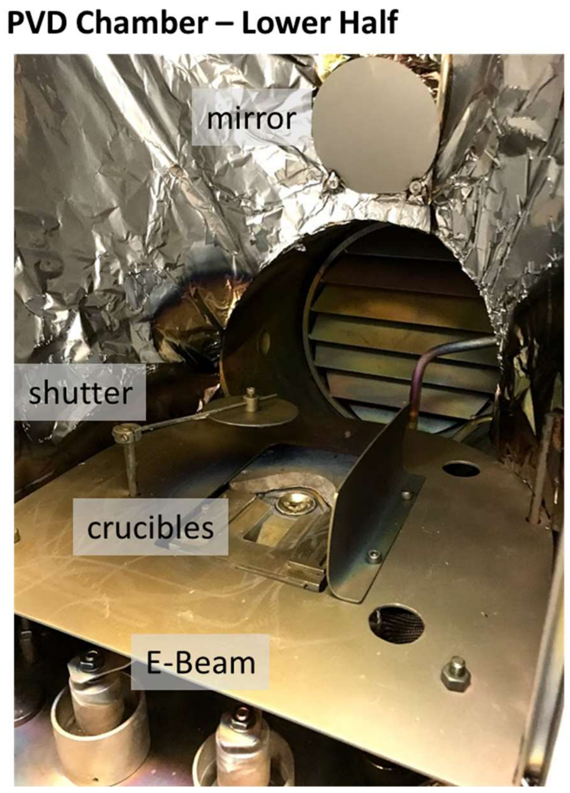


Figure 5-7 Fabrication flow for Pt/Al2O3/TiO2/Ti/Pt ReRAM structures (a) Thermal SiO2 (b) Mask 1: PVD of bottom electrode (c) Mask 2: PVD of ReRAM stack and top electrode

The material deposition was performed using the Denton Ebeam/thermal evaporator. Figure 5-8 shows the PVD chamber setup used for the fabrication. On the lower half of the chamber, shown in Figure 5-8(a), the E-Beam is generated and directed to the crucible. The material to be sputtered is placed in the crucible and magnets on the side of the chamber are used to direct the E-Beam towards the crucible. The chamber is brought to a low-pressure environment to accelerate the conditions for evaporation. A shutter resides over the crucible preventing any early evaporated material from reaching the wafer, which is mounted on the upper half of the chamber. A mirror mounted on the sidewall of the chamber allows for the material in the crucible to be observed through a window, to ensure that the material has evaporated. Figure 5-8(b) shows the upper half of the PVD chamber. The wafer along with a hard-mask is mounted onto the wafer-clamp facing the crucible (upside down). A sensor to the side of the wafer clamp is used to measure the amount of deposited material, which is used to calculate the thickness of the material deposited.



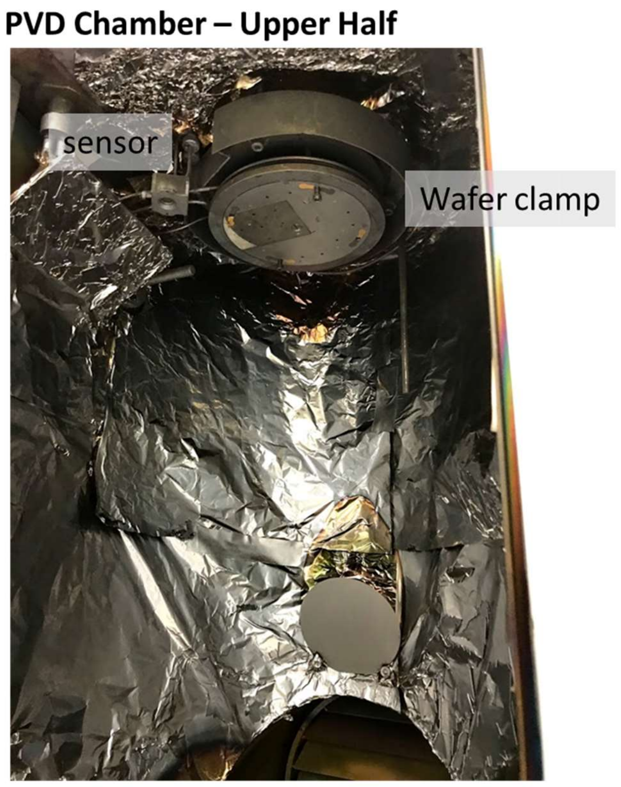


Figure 5-8 – (a) PVD chamber used for fabrication (b) Fabricated test wafer of discrete devices with probe measurements

For Platinum, the evaporation temperature is 1768 deg-C (3214 deg-F). My fabrication process began with loading the Denton E-Beam/Thermal evaporator with the materials in the crucibles (see Figure 5-9 below).

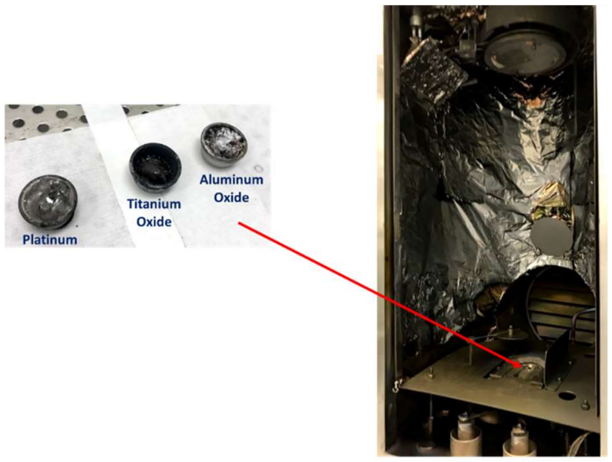


Figure 5-9 - Crucible materials into PVD Chamber

After cleaning the hard aluminum mask with Isopropyl Alcohol (IPA) to wipe down any debris, the mask was clamped onto a wafer and mounted to the chamber. Figure 5-10 shows the wafer clamped with the first mask and the platinum, which is the bottom electrode, being deposit onto the wafer. The two-circular alignment markers can be seen in the corners of the wafer. This is used as reference when clamping the second mask onto this wafer.

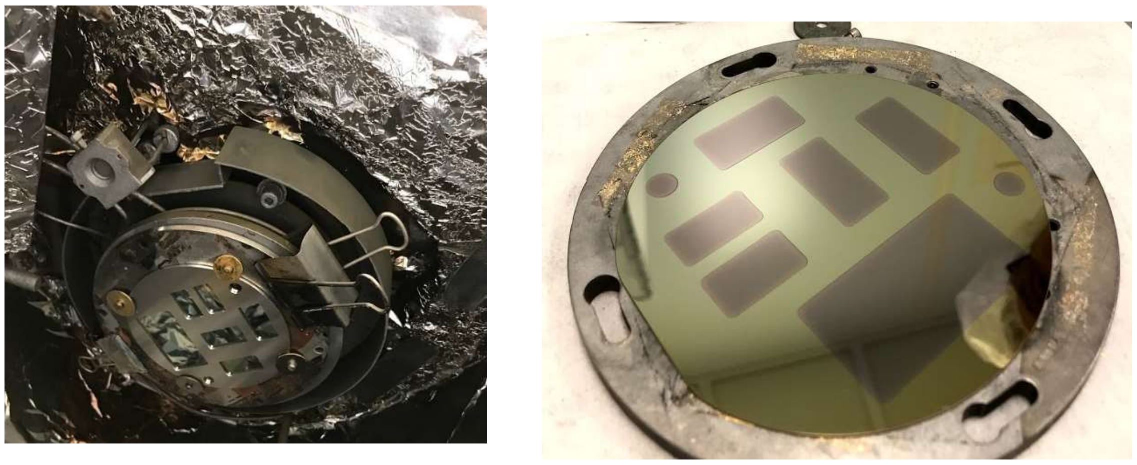


Figure 5-10 – Platinum Deposition on First Mask

The above deposition process was repeated with the second mask to deposit the remaining materials onto the wafer. These include the metal oxide materials (Aluminum Oxide and Titanium Oxide), and the top metal electrodes (Titanium and Platinum). Figure 8-11 shows the final fabricated wafer with three device diameters of 5.94mm, 7.56mm, and 14.04mm. Since the filament width will be largely localized to the stress location, the shortest path through the oxide layers will limit the width of the filament. Thus, the exterior dimensions of the bottom electrode or the top electrode should be largely irrelevant.

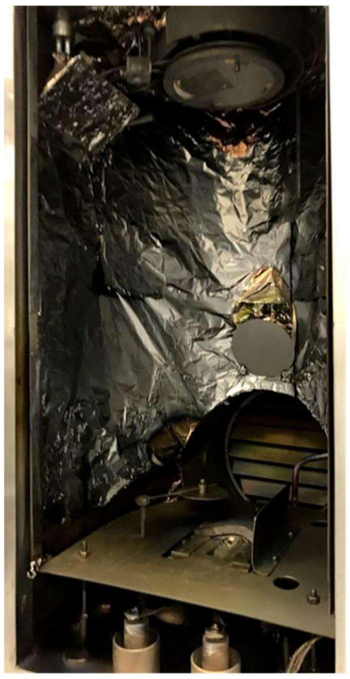
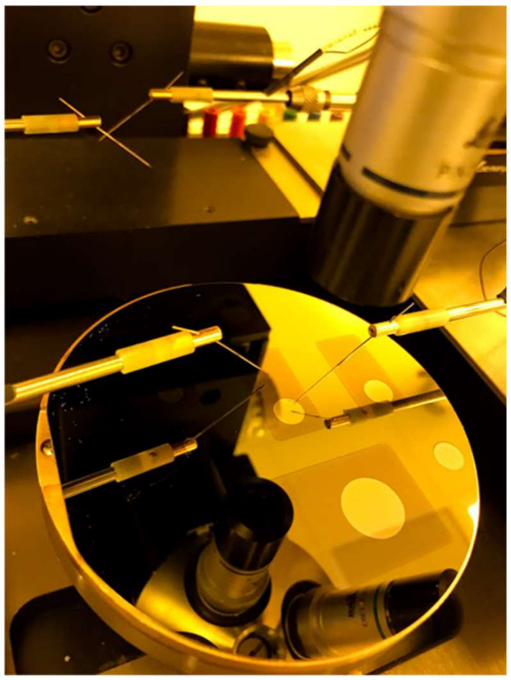
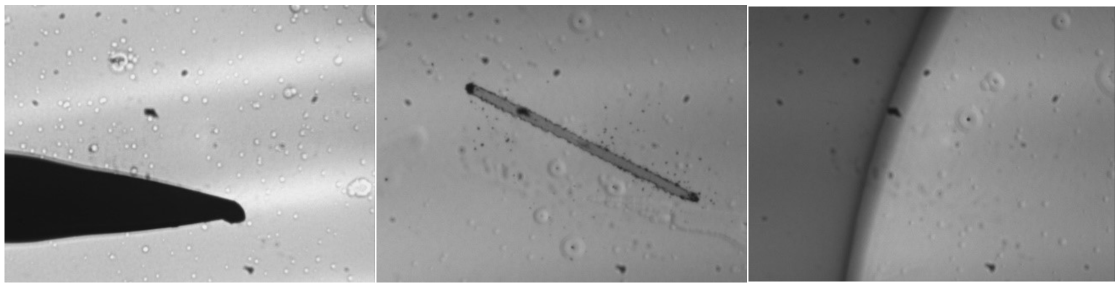



Figure 5-11 PVD Chamber and MicroProbe Station

Figure 5-12 shows the probe landed on the wafer, the etch mark (b), and the boundary of the top-electrode and metal stack, at a magnification of 2.5x. The figure shows some of the surface deformities from the deposition, which is a limitation with the equipment and process used.



(a) (b) (c)

Figure 5-12 - Die Photograph of Fabricated Devices (a) Probe Landed (b) Probe etch mark (c) Top-electrode/Metal Stack boundary

In order to analyze the composition of the ReRAM stack, I took a cross-section of the ReRAM bitcell. The Scanning Electron Microscopy (SEM) cross-section photo and the EDS spectra of the stack are shown in Figure 5-13, confirming the presence of the various materials deposited.

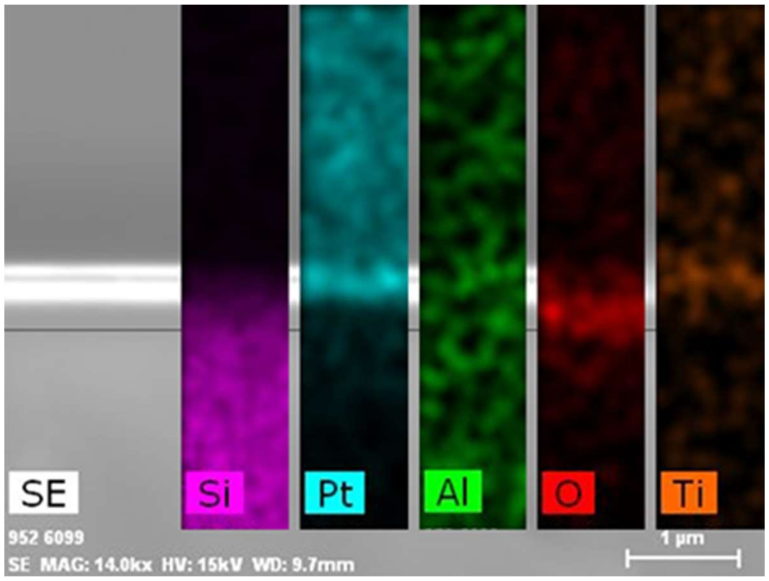


Figure 5-13 SEM Cross-section photo with EDS spectra of the ReRAM stack

SE is the Scanning electron image of the stack, where the stack can be seen as the lighter region. Si displays the presence of the Silicon substrate and is largely located beneath the stack. Pt displays the platinum element deposited in the stack layer. The Aluminum did not show up local to the stack alone and could be an artifact of the tool. Oxygen was detected in the stack and should be present in the Aluminum-Oxide and Titanium Oxide layer. Titanium is also shown to be present slightly higher than the rest of the layers, as part of the top electrode. Though the resolution of the individual material position and thickness is quite low, I can use this EDS spectra to confirm the presence and rough location of the various materials.

5.5 ReRAM Resistive Switching Behavior

The first part of my characterization consists of confirm the resistive switching behavior. In this experiment, my intent was to confirm that the resistant state itself could be altered between the low and high resistant states. Characterization was performed on the fabricated devices at room temperature. The Agilent 4155C parametric analyzer was used to drive the probe points and apply the program pulse. Figure 5-14 shows the oscilloscope measurement of the applied voltage pulse. The voltage ramps from 0v to a peak of 5v, with a step duration of 92ms, which was the shortest pulse duration possible with the parametric analyzer.

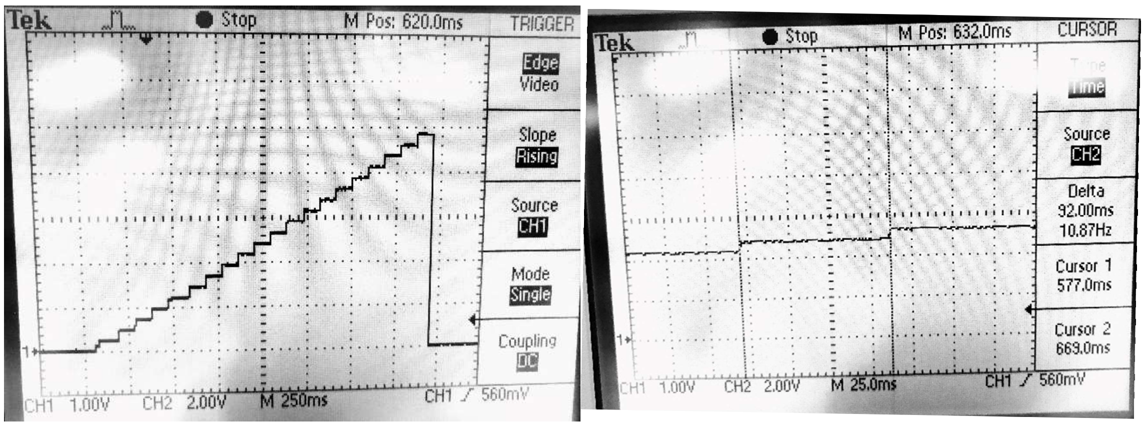


Figure 5-14 – Oscilloscope Measurement on the Applied Program Pulse

Figure 5-15 shows the SET transition from a high-resistive-state (HRS) to a low-resistive-state (LRS) and the RESET transition from HRS to LRS using bipolar program mode operation. The program operation was performed by applying a voltage from 0 to 6v with a current compliance of 100uA for SET operation and 1mA for RESET operation. The x-axis shows the voltage applied and the y-axis shows the current measured across the cell.

At the positive voltages, as the voltage ramps from 0v up to 4v, the current measured is at 1uA, reflecting the state of the cell. For this cell, this seems to reflect the open-circuit current of not a fully formed filament. At around 4v, I see the cell transition to abruptly to a higher current state. This could reflect the conductive filament being formed across the electrode material. Beyond 4v, the cell retains its lower-resistance state. A subsequent SET pulse, ramping again from 0 to 6v, confirms that the cell-state is retained and remains SET.

On the negative voltage side, as I ramp down the voltage from 0v to -6v, I see that the cell is able to maintain the low resistance state (and the filament) for most of the region. At -5.8v, the cell abruptly switches to a low-current/high-resistance state again. I can visualize that the negative bias repaired the oxygen vacancies created during the SET pulse, thus "breaking" the conductive filament between the two electrodes. A subsequent RESET pulse, ramping from 0v to -6v, confirms that the bitcell remains in a higher resistance state.

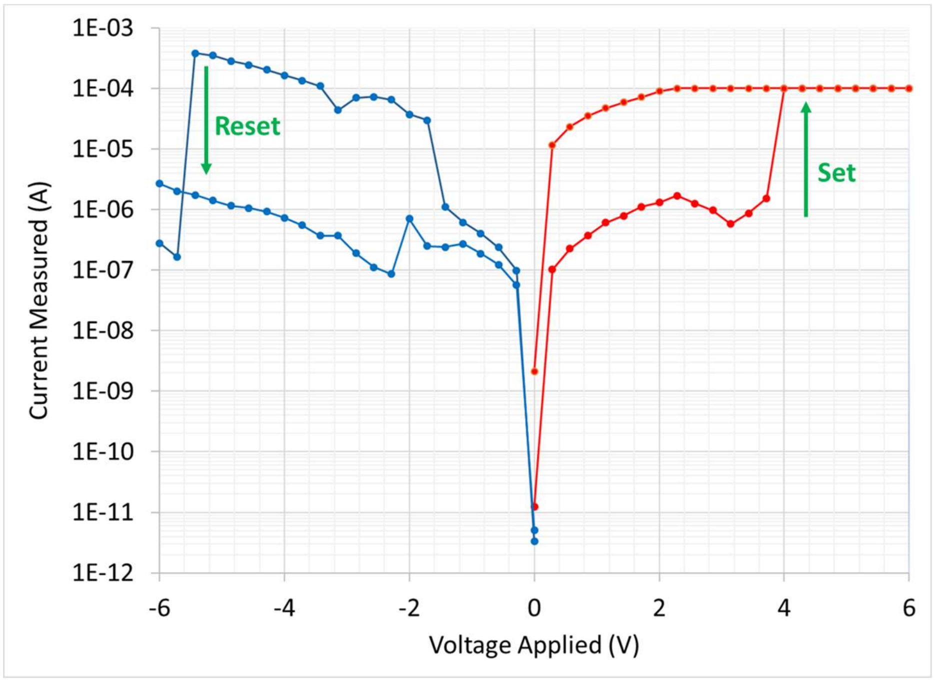


Figure 5-15 - ReRAM Switching between LRS and HRS in bipolar program mode

Published literature has shown two modes of write operations for ReRAM bitcells – bipolar and unipolar modes. Bipolar mode involves applying a positive voltage for SET-going operations, while using a negative voltage for RESET-going operations. Unipolar mode, on the other hand, uses positive voltage for both SET-going and RESET-going operations. The difference between the two operations is determined by the maximum voltage applied (higher for SET-going) and the current-compliance limit applied (higher for RESET-going). The results shown previously used the bipolar mode of operation. Unipolar mode is based on thermal acceleration of redox transitions and is simpler to implement but can lower cycling endurance. Bipolar mode is based on ionic migration assisted by electric field, has higher endurance due to the defects being conserved and is therefore generally a more popular method of program [28].

TiO_2 has been observed to switch in both bipolar and unipolar methods of resistive switching. I confirmed resistive switching operation in unipolar mode as well, as shown in Figure 5-16. The x-axis shows the voltage applied, and the y-axis shows the measured current.

For SET, the program voltage was ramped to 6v, with the current compliance set to 100uA, while for RESET, the program voltage was ramped to 3v, with a current compliance of 1mA. The figure confirms both successful SET-going and RESET-going operations. For SET-going cell starting at a high-resistance state (with low current measured), at 3.8v, there is an abrupt change in current measured reflecting a state transition to LRS state. After the transition, the current measured is higher, reflecting a low-resistance state implying the creation of a filament. For RESET-going cell, the current measured is linear with increased voltage, implying a constant resistance of 18.5K-Ω. At 2.5v, the RESET-going cell's measured current abruptly drops implying a state transition to an open-circuit, HRS state.

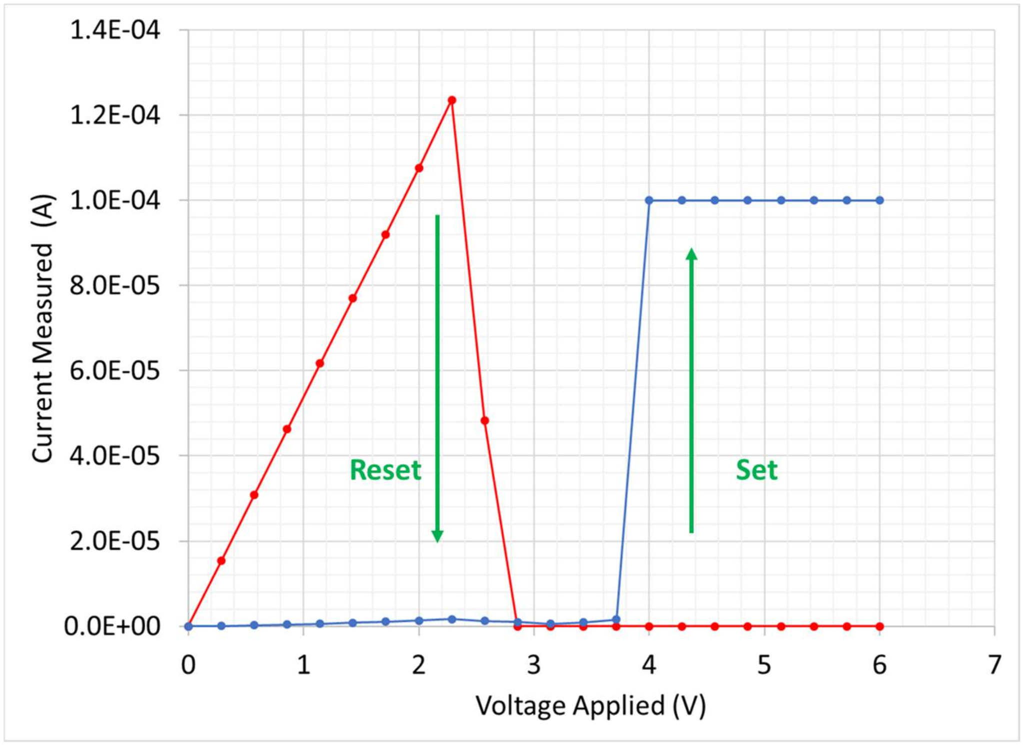


Figure 5-16 - ReRAM Resistive switching in Unipolar program mode

5.6 Threshold Behavior at Low Current Compliance Limits

My next set of measurements were intended to confirm the threshold behavior of ReRAMs. At low-current compliance limits, a program pulse does not affect the state of the bitcell permanently. In this mode, the bitcell acts as a passive device, allowing current to pass when the bias is present, but not affecting the overall state of the bitcell. Figure 5-17 shows the threshold behavior of the device on an LRS cell.

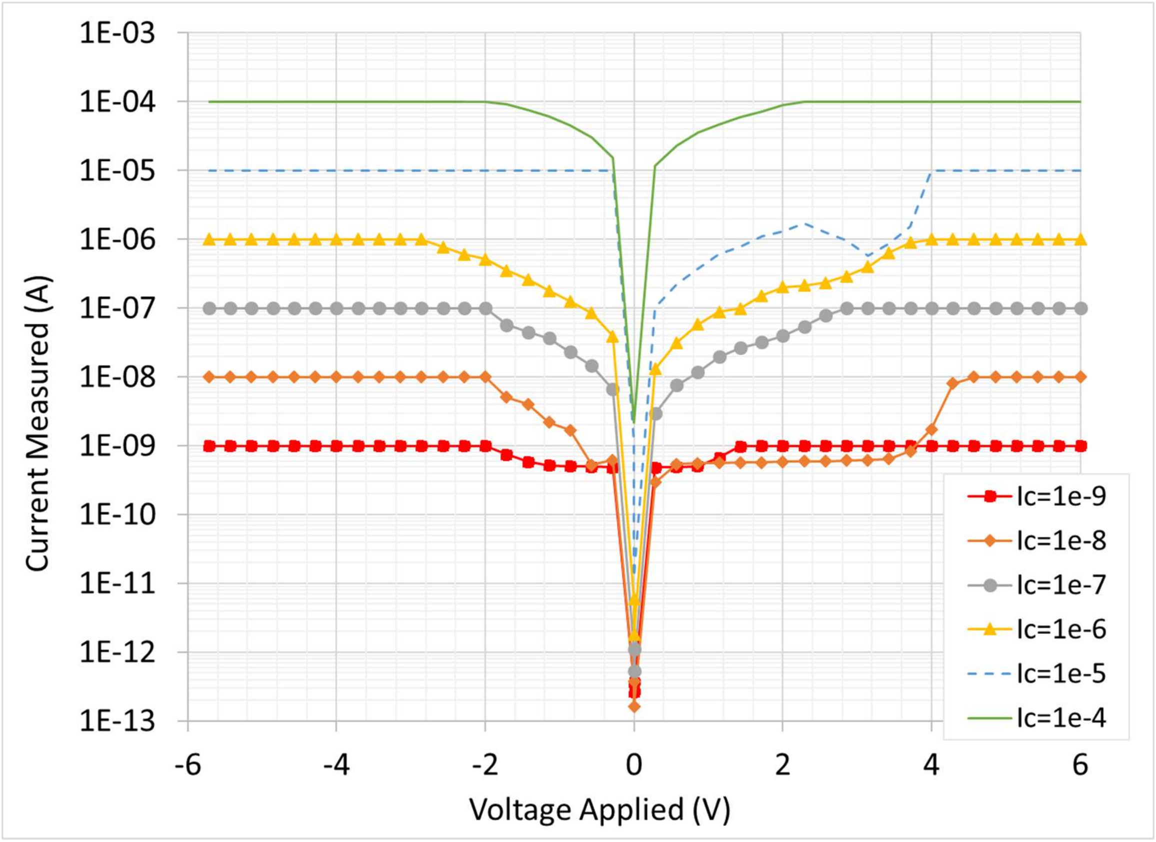


Figure 5-17 - ReRAM Threshold behavior at low current compliance (Ic) limits

I performed the measurement by ramping the voltage from 0 to 6v, and again from 0 to -6v for different current compliance levels. Different current compliance limits (Ic) were applied, starting from a low current compliance of 1e-9 and increasing to 1e-4, in orders of magnitude. The cell had an initial state of a low-resistance-state (conducting), and therefore as soon as the voltage rise, the current measured is clamped to the limit set by the compliance. At each successively higher current compliance level, the bitcell's state was not altered from its set/LRS state to a reset state/HRS, even when the negative voltage bias was applied.

5.7 Time Dependent Volatility Behavior

I next characterized the data retention of the cell. Data retention is defined as the duration after program for which the programmed state is maintained in the bitcell. Previous work from literature survey [27] indicates that the initial conductance of the bitcell is the key dependent variable for predicting amount of state change. The plasticity model proposed in [18] for example lists the following relationship between the change in conductance and the initial state.

$$\Delta G = \lambda_t \lambda_G$$

Here, λ_G represents the initial conductance and λ_t represents the change in time. This compact model implies that the change in conductivity with time is proportional with initial state of the bitcell. Conductance of the bitcell is measured as the inverse of the resistance, $1/R_{init}$, where R_{init} is the initial resistance.

My characterization method is as follows. I applied a program pulse in bipolar mode consisting of either positive or negative voltage bias, depending on whether cell was SET-going or RESET-going. My current compliance varied from 1e-9 to 1e-3 A.

For my experiment, I measured the cell resistance immediately after program (Rinit) and after a wait-time of 5 minutes and 10 minutes to assess the change in resistance.

I start from an LRS bitcell or a RESET bitcell, in which there is no conductive filament that has formed between the top and bottom electrode. Applying the program pulse of different current compliance either successfully or unsuccessfully completes the formation of the filament. As stronger, meaning one with higher current compliance, program pulse is applied, the probability of the filament formation is higher. Additionally, the thickness of the filament formed is also larger. Conversely, weaker, or lower current compliance, may produce filaments that are thinner or not at all formed. There is some movement of the filament after the program pulse which contributes to the filament relaxing causing the breaking or thinning of the filament. I expect that cells placed in intermediate states of resistance have a higher probability of this happening, causing movement towards a more RESET state.

Figure 5-18 presents the summary of the data collected. The x-axis is the initial resistance of the cell, measured immediately after the program pulse is applied, in log scale. The y-axis is the change in resistance after 5 and 10 minutes had elapsed, also presented in log scale. These results confirm the expected direction of resistance for the bitcells whose initial resistance, R_{init} was below 10-MΩ. For these cells, I see that for the most part, the change in resistance increased after 5 or 10 minutes, implying that the bitcell became *more* reset, or moved towards a higher resistant state after the wait time. For two bitcells in this region (below 10-MΩ), there was no change, and for one bitcell in this region (below 10-MΩ), the bitcell reduced in resistance slightly. These were anomalous behavior, whose cause needs to be investigated further. However, for all other bitcells, there was an increase in resistance, which is in line with the expected filament relaxation behavior.

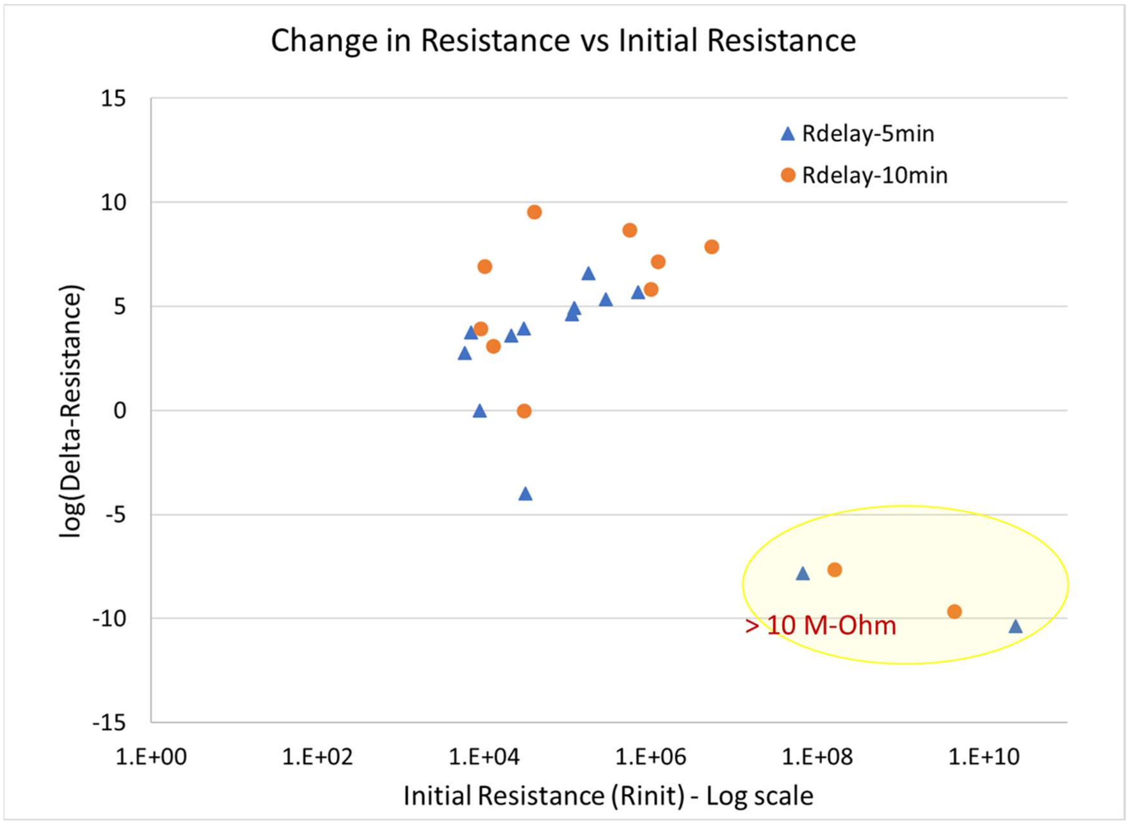


Figure 5-18 - Change in Resistance after 5 and 10 minutes delay as a function of the initial resistance. Log(Delta-Resistance) is calculated for the y-axis

For cells with Rinit above 10-MΩ, I observed a decrease in resistance after the 10-min wait time, for these hard-Reset cells. These were very high resistance state bitcells, whose resistance after the wait time shifted to a lower resistance state. This was true for both the 5- and 10-minute wait time. It is unclear the exact mechanism for this behavior. Other studies on single-crystal TiO2 ReRAM have indicated electrochemical resistive switching behavior after a post-annealing step [55], which could be a possible explanation. Since the commonality amongst these cells is that they all are very high RESET state, I can theorize that this might be caused by a transfer of defects from the top electrode to the bottom electrode, causing the bottom electrode to be the defect reservoir. This behavior was described in [28] as complementary switching (CS) during the absence of a current limitation with a positive voltage bias. After the wait time, the oxygen defects could have migrated from the bottom electrode back towards the top electrode causing the cell to move towards a lower-reset-state.

I categorized the measurement taken by the cell size and plotted the result in Figure 5-19. The bitcell diameters for Cell 2, 3, and 6 were 5.94mm, 7.56mm, and 14.04mm, respectively. The x-axis is the initial resistance of the bitcell in log-scale, and the y-axis is the change in resistance (Rdelay – Rinit), where Rdelay is the measured resistance after the delay wait time, again plotted in log scale. The data present here is for the combined 5- and 10-minute wait times. The plot confirms that the behavior observed is present on multiple devices of varying sizes and is not a function of the cell size. I do notice that the initial resistance of the cell seems to have a slight relationship to the cell diameter, with larger cells having a lower initial resistance. Although I did not focus the characterization on the impact of cell diameter, this would be a study for future work.

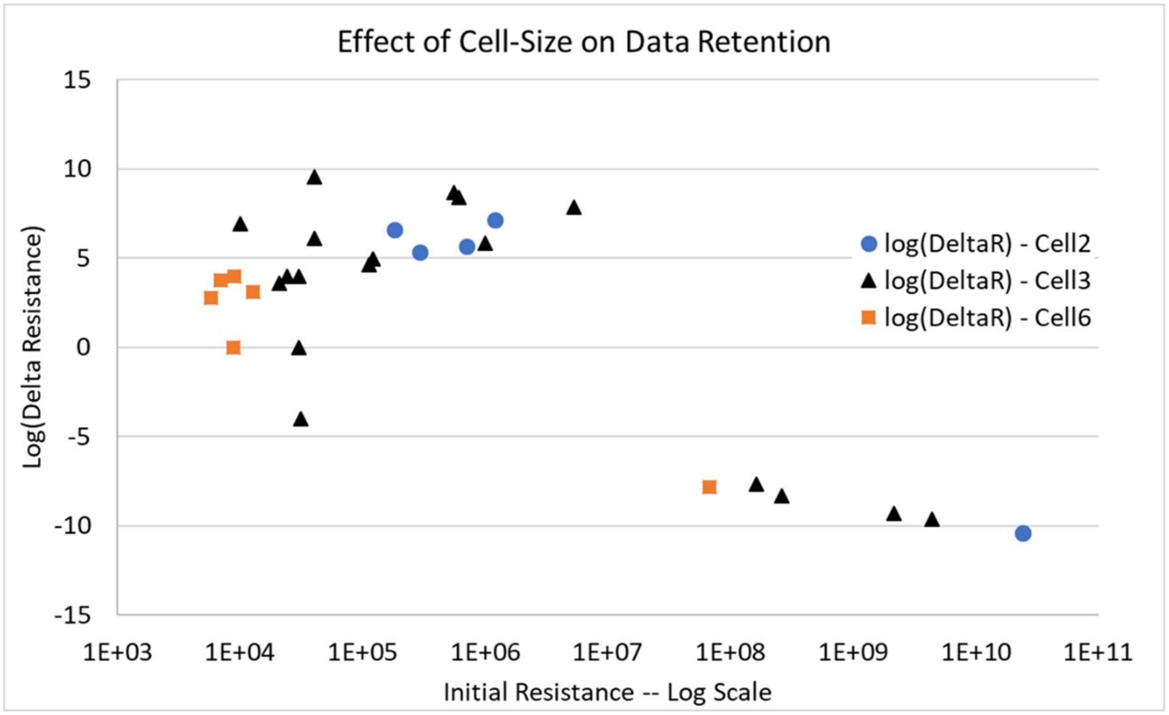


Figure 5-19 - Resistance change over time grouped by Cell sizes with trend observed across multiple devices. Diameters of Cell 2=5.94mm, cell 3=7.56mm, cell 6=14.04mm. Log(Delta- Resistance) is calculated for the y-axis.

Using the data on the bitcells below 10-MΩ, I fit the data to a linear equation of the log-log data points. Equation modeling based on the observed data for cell resistances below 10-MΩ yields the following relationship.

$$log(\Delta R) = 1.64 * \log(R_{init}) - 3.37$$

$$\Delta R = \frac{R_{init}^{1.64}}{10^{3.37}}$$

Here, R_{init} is the initial resistance measured immediately after the program operation and ΔR is the change in resistance after a wait-time.

Figure 5-20 plots the fit of measured results against the predicted equation model for bitcells with their initial resistance below 10-MΩs. The R-square of the fit is 0.57.

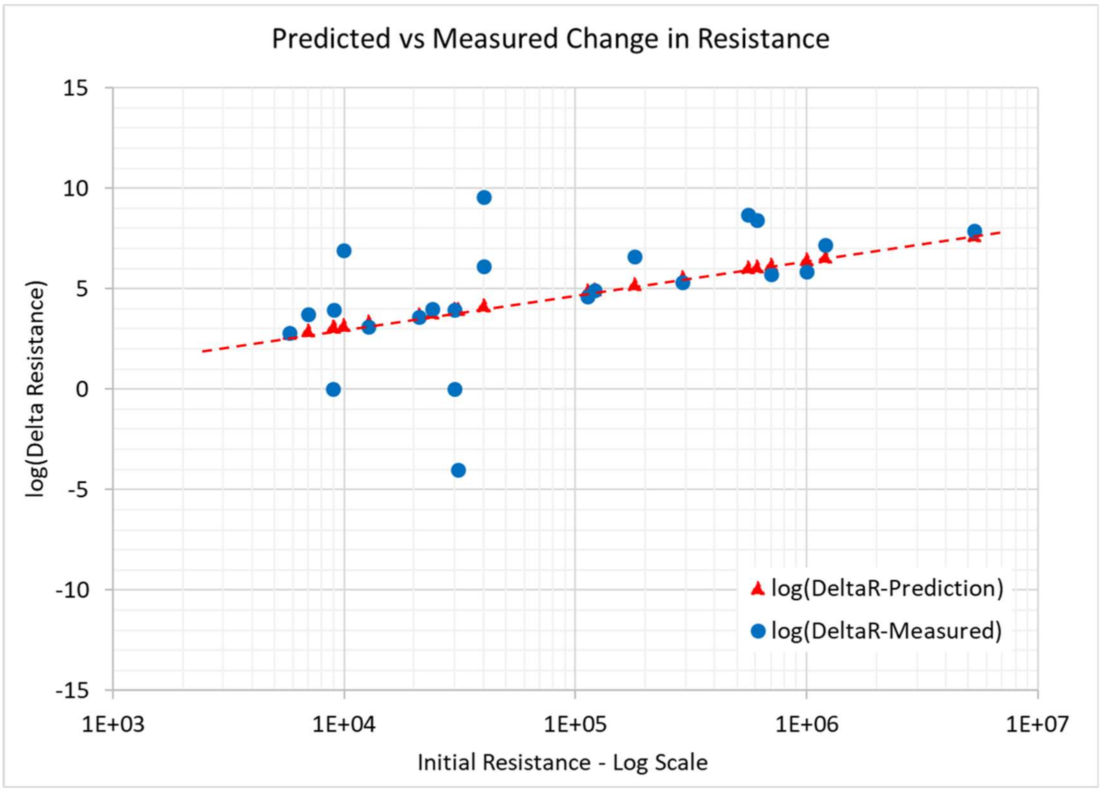


Figure 5-20 - Predicted vs Observed change in resistance for cellstates with Rinit below 10MΩ. Log(Delta-Resistance) is calculated for the y-axis.

For my final set of experiments, I measure the effect of current compliance applied to the data retention time as a function of time with measurements at 2 min, 4 min, and 8 min. The cell was initially placed in an LRS state and increasing amounts of current compliance was applied. The program current compliance level affects the placement of the cell, with low current compliance levels not successfully moving the cell from an HRS to an LRS. Figure 5-21 demonstrates the observed cell relaxation for the time range of around 10 minutes, collected at the following four specific points: Immediately after program, 2 min, 4 min and 8 min. The x-axis tracks the time elapsed after the program pulse, measured in seconds, while the y-axis tracks the actual resistance measured. A read voltage of 50mV with a current compliance set to 1uA was used for the measurement. I see the change in resistance of four bitcells over the measured time period. The dashed lines in the plot denote possible range for intermediately placed cell that have a high change in resistance over the time period. Bitcells placed below the bottom dashed line would be well-SET cells (LRS), while those placed above the top dashed line would be well-RESET cells (HRS).

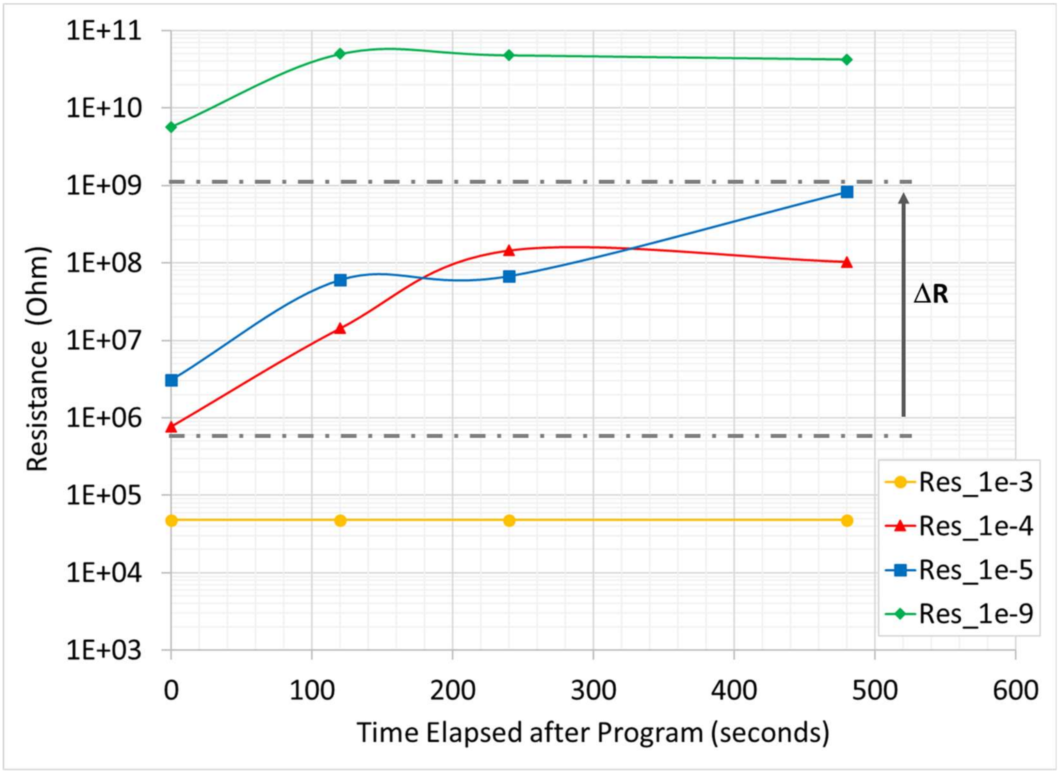


Figure 5-21 - Resistance change for different Program Current Compliance values

For this cell, I define a "well-SET" cell as below 100kΩ, and a well RESET cell of above 100MΩ. The results show that a well-SET cell, formed by applying a high amount of current compliance (Res_1e-3), is able to retain its SET value of 48kΩ through the measured time. Similarly, a well RESET cell (Res_1e-9), with a resistance value of 5.7e9Ω, remains RESET which a final measured value of 4.25e10Ω. The two intermediately placed cells (Res_1e-4 and Res_1e-5), show the resistance values increase with a much higher delta change in resistance. The cell placed with a 1e-5A current compliance (Res_1e-5), changed from 3.11e6Ω to 8.29e8Ω. These observations confirm the relaxation behavior of an intermediate cell with a filament relaxing to break its conductive bond, resulting in a higher resistance state. Note that this relaxed higher resistance state is still lower an order of magnitude lower than the well RESET state of 4.25e10 with the Res_1e-9 current compliance. The results indicate that well-set cell with a high current compliance of 1mA retained the state for the full 8-min duration, while intermediate program Ic levels of 1e-4 and 1e-5 shifted the cell state to two orders of magnitude higher resistance.

To study the effect of current compliance on the change in resistance, I plot the observed data in a different way. Figure 5-22 shows the change in resistance as function of the current compliance for three measurement points – immediately after program (Rinit), 2 minutes and 8 minutes after program (R2min and R8min, respectively). The x-axis tracks the program current compliance used (in amperes), and the y-axis tracks the resistance measured on the bitcell. The program pulse was applied at a specific current compliance level, and then the resulting resistance level was measured at the three delay points. Note that I started with an HRS (RESET) cell prior to the measurement. I observe that at 1e-9 and 1e-6, the cell remains in the HRS state. At program current compliance of 1e-5 and 1e-4, there is a marked shift in the bitcell resistance, starting at a lower resistance level and gradually moving to a higher resistance level. At Ic of 1e-5, the bitcell resistance started at 3.11MΩ and after 2 minutes, shifted to 60.7MΩ, and after 8 minutes, measured to be 829MΩ. Similarly, for Ic of 1e-4, the bitcell resistance started at 0.7MΩ and after 2 minutes, shifted to 14.3MΩ, and after 8 minutes, measured to be 103MΩ. Finally, the bitcell that was programmed with an Ic of 1mA, remained as a SET cell, below 50k Ω. These results confirm that intermediate current compliance limits show the greatest change in resistance.

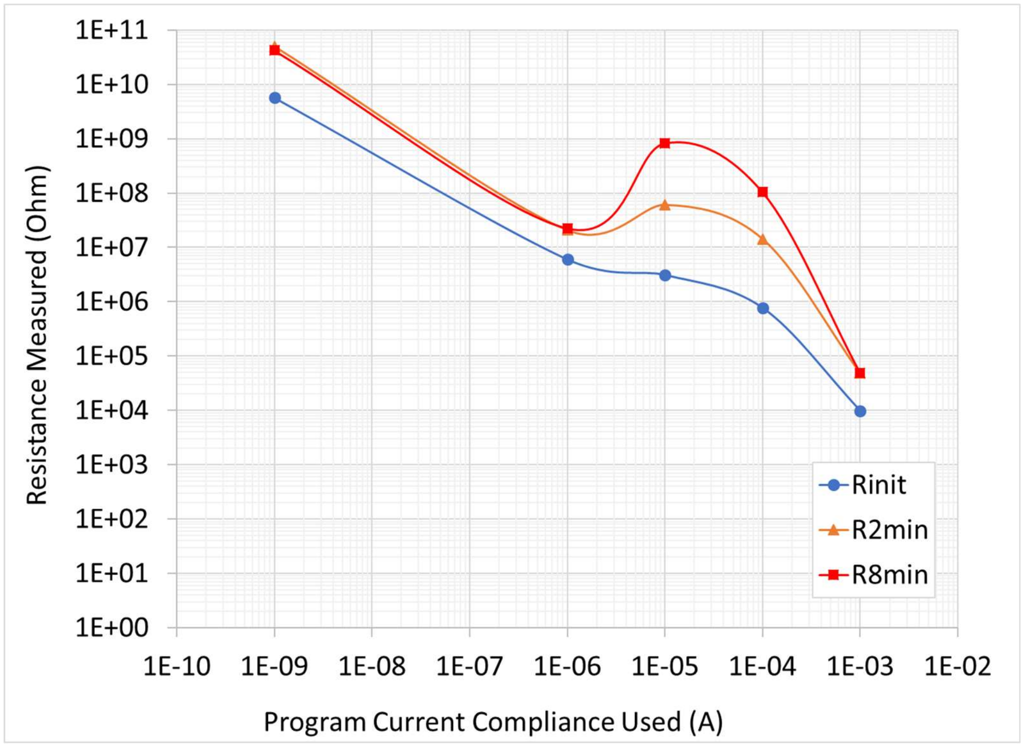


Figure 5-22 - Resistance change as a function of Program Current Compliance.

One interesting observation to note is that even for the specific points where the resistance does not shift, the Rinit datapoints all measure to be slightly lower resistance. One possible cause of this could be due to the measurement following a program pulse with a high voltage bias (6v) possibly having an effect on the state of the cell. This could possibly be due to thermal effect from the high voltage applied during the program pulse and is a topic for future exploration.

In this section, I presented data that shows that the cell's data-retention time could be modified by reducing the current compliance applied during the program pulse. This intermediate current compliance acts as a digital volatile mode for the bitcell. A system making use of the cell in the digital volatile mode must calibrate the read threshold currents for this lower range, to properly interpret the intermediate state as well.

5.8 Impact on Write Energy and Endurance

In this section, I estimate the benefit in write-endurance that can be gained from the lower write-current applied. From write-energy point of view, I see that the intermediate mixed-volatility state requires 1-2 orders of magnitude less write-current. Instead of 1mA, applying 10uA might suffice to program the cell in the intermediate state. The write energy is the product of the current, voltage, and the duration of the stress applied to the cell. From a write-stress point of view, the cell is in this intermediate mode is now seeing 100x lower write energy per write operation. The following equation from [33] relates the relationship between the energy applied per cycle to the maximum amount of energy tolerated by the cell.

$$E_{max} = N_{cmax} * E_{1-cycle}$$

Here, E_{max} is the maximum energy that the ReRAM bitcell's dielectric material can sustain, E_{1cycle} is the energy seen by the cell in one cycle, and N_{cmax} is the maximum number of write cycles that can be performed. N_{cmax} is the measure of write-endurance for the cell. As the equation points out, there is an inverse relationship between the energy applied per cycle to the overall number of write cycles tolerated by the cell.

Since I apply 100x lower write energy per cycle, I can expect that the N_{cmax} would increase by 100x. My original stated write-endurance for ReRAM was 10^5 to 10^8 cycles. This can therefore be expected to be increased to 10^7 to 10^10 cycles. Although this is still not near the write endurance tolerated by the DRAM cell, this amount of improvement allows for the cell to approach the write-endurance limits needed for main-memory applications. Furthermore, by combining wear-leveling techniques used in flash memory chips, the effective write-endurance could be further increased.

In terms of the total write energy, there is an increased amount of write cycles needed to perform the refresh in the cases where the data needs to be maintained for long periods of time. For the calculation given above, there is a 100x reduction in the write energy applied per pulse. However, after 100 refresh cycles, where the data is written back to the cell during the refresh cycle, we lose the benefit of the write energy and write endurance to the cell. In this case, for those data where the data needs to be persistent, we may selectively apply a high write energy to begin with to store the data in a non-volatile state. In addition to the write-endurance impact, there is an impact to the overall system performance as well. Since write energy is a function of the current amplitude and the duration of the program pulse, the lower write energy could also translate to a faster write cycle. For certain applications, having the lower write cycle might be critical for overall system performance where the shorter write latencies could be more easily hidden and prevent stalls due to write operations.

5.9 Post-Characterization SEM

I did a final SEM photo of the characterized wafer to assess the thickness of the material deposited. I used Tescan GAIA FIB/SEM machine from UMD's AIM lab to perform this measurement. I first performed a FIB (Focused Ion Beam) cut on the wafer to ensure a sharp cross-section edge to make the measurement. Figure 5-23 shows the wafer material inside the SEM chamber. The wafer is sliced and mounted onto a vice inside the chamber. The figure shows the wafer positioned directly under the microscope.

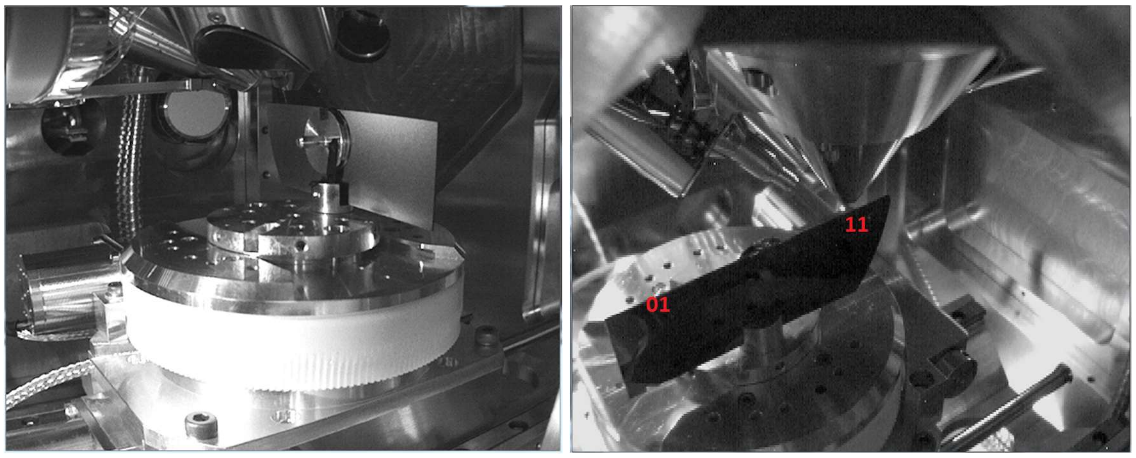


Figure 5-23 - Sliced Sample inside GAIA SEM Chamber

Figure 5-24 shows the inverted cross-section photo of the wafer. Table 5-2 summarizes the measured thickness of the materials against the target thickness.

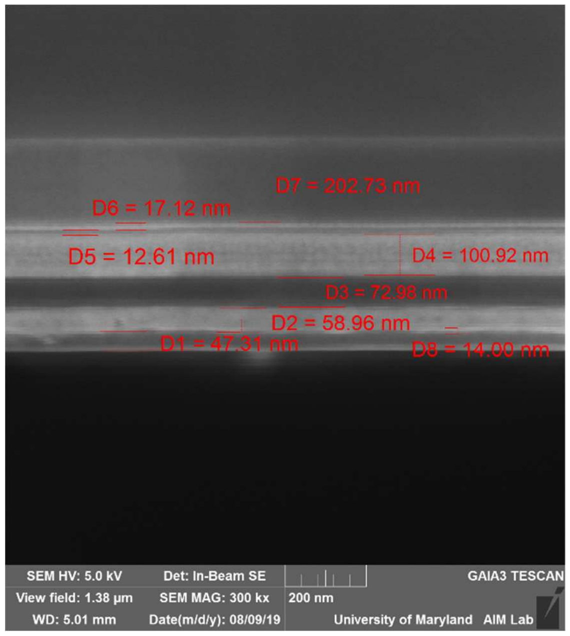


Figure 5-24 – SEM Thickness Measurement

Layer #	**Material**	**Target Thickness (nm)**	**Thickness Measured (nm)**
1	Platinum (Pt)	60	73
2	Aluminum Oxide (Al2O3)	5	59
3	Titanium Oxide (TiO2)	30	
4	Titanium (Ti)	15	14
5	Platinum (Pt)	60	47

Table 5-2 – SEM Analysis of Deposited Thickness

The measurement showed that the material deposited is on the order of the target thickness of the different materials I targeted. The aluminum and titanium oxide material could not be differentiated in the SEM photo mode, but I estimate the sum thickness to be larger than the target thickness of 35nm. Since this device was not a virgin material, the process of applying the electrical stress likely caused the material to diffuse into adjacent layers. This can be seen at the bottom of the wafer, being diffused into the silicon. Overall, the thickness of the material measured appears to be slightly larger than my intended thickness overall. In the future, using smaller mask dimensions and alternate deposition process might more accurately control the deposition of the materials.

5.10 Conclusion and Future Work

A ReRAM device composed of a TiO2/AlO2 metal-oxide stack was fabricated, and characterization results were analyzed. Resistive switching and threshold behaviors were observed. Additionally, time-dependent relaxation of the cell resistance was observed, causing those cells placed in intermediate cell states, by using a lowered program current compliance, to see the greatest shift. This is in-line with my target use of using lower write-energy to place the cell in a mixed-volatile state having a lower data retention time.

As mentioned in the beginning of the chapter, the motivation for this research work is to verify that we are able to observe that ReRAM could be operated ina n intermediate state where the formed filament of oxygen vacancies in the metal oxide is able to repair itself after a period of time. The experiment results are a proof-of-concept of the possibility of ReRAM as a digital volatile memory. With regards to scaling, I expect the observed behavior to be retained, since scaling occurs in the dimension of the ReRAM metal planes and typically not as much in the thickness between layers. Since the filament is localized to the points of the stress, the movement of the oxygen vacancies should follow at a similar rate even at advanced nodes. Volume data on this phenomenon would provide more data points which would lead to better averaging of the program current compliance and the expected rate of the relaxation.

Several points of observation merit a closer look. I have mentioned these in the experimental discussion, with regards to anomalous points of data observed for very high resistance bitcells and the change in resistance after a delay. These can be further mapped to a function of the current compliance applied and the sequence of preceding program pulses. Due to the large dimensions of my bitcell, it is possible that multiple filaments have formed in parallel, that may be the cause of the cell behavior at the very high resistance states (above 10 MΩ). For this reason, future work can try to make the dimensions smaller, towards the target dimensions seen in the intended application.

Oxygen partial pressure has been known to have a strong impact on the movement and retention of oxygen vacancies in metal oxides [55, 56]. The effect of oxygen partial pressure in introducing contaminants to the material layers during the fabrication process needs to be studied more closely. The PVD fabrication for my experiment was performed in a low-pressure chamber, however there could be oxygen contaminants between the layers. Specifically, between the mask steps, where the top electrode of Titanium might have oxidized to form TiO2. The SEM analysis did not have sufficient resolution to identify the regions clearly. Future work can analyze the material fabricated with higher resolution. In the operational mode we expect to use, the temperature ranges between -40C to 100C, and therefore we do expect a high variation from the oxygen partial pressure on the relaxation of the oxygen vacancies. In space applications, the lower oxygen partial pressure might slow the movement and relaxation of the oxygen vacancies, thus increasing the data retention of the filaments created in the intermediate state.

Lastly, the biggest change is to gather volume data for the characterization results so that the noise could be further isolated, and the cell retention relationship could be more robustly developed for design of the memory system application. This requires fabricating a full array, with more than 1000 bitcells so that statistical analysis could be performed to more completely characterize the bitcell behavior.

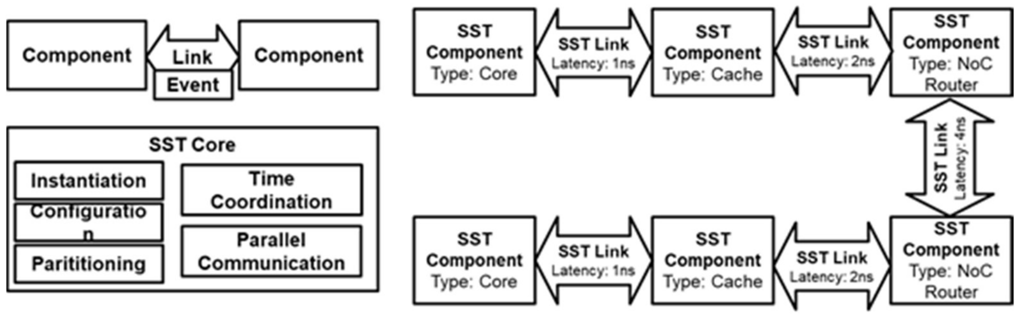
6 Architecture-Level Simulations

In the next thrust of my research work, I looked into architecture-level simulations that would provide the impact of various design configurations in my ReRAM architecture. I compare this to a conventional DRAM based architecture and vary key parameters to analyze the impact of them. I provide a brief introduction into the simulation methodology I used and then provide the results of my baseline architecture comparison. I next study the impact of a central ReRAM based design with varying number of cores on the performance and the energy consumption of the architecture.

6.1 SST Simulator

SST is a simulation tool developed by Sandia National Laboratories, which provides a flexible framework as a "Parallel Discrete-Event Simulator" and allows for a multitude of custom simulators. The tool has demonstrated scaling to over 512 processors, and comes with many built-in simulation models for processors, memory, and network, including DRAMSIM. The tool follows a modular OpenMPI interface based on linking together various components (see Figure 6-1 from the SST website).

The figure shows the operation of the simulation framework driven by an SST core engine that keeps track of the instantiated elements, components, and the links in the simulation. Each component represents a physical structure in the architecture, such as a CPU, the network router, the memory, or the cache, for example. Each component is connected to another component through a link with a latency property, which is used to track the timing of the simulation. This framework allows for modular use of different elements that are developed outside of Sandia. For example, to model the DRAM memory, I used DRAMSIM3 as the backend memory model.



Reference: http://sst-simulator.org

Figure 6-1 SST Component-based Framework

The SST framework is component based, cycle-accurate simulator for fast comparison of different architectures. I have used SST to model the ReRAM-CPU architecture using the following external components (element libraries):

● MemHierarchy	- Cache and Memory
● DRAMSim	- DDR DRAM Memory

● Miranda	- Pattern-based CPU model
● Merlin	- Network router model and NIC
● Messier	- Model ReRAM with asymmetric read & write latencies

6.2 Baseline Architecture Comparison

I performed an initial simulation on the STREAM and GUPS benchmark on the architecture shown in Figure 6-2. The DRAM architecture was roughly based on the Intel Knights Landing platform and a comparative architecture using ReRAM instead of DRAM was used. The left side of the figure shows the baseline DRAM architecture. A mesh topology with 6 rows and 8 columns is used to support 36 CPU processor units, along with dedicated L1 and L2 cache blocks. Additionally, there are six memory controllers that connect to 4GB DDR3 main memory blocks, to provide a total capacity of 24GB.

On the right side of the figure, the ReRAM architecture that I used is presented. This version shows a tiled architecture, again with 36 CPU processor units with dedicated L1 and L2 cache blocks. The main memory in this architecture, however, consists of 36 ReRAM blocks each of 0.9GB located adjacent to the CPU tiles, along with the memory controller. This is in-line with the tiled ReRAM-CPU layout that I presented earlier. This architecture also uses a mesh topology of 9 rows by 9 columns.

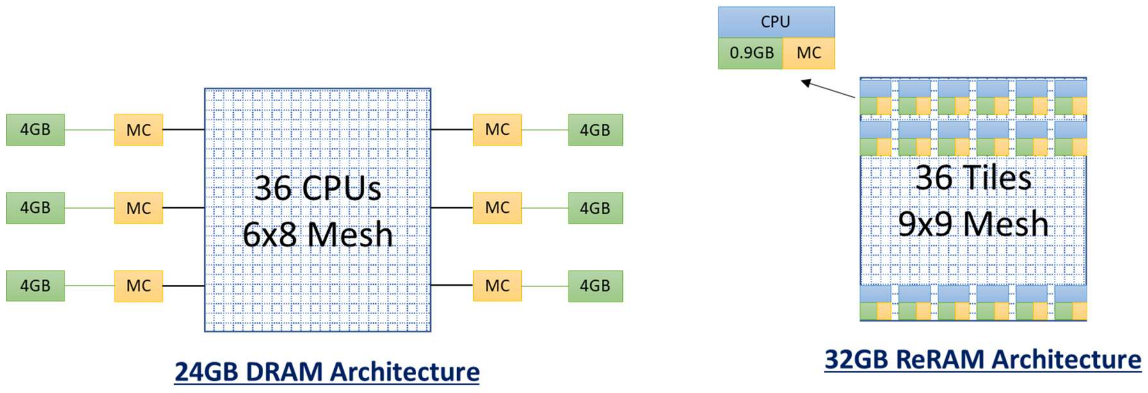


Figure 6-2 Architecture Comparison

The key architecture parameters are provided in Table 6-1 for comparison. I used our in-house DRAMSIM2 simulator to model a dual-channel DDR3 Micron device with a speed grade of 1333-J. For the ReRAM memory, I used the Messier element in SST to model asymmetrical read and write latencies of 200ns, and 1us, respectively. The peak memory bandwidth for DRAM is 10.4 GB/s per channel, for an aggregate bandwidth of 124.8GB/s. A very high NoC link bandwidth of 96GB/s per link was simulated to allow the NoC latency not to be an issue for the comparison.

Main Memory	DRAM	ReRAM
# of Cores	36	36
# of Mem Controllers	6	36
L1 Cache size	32KB	32KB
L2 Cache size	1MB	1MB
Main Mem Parameters		
Mem size	24GB	32GB
Mem Latency	~30ns DRAMSIM2 simulated	R: 200ns W: 200ns ,1us
request_width (access granularity)	64B bus-width = 8B Burst-length = 8	8B bus-width = 8B burst-length = 1
Mem Bandwidth	10.4GB/s per channel 124.8 GB/s aggregate	0.3 GB/s per bank 10.8 GB/s aggregate
Topology	6x8 mesh	9x9 mesh
NoC Link Bandwidth	96 GB/s	96 GB/s

Table 6-1 Summary of SST Architecture Details

The following lists the pseudo code for both benchmarks.

STREAM Benchmark: `a[i] = b[i] + k * c[i];`

```
MemoryOpRequest* read_b = new MemoryOpRequest(start_b + (i * reqLength), reqLength, READ);
MemoryOpRequest* read_c = new MemoryOpRequest(start_c + (i * reqLength), reqLength, READ);
MemoryOpRequest* write_a = new MemoryOpRequest(start_a + (i * reqLength), reqLength, WRITE);
write_a->addDependency(read_b->getRequestID());
write_a->addDependency(read_c->getRequestID());
```

GUPS Benchmark: `a[b[i]];`

```
MemoryOpRequest* readAddr = new MemoryOpRequest(addr, reqLength, READ);
MemoryOpRequest* writeAddr = new MemoryOpRequest(addr, reqLength, WRITE);
writeAddr->addDependency(readAddr->getRequestID());
```

The STREAM benchmark consists of two read operations followed by a dependent write operation. The GUPS benchmark has a read and a dependent write operation, with the address being randomly generated. The STREAM benchmark has dense memory access, meaning that the address locations in memory are accessed in sequential order and therefore I expected that DRAM's higher-access granularity would be more favorable for this benchmark. The GUPS benchmark has sparse memory access, for which I expect ReRAM's low-access granularity to be more favorable.

Figure 6-3 shows the SST simulation result of the comparison between DRAM and ReRAM based main memory architecture. The y-axis in the plot reports the execution time of the simulation, where a lower number is better (faster). The simulation was performed with a Miss Status Hold Register (MSHR) queue depth of 2, meaning that at any time, two outstanding requests could be stalled at the individual memory controller. The plot shows the results for both Stream and GUPS benchmarks for DRAM, and two version of ReRAM – one with 200ns write latency and a second with 1us write latency, both versions have a read latency of 200ns. The access latency for DRAM is set by DRAMSIM as a function of the pending requests and stalls.

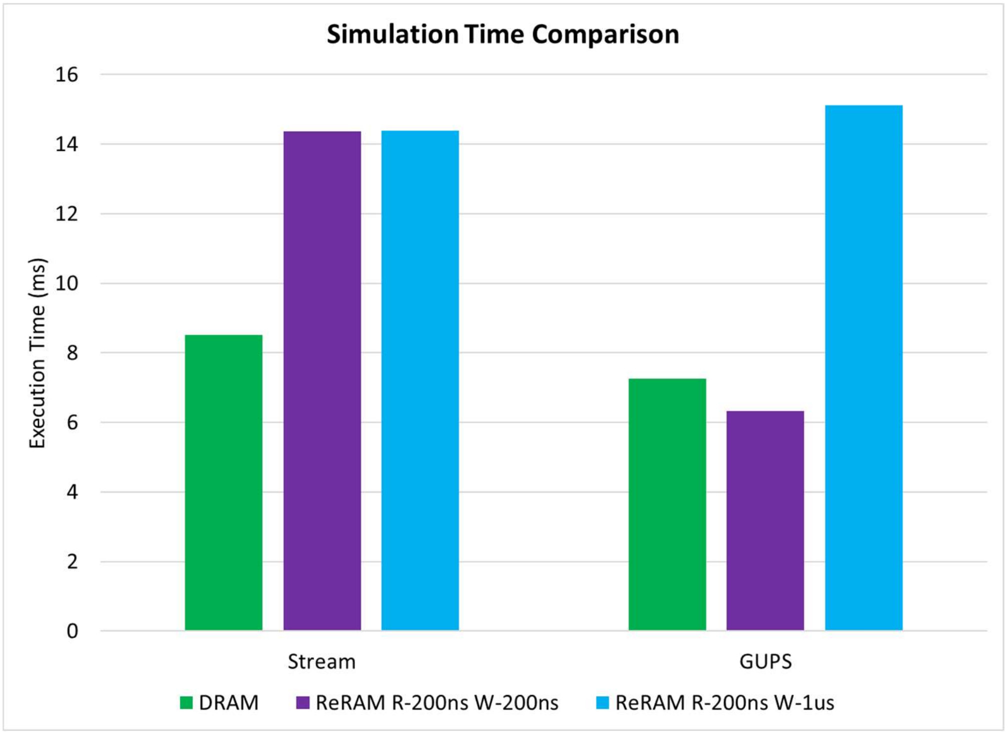


Figure 6-3 SST Simulation Result

The comparison show that DRAM outperforms ReRAM for Stream applications regardless of the write latency times. My simulation result shows that when the write-latency is reduced, for the STREAM benchmark, there is no noticeable improvement with the improved ReRAM write-time, and DRAM performs more favorably as expected. Because of the ratio of read to write operations is 2 to 1, the effect of a "faster" write latency does not improve the overall execution time in this scenario. For GUPS benchmark, however, ReRAM slightly outperforms DRAM in the shorter write latency configuration. However, DRAM still outperforms ReRAM when the write latency is 1us. This is due to shorter request length of GUPS and the irregular access pattern not allowing for the write requests to be re-ordered and thus mitigated.

The memory latency breakdown of both simulations is presented in Figure 6-4. The latency is reported from both the memory controller point of view, and from the CPU overall point of view. The memory controller latency is largely dominated by the memory latency itself, with some additional overhead depending on the number of stalls seen by the requests. As seen in the figure, there is a huge discrepancy between the two. For DRAM, the average memory controller latency for both benchmarks were 32ns. However, the average CPU latency for STREAM was 194ns, while for GUPS was considerably higher at 951ns. This goes back to my original motivation of addressing the memory bandwidth wall problem resulting in these huge discrepancies. The problem is worse for GUPS due to its inherent finer granularity which prevents access overhead from being amortized over a larger amount of data.

For the STREAM benchmark, the latency reported from ReRAM's memory controller point of view was close and slightly higher than the average overall latency from CPU point of view. The reason for the CPU latency being lower can be explained by a higher percentage of cache hit with the STREAM benchmark that allows for 67% of the accesses to be serviced by on-board caches. Since cache access is much lower than the ReRAM main-memory access, the overall latency is slightly lower. For the GUPS benchmark, however, the DRAM trend is also present with ReRAM – the overall CPU latency is much higher than the memory latency itself. Here, this implies again that there is a higher number of stalls that cause the performance of the system to be limited, not by the memory latency but by queuing of the requests.

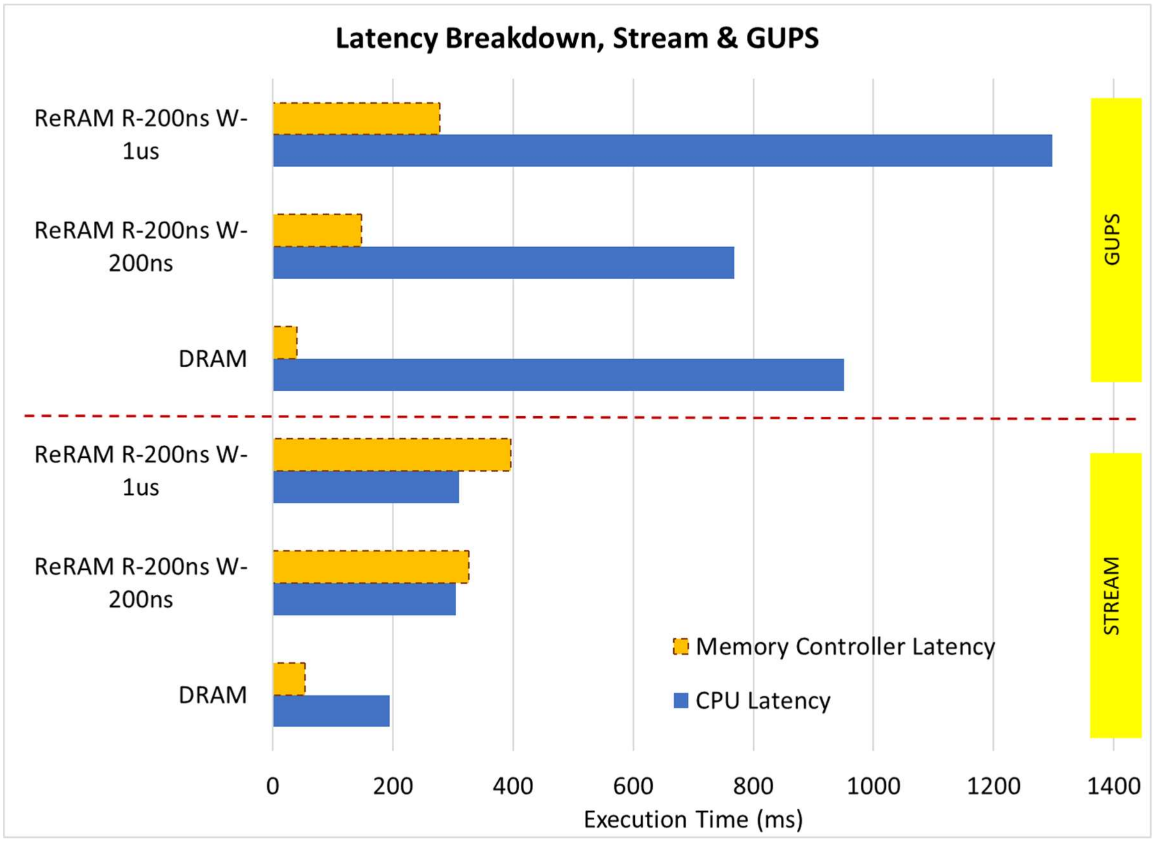


Figure 6-4 Memory Latency Breakdown, Queue Depth=2

With regards to the impact of the write-latency, STREAM benchmark reported very little change between 200ns write time and 1us write time, as I saw in the previous simulation study. With GUPS however, I see that average access latency, which is a combination of the read and write times, is reduced with the "faster" 200ns write latency time. The ratio for write-to-read is also higher with GUPS with 1-to-1 vs 1-to-2 with STREAM, causing the higher write latency to negatively impact the GUPS more.

The difference between the memory latency and the overall latency is an indication of the amount of stall occurring in the architecture. Because the access pattern is random, the cache hit rate with the GUPS benchmark is close to 0%, combined with the queue depth of 2, this causes more of the memory requests to be stalled at the CPU with the GUPS benchmark. The results imply that finer access granularity on ReRAM benefits GUPS benchmark with 200ns write latency. For the 1us write latency, the results show no ReRAM advantage for the GUPS benchmark. In the STREAM case, DRAM performs better over ReRAM regardless of the write-latency.

6.3 Impact of Memory Parallelism for ReRAM

Memory queue depth and the number of memory controllers are some of the key parameters that affect overall system performance. To assess the impact of the queue depth on the performance, I increased the MSHR queue depth from 2 to 10 at the Memory Controller. The results are presented in Figure 6-5. The y-axis again is the execution time of the simulation with lower execution time meaning faster performance of the architecture. Two additional sets of information are presented from the previous graph – the results with queue depth of 10 (5x queue depth from previous) for ReRAM.

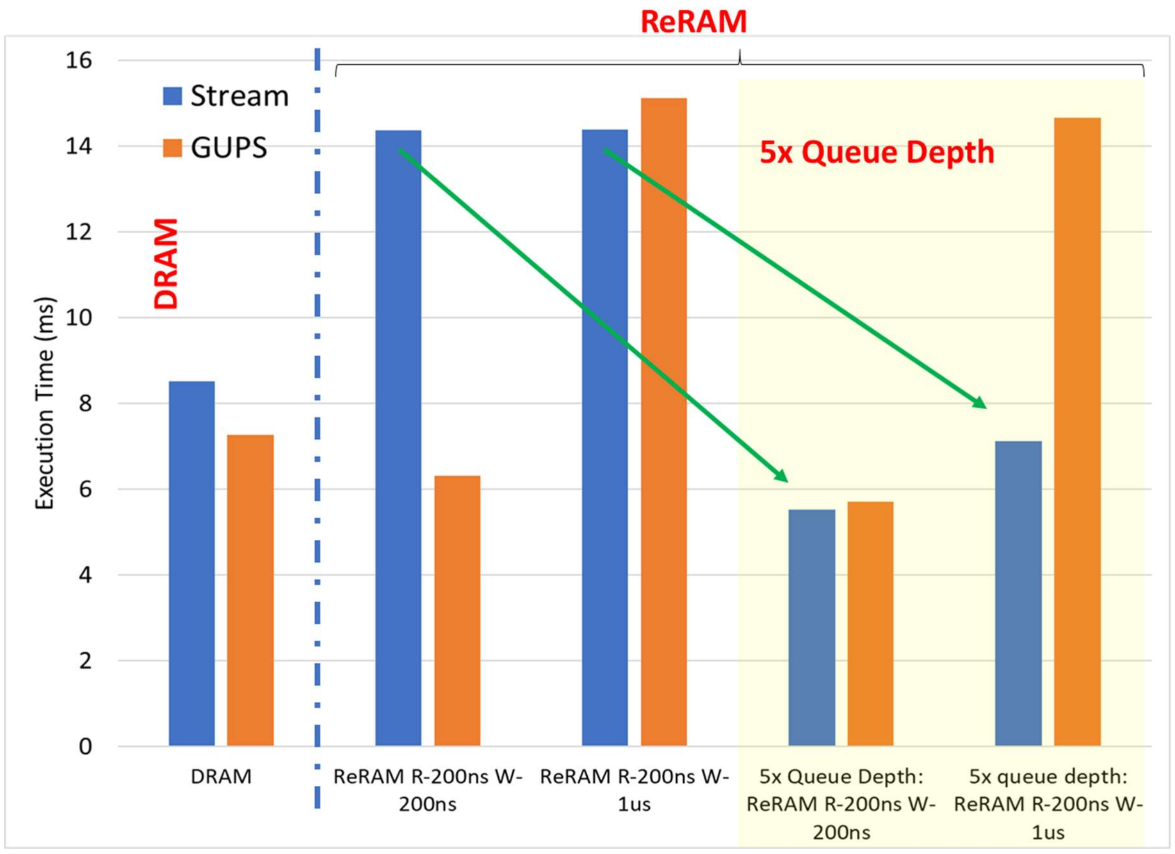


Figure 6-5 Impact of Queue Depth

In comparison with the queue depth of 2, I see a significant improvement in the execution time in the case of STREAM benchmarks for both the slow and fast write latency times. There was little to no improvement of the performance as the queue depth was increased for the GUPS benchmark. This implies that increasing the queue allowed for more memory requests to arrive at the memory controller, and potentially be combined due to any locality of the memory requests. The increased queue depth helped efficient scheduling of multiple requests that may be related in the STREAM case. The STREAM memory mapping was assigned to be interleaved with an 8B offset, which allowed for the parallelism of the architecture to help efficient servicing of the memory requests. There was minimal change with the GUPS benchmark due to limited temporal locality, with the cache miss rate being close to 100%.

For my next study, I observed the impact of the number of memory controllers on the performance. To do this, I increased the number of memory controllers to be twice as the original architecture, again using mesh topology. Figure 6-6 shows the impact on the performance. The two additional sets of data are for the ReRAM simulations with the number of memory controllers being 72, while the previous simulation used 36 memory controllers. The impact of the increased can be seen most drastically in the GUPS simulation for both the slow and fast write ReRAM memories.

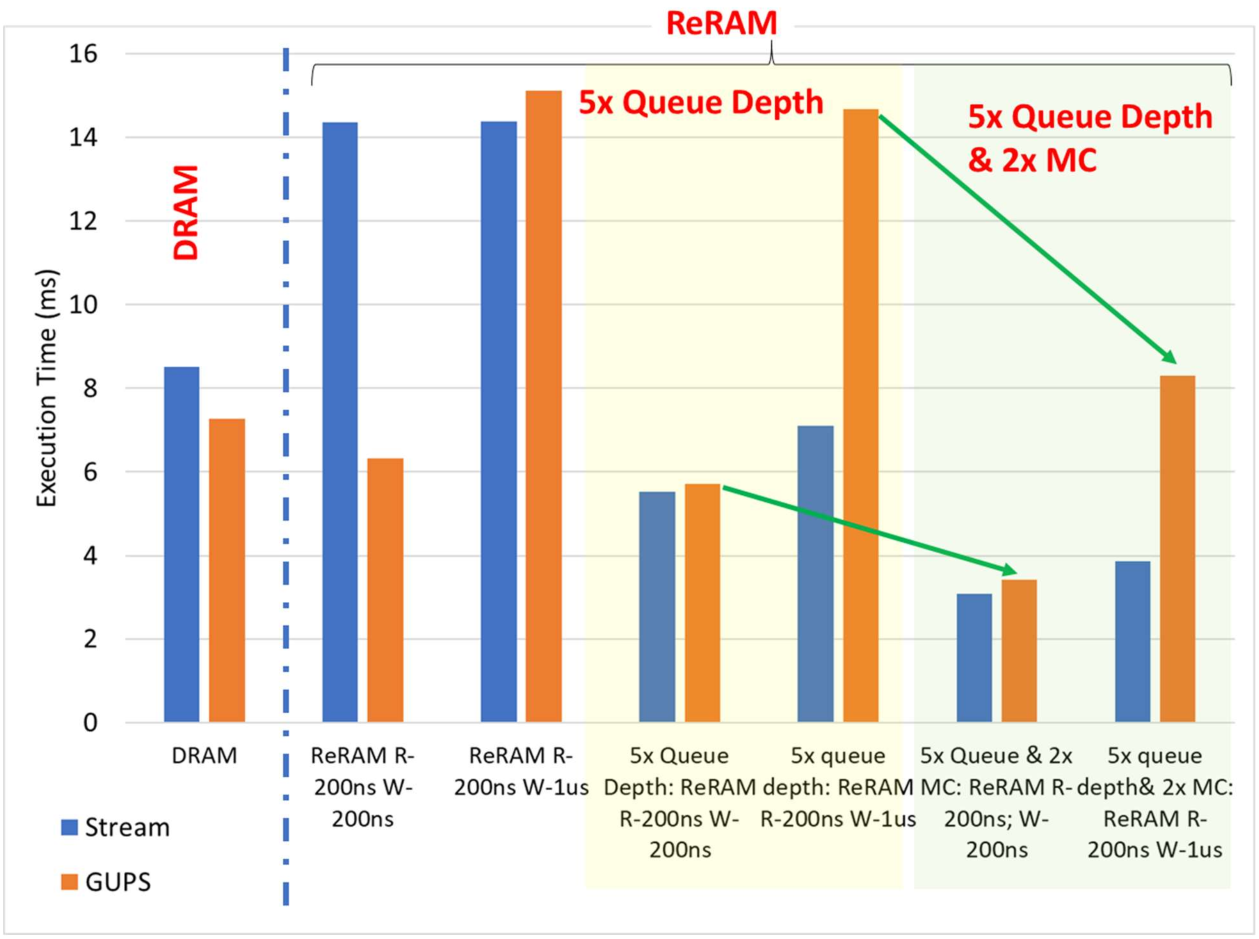


Figure 6-6 Impact of Queue Depth and Multiple Mem-Controllers

For the STREAM benchmark, queue depth helped improve the performance by efficient scheduling of multiple requests that may be related. Additionally, since the memory address mapping was interleaved across banks with an 8-Byte offset, the increased queue depth allowed for parallel processing of memory requests. The cache miss rate for the STREAM benchmark was noted to be 37%. For the GUPS benchmark, there was minimal change to the performance improvement due to the increased queue depth. This can be attributed to little spatial locality, with a near 100% cache miss rate.

Increasing the number of memory controllers improved both STREAM and GUPS, with a larger improvement for GUPS benchmark. This implies that the higher queue depth within a memory controller is beneficial for STREAM benchmarks to allow for more efficient grouping of memory requests to take advantage of spatial locality. For sparse memory access benchmarks, such as in the case of the GUPS, independent parallel memory controllers are needed to allow for parallel servicing of memory requests.

6.4 Motivation for Central ReRAM Design

At the architectural level, SST simulations were used to help answer the question of what performance benefits can be gained at the expense of non-volatility or data-retention. I utilized SST to model non-symmetric heterogeneous NoC architectures to support the monolithic ReRAM-CPU architecture. Based on my previous simulation results, I believe that a hybrid memory system utilizing both DRAM and ReRAM would be beneficial to deliver the advantages relevant for each technology based on the benchmark and application need. Additionally, this approach allows different processor type to be integrated into the same chip, including GPUs and/or accelerators. Figure 6-7 shows the floorplan of such a system.

From my area studies, I know that interspersing ReRAM peripheral logic within a core incurs a significant area penalty. Furthermore, since each core does not have a dedicated ReRAM tile, and because graph algorithms do have irregular sparse access patterns, the memory architecture needs to support requests from any processor on the chip. The centrally located ReRAM block in Figure 6-7 is designed to act as a single embedded memory IP with a separate internal NoC based on the torus topology. In addition to the NoC router circuits, the area underneath the ReRAM array could be used to store cache memory that can act as the last-level-cache for the memory. Four DRAM memory controllers are placed in the corner to allow access to an external DRAM memory off-chip. This hybrid memory system would allow for ReRAM to function as the Main-Memory and rely on DRAM as either a Last-Level-Cache (LLC) or as a selective cache for write-intensive applications only.

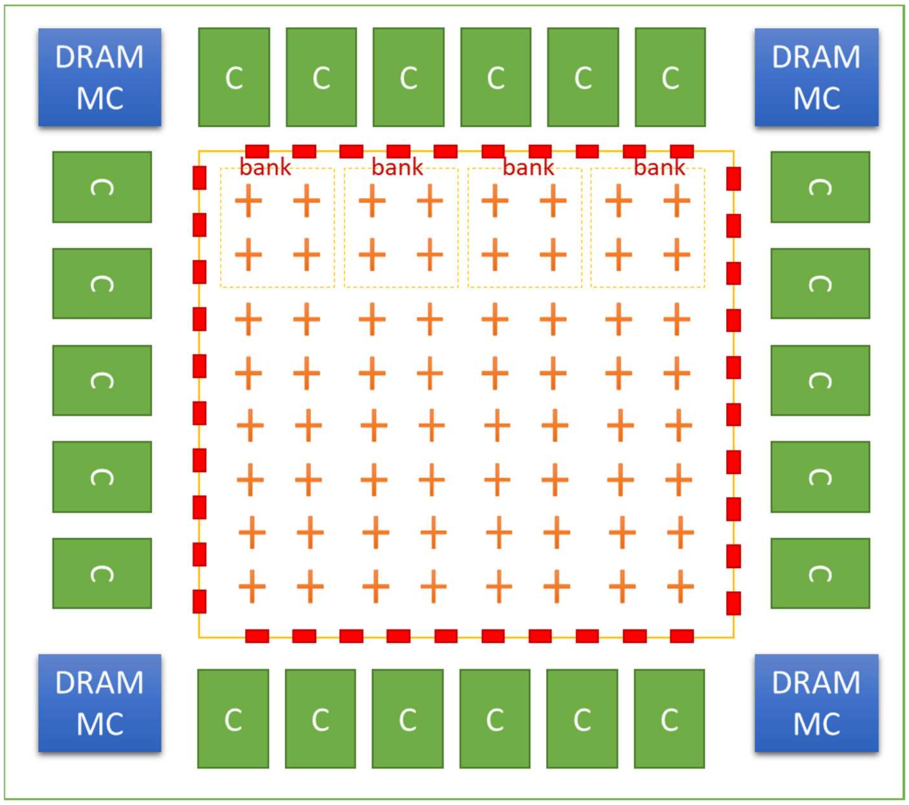


Figure 6-7 Hybrid ReRAM-DRAM System Floorplan

The ReRAM memory controller would coordinate access to n number of banks, where n needs to be selected to tradeoff between fine-grain granularity and reducing area overhead. The bank controller will also be capable of supporting Streaming Mode to perform a burst-mode from adjacent 8 banks to match DRAM granularity and improve streaming bandwidth. Figure 6-8 shows the design for the ReRAM memory controller, which coordinates multiple banks. With each bank, a bank controller will contain an incoming request queue, a data-buffer to store the read and write data (64-bits), and the circuit to initiate the Read/Write kickoff signal to all 16 arrays.

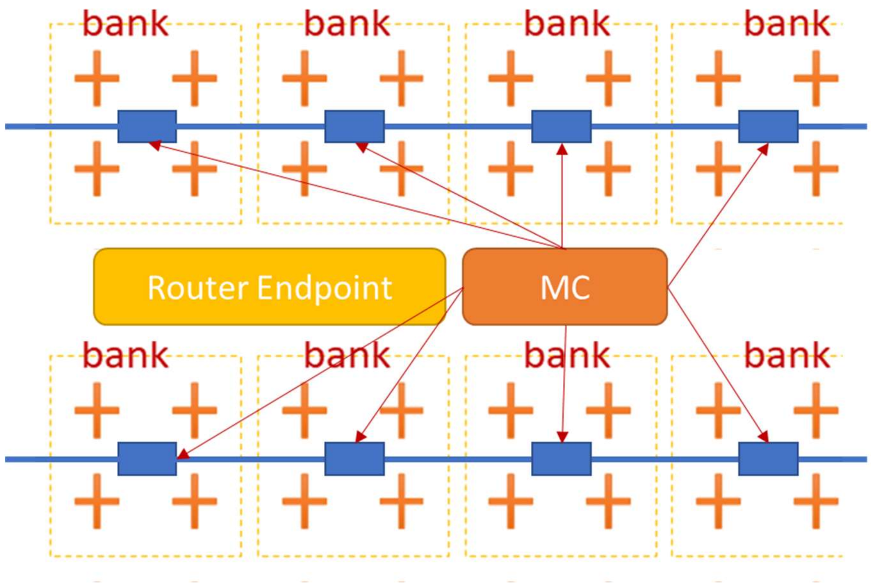


Figure 6-8 ReRAM Memory Controller Design

I performed architectural simulation using SST to model the system floorplan shown in Figure 6-7, and to select the optimal ratio and grouping. My performance results, presented in the next section, indicate the impact of the write-timing on this architecture. Additionally, based on the size of the network needed to facilitate this approach, I also looked into alternative NoC topologies that might better meet the throughput required. The next chapter goes over the NoC topology study results. Once an optimal configuration is selected, I could generate the overall ReRAM embedded block design & external interface block. This block can be used to generate a floorplan layout and provide area estimates to identify placement of individual components to achieve such a system.

6.5 Area Floorplan Central ReRAM Design

I performed a next level estimate for the bank and memory controller circuits that would reside beneath the ReRAM. I used the 45nm layout, shown in Figure 6-9, to estimate I obtained previously for my repeating block. The total layout area shown in the figure is 625um x 625um = 400,000 sq um for the 4MB 2-level stack. The vscale_core circuit had a standard-cell efficiency in this space of 60% for a total consumed area of 240,000 sq um.

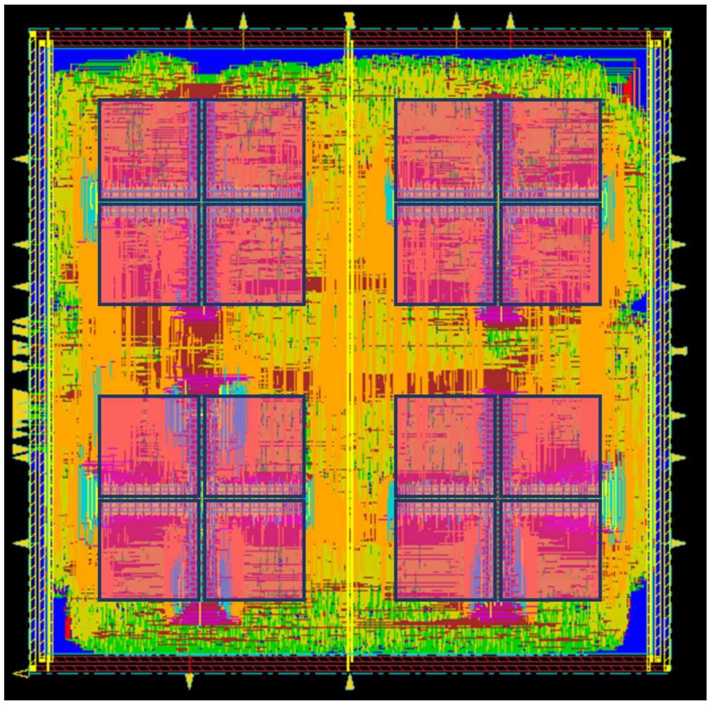
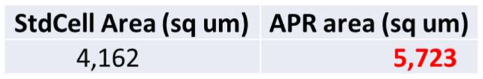


Figure 6-9 – Memory Footprint for Central ReRAM Design

I propose placing the following circuits underneath the ReRAM block: Bank Controller, Memory Controller, NoC router, and SRAM cache. To estimate the areas for the bank and memory controller, I synthesized a representative Verilog file to model the functions and used it to estimate the APR area. I extrapolate this by using the area reported from the VSCALE_CORE layout study where the standard cell area was 22,088 sq um, and the APR area was 30,373 sq um.

The bank controller logic has three main functions, as shown in Figure 6-10: an incoming request queue, a circuit to initiate the read and write kickoff signals to all 16 arrays, and a data buffer to store the read and write data. I model a 32B register file for the incoming request to support eight 32-bit command requests. A 64B register file is used to model the data-buffer to store pending read and write data. The synthesized area for this circuit was 4,162 sq um, which translates to 5,723 sq um after the APR step. As a square block, this circuit could be expected to take up 76um x 76um of area underneath the ReRAM array.



StdCell Area (sq um)	APR area (sq um)
4,162	**5,723**

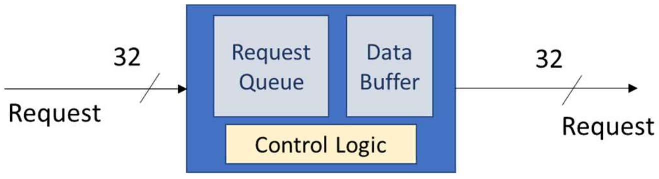


Figure 6-10 – Bank Controller Area

The memory controller logic coordinates 8 different banks and has the following functions:

- Address decode to select one of 8 banks, with additional control logic to select multiple in stream-mode
- Incoming request queue of 32B to support eight requests of 32-bit commands
- Control logic to reorder pending requests
- Data buffer of 256B to store read and write data.

The synthesized netlist for this logic reported a total area of 16,147 sq um, which I extrapolate to be 22,204 sq um after the APR step. Since this logic block will be shared among 8 banks, this circuit could be expected to use a square footprint of 18um x 18um for each bank.

The Bank, Memory, and NoC controller will be placed in a central area of the ReRAM mat, which has an area of 343x343um. The spacing between the arrays within a bank is 125um. Figure 6-11 shows the relative sizes and placement of the blocks, with the bank controller (B) being 75x75um, and the memory controller (M) being 18x18um, which is shared among 8 banks. The remaining area in the block can be used for the NoC router.

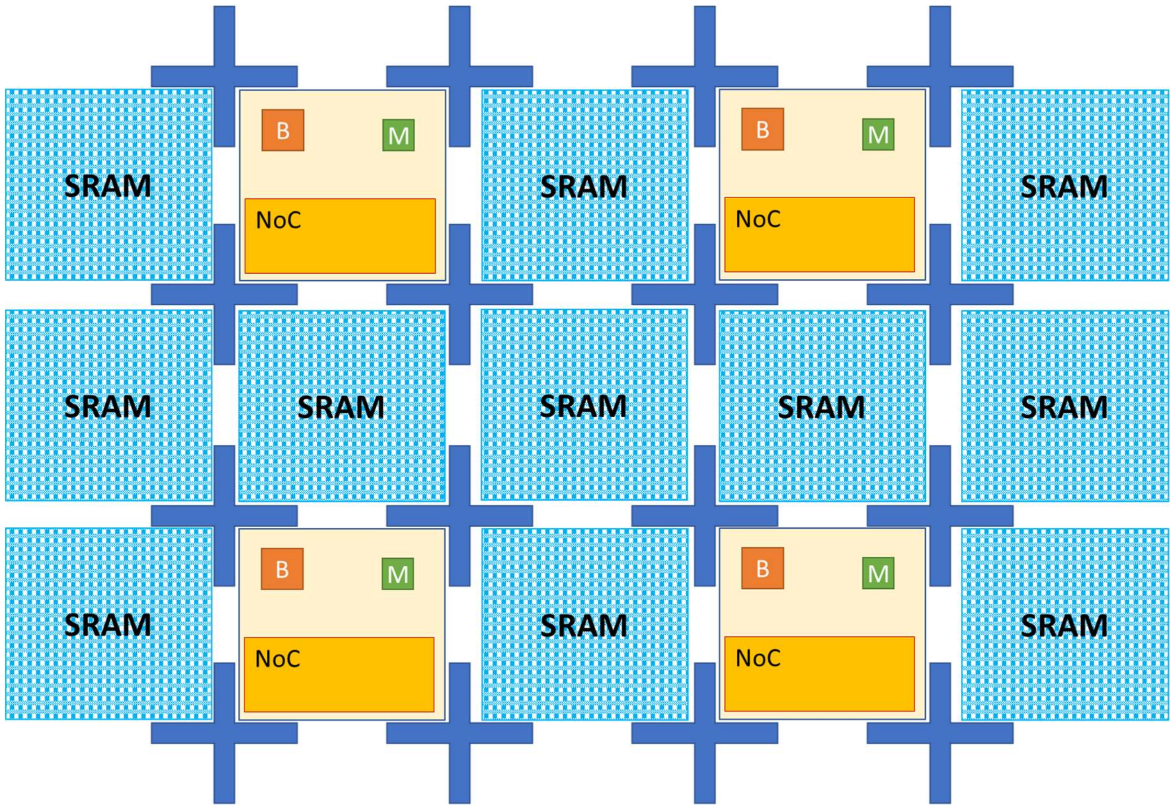


Figure 6-11 – Placement of Control Logic, Buffers, and SRAM

SRAM arrays will surround the central area with size of 125umx125um. These SRAM arrays can be used as the last-level cache on the chip and can operate independently from the main-memory ReRAM control logic. In one bank, I can fit 3 of these SRAM arrays. Assuming 75% array efficiency and using the academic OpenRAM bitcell which has a size of 1.344um x .707um, the total SRAM storage per array is 4.1kB. A commercial version of the SRAM bitcell could foreseeably be drawn 2.5x smaller, and thus achieve 10kB of SRAM capacity per array.

Finally, I consider the routing channel to connect the main-memory and the independently operating SRAM cache memories to the NoC router endpoints. The NoC interconnects can be drawn in metal-7 and metal-8 which are available to be used in the regions between the ReRAM arrays and over the SRAM arrays. This is illustrated in Figure 6-12 below. The horizontal tracks are metal-7 and the vertical tracks are in metal-8 and these would provide a global interconnect channel to the ReRAM and SRAM arrays.

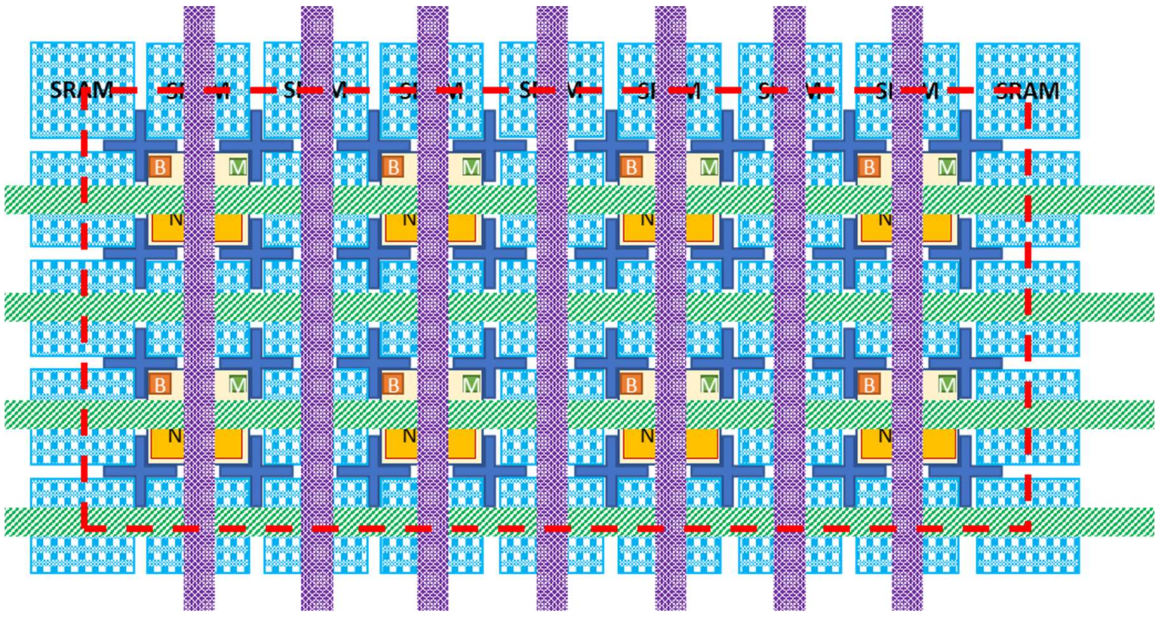


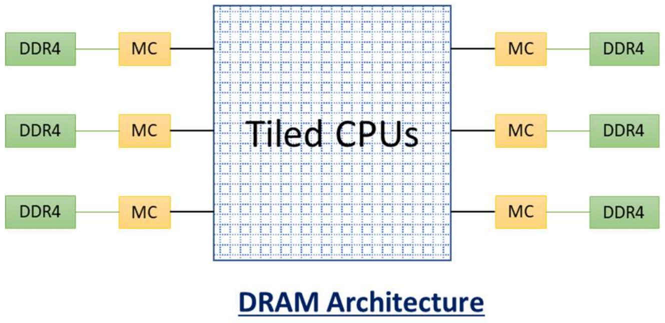
Figure 6-12 – Interconnect Routing over Central ReRAM Floorplan

For the NoC routing channel, I propose a total target of 32B interconnect width and allocate 16B for Main-Mem and 16B for the SRAM Cache in order to keep the two memory systems separate and to allow for different priorities and address schemes to be implemented between them. The available spacing for this in the floorplan above is 125um wide. This requires a metal pitch of .5um in the metal-7 and metal-8 for this routing which should be achievable in this technology.

6.6 Write Performance Impact of ReRAM

For the next phase of my simulation efforts, I focused on the Central ReRAM architecture surrounded by several CPU modules. As mentioned earlier, such a central architecture has several possible advantages over a tiled-CPU network in cases where the memory access pattern is not localized to the tiles immediately above it. In my prior benchmark simulation results, I found this to be true. Additionally, separating the CPU modules allows for a contiguous area for the design implementation and avoids having to incur the area penalty I had observed.

I first performed a high-level comparison using the SST simulation framework to compare the DRAM and ReRAM based architectures shown in Figure 6-13. The DRAM figure shows a central tiled CPU architecture with 6 memory controller (MC) access points, 3 on each side, to connect to external DDR4 devices. The ReRAM architecture shows centrally located ReRAM arrays, grouped by banks, which are accessed by surrounding CPU processors (labeled as C).



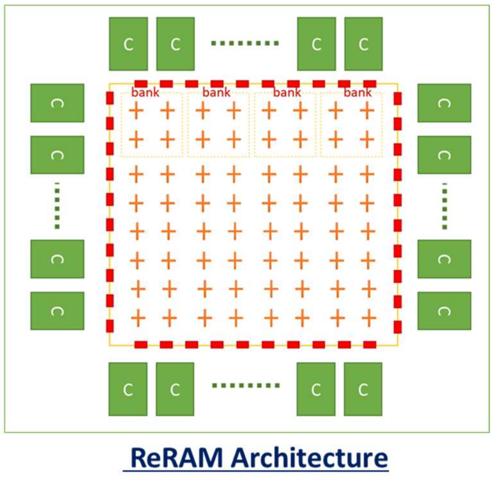
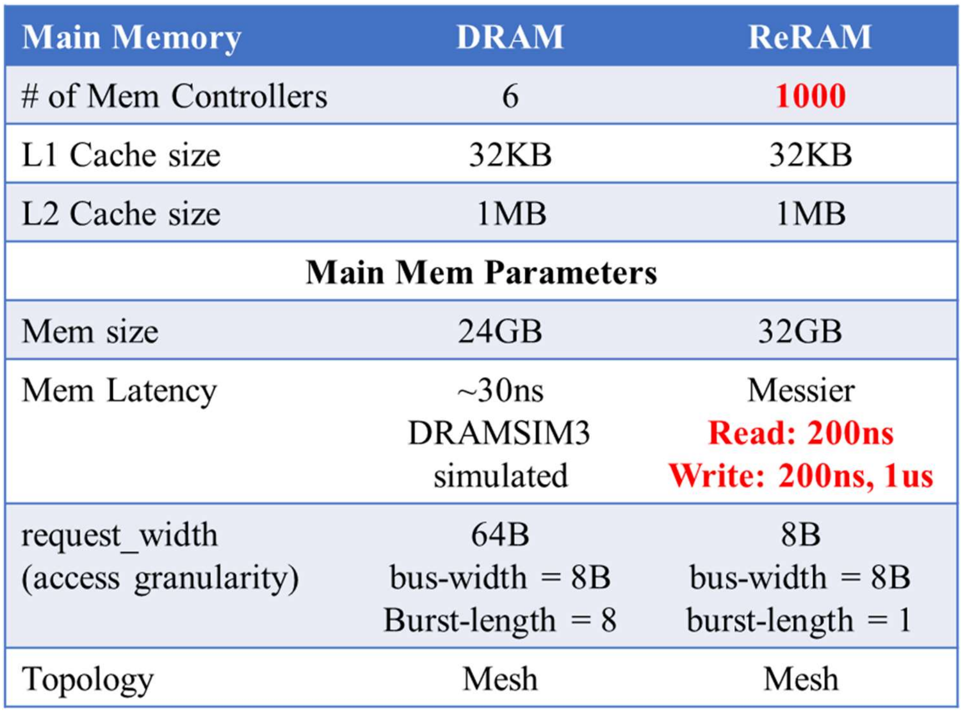


Figure 6-13 - DRAM ReRAM Architecture Comparison

For my simulation, I assumed the following system specifications. If one bank needs to provide 64-bits request width, and assuming that ReRAM is capable of a per-array bandwidth of 4 bits, then for a single bank, I would need to access 16 arrays in ganged mode. In order to provide a sustainable BW of 16B/ns, assuming 200ns ReRAM latency, I would need to group 400 banks per core. Therefore, a single core needs to coordinate with 400 banks for reasonable bandwidth performance. Tying in my previous area calculations, a single bank of 4MB (assuming a 2-layer ReRAM stack) at the 16nm is estimated to take up 0.4mm^2 in area. A full-chip die area of 686mm^2 can fit 8575 banks, assuming 50% array efficiency. This can support 8575/400 = 21 cores for a VLIW type of architecture with 8B granularity. This translates to a full-chip memory capacity of 8575 banks * 4 MB/bank = 32 GB. With an 8-level stack, this capacity scales to 128GB. As the vertical stack increases the capacity scales, but would require higher ReRAM peripheral area usage, leaving less amount of unused space underneath the memory. For my simulation of the ReRAM's mesh topology, I used a ratio of 8 banks per memory controller to provide a total number of 1000 memory controllers on chip. The system ratio of Core to Memory-Controllers to Banks to Array is 21:1000:400:16.

The DRAM architecture was based on the Intel Knights Landing platform [23] and a comparative architecture using ReRAM instead of DRAM was used. The characteristics are listed in Table 6-2. The CPU model used 8 issues per core per cycle and the mesh NoC topology is used. I used the hardware-verified DRAMSIM3 simulator to model a dual-rank DDR4-2666 DRAM device operating at 2.66GHz and also a High-Bandwidth-Memory-2 (HBM2) version of DRAM main memory. For ReRAM, I assumed a centrally located memory IP with 1000 access memory controllers with the support circuits and bank-select logic located underneath the memory, while the CPUs surround the array.



Main Memory	DRAM	ReRAM
# of Mem Controllers	6	**1000**
L1 Cache size	32KB	32KB
L2 Cache size	1MB	1MB
Main Mem Parameters		
Mem size	24GB	32GB
Mem Latency	~30ns DRAMSIM3 simulated	Messier **Read: 200ns** **Write: 200ns, 1us**

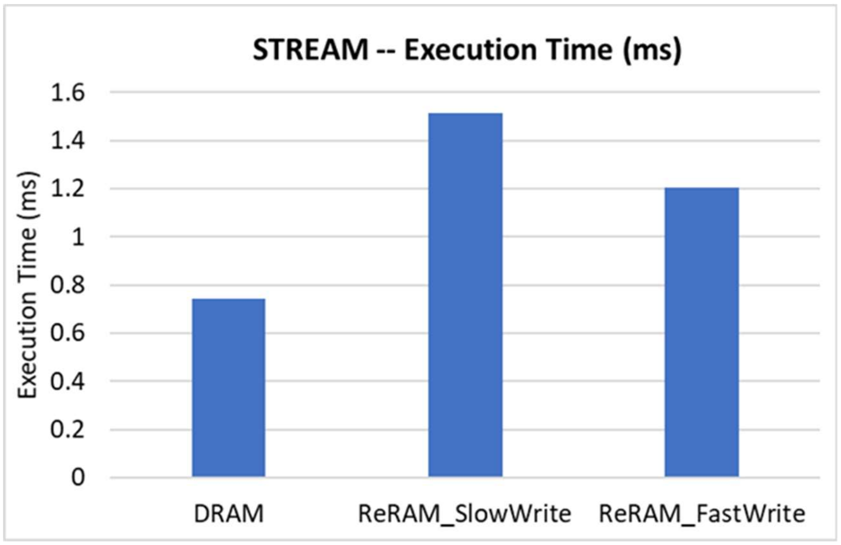
request_width
(access granularity)
	64B
	bus-width = 8B
	Burst-length = 8
		8B
		bus-width = 8B
		burst-length = 1

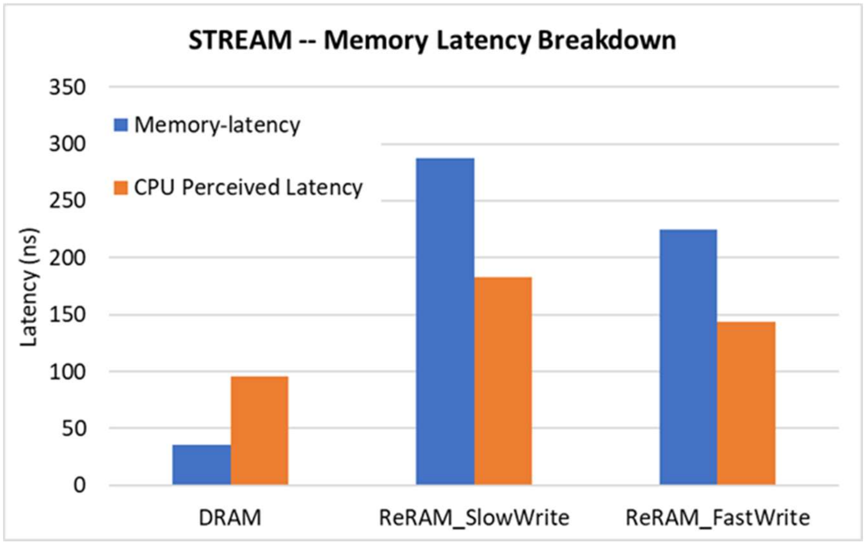
Topology	Mesh	Mesh

Table 6-2 - Architectural Parameters

In order to understand the impact of the longer write latency of ReRAM, I compared DRAM with two versions of ReRAM: SlowWrite and FastWrite. For the ReRAM SlowWrite version, I used a write-latency of 1us, while for the FastWrite version, I used 200ns. The read latency was set to 200ns for both versions of the ReRAM. Figure 6-14 summarizes the result of the architectural simulation for the STREAM benchmark. The y-axis in the top plot shows the overall execution time in ms, with DRAM over 2x faster than the ReRAM-slowWrite option. The ReRAM FastWrite with 200ns latency is slightly faster, but still performs worse than the DRAM configuration.

The memory latency comparison, however, shows a much higher delay difference between the two architectures. The DRAM memory latency on average is 32ns, which is over 8x faster than ReRAM_SlowWrite. However, the CPU perceived latency is only 2x faster, despite this large difference. The MSHR occupancy comparison shows the reason for the discrepancy, with DRAM having a much higher occupancy resulting in a greater number of bottlenecks at the memory controller and stalls from the CPU point of view.





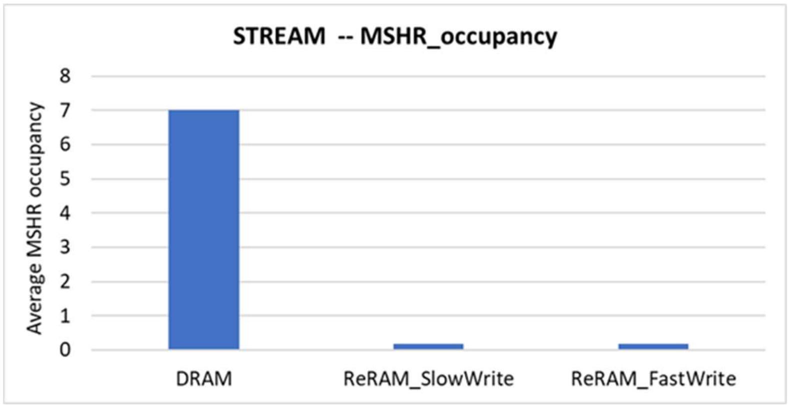
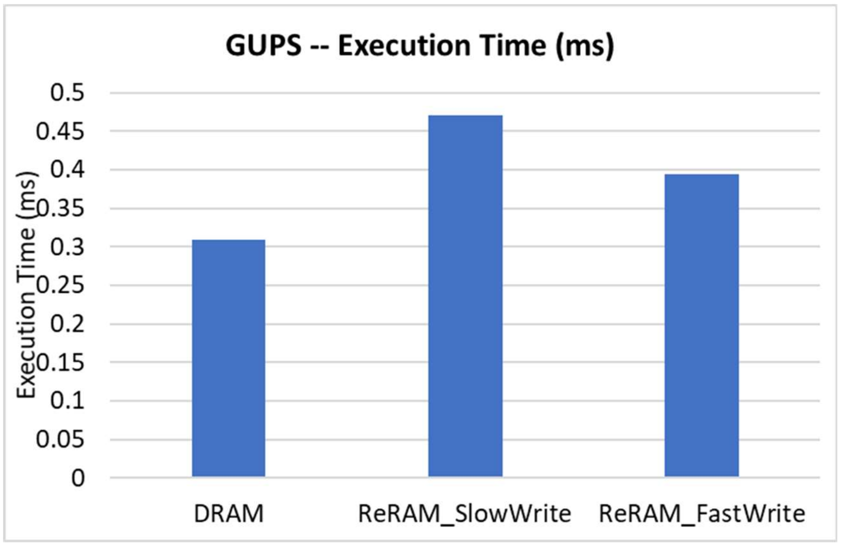


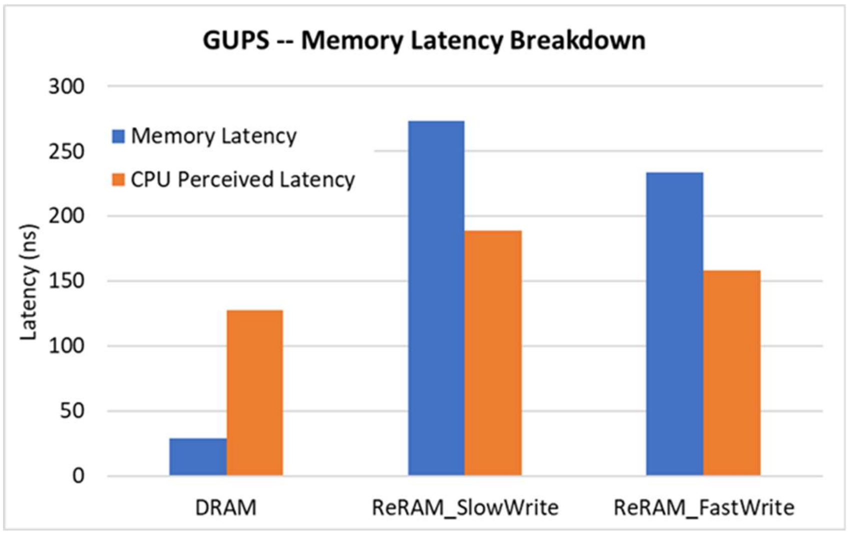
Figure 6-14 - SST STREAM Benchmark Comparison for 21 cores

Figure 6-15 shows the results for the GUPS benchmark, which has a finer access granularity of 8B. Again, the results show that overall DRAM much longer RunTime, by 1.5x when compared to the ReRAM_SlowWrite case. At the memory level, DRAM latency is faster, but from CPU point of view, overall perceived latency is slower due to a bottleneck at memory controller, which is shown in the MSHR_occupancy comparison.

For the STREAM benchmark, DRAM is faster overall by 2x, and by 1.5x for the GUPS benchmark. Though there are more pending requests due to the limited number of memory controllers with the DRAM architecture, the higher latency with ReRAM results in an overall longer latency time. This observed trend was consistent for both STREAM and GUPS benchmarks at the 21-core level.

Next, I increased the number of cores to 68 cores, which is the number used in the Intel KNL chip. I simulated the comparison with both 21 cores and 68 cores, and the results are shown in Figure 9-4. The results indicate that when the number of cores is low (21), DRAM-based architecture outperforms ReRAM, even for GUPS type of algorithms. Although there was a small performance improvement with the ReRAM_FastWrite version, this still was not enough to overcome DRAM architecture performance.





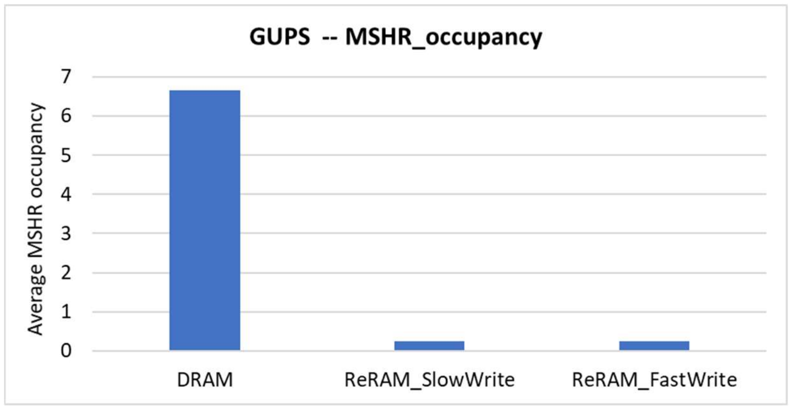


Figure 6-15 - SST GUPS Benchmark Comparison for 21 cores

However, when the number of cores was increased from 21 to 68 cores, I see that in both STREAM and GUPS based benchmarks, ReRAM is able to outperform DRAM-based architecture. This is due to the higher amount of memory access requests needed with the higher core count. This requirement is more easily met by a more parallel memory system such as the one architected with the ReRAM based main memory. I see this reflected in the bottom plot in the figure of the memory latency breakdown for the stream benchmark. Comparing the impact of increasing core count on the CPU perceived latency, I see a sharp increase for DRAM, while minimal impact to the ReRAM scenarios. This increase in latency can be attributed to a higher amount of bottleneck resulting in more stalls. Therefore, as the core count is increased, there needs to be enough parallel request to fully exploit the high amount of parallelism afforded by ReRAM and overcome the higher latency with ReRAM.

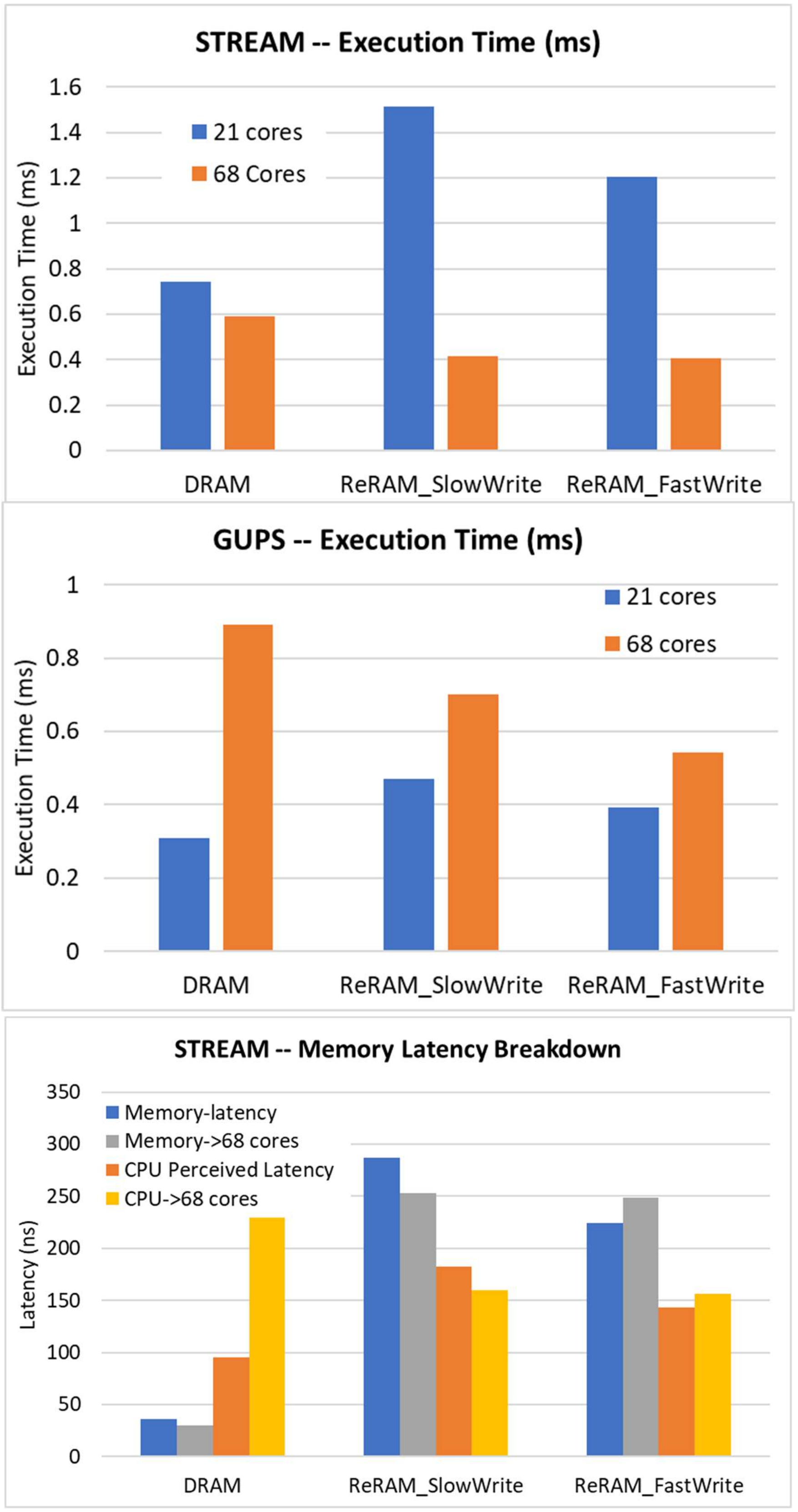


Figure 6-16 – Impact of Increasing Core Count

Table 6-3 compares the memory bandwidth processed in each of the simulated conditions. At lower core count, DRAM based architecture provides STREAM bandwidth of 76GB/s is nearly 40% higher than the one provided through ReRAM based architecture. At higher core count, ReRAM provides a higher bandwidth of 138GB/s, while the DRAM-based architecture's STREAM bandwidth is 30% lower at 95GB/s.

Bandwidth (GB/s)	**Cores**	**DRAM**	**ReRAM_SlowWrite**	**ReRAM_FastWrite**
GUPS	21	1.36	0.89	1.07
	68	1.53	1.94	2.51
STREAM	21	76	37.45	47.07
	68	95.6	136.63	138.6

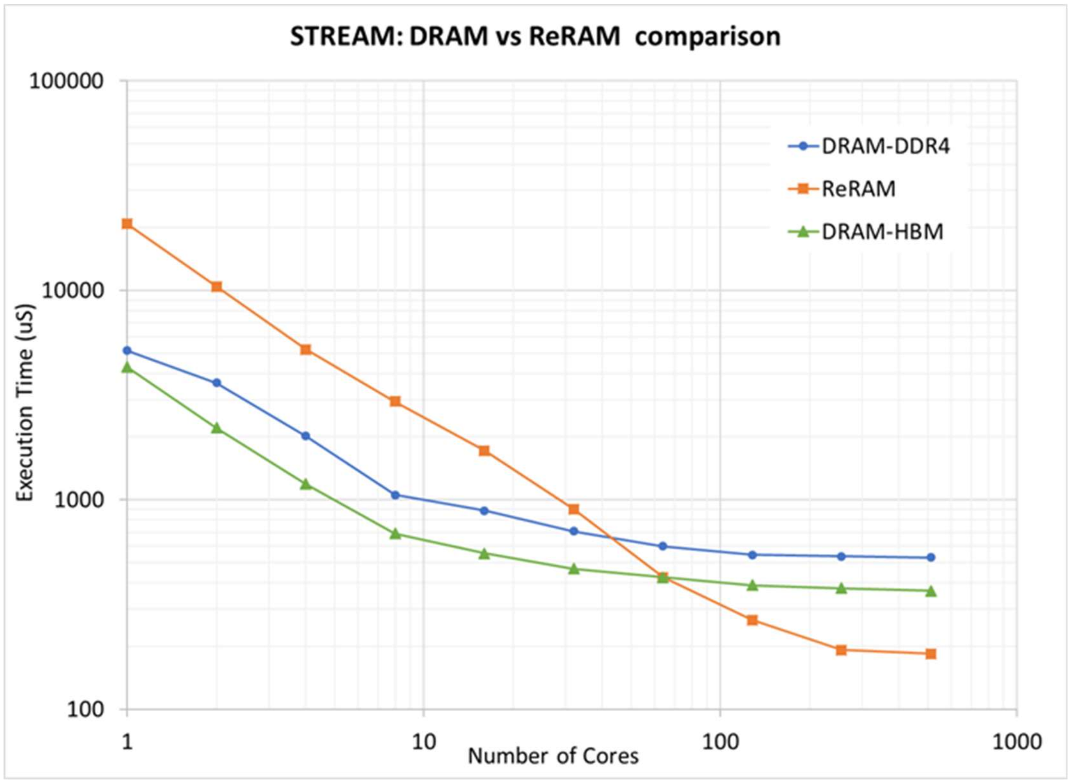
Table 6-3 Bandwidth Comparison

6.7 Impact of Core Count

Based on the previous section, I see that the advantages of the Monolithic ReRAM architecture's parallelism can only be exploited when there are sufficient number of accesses, realized at higher core counts. To study the impact of the core count, I performed a set of simulations varying the core count and compared the performance on a ReRAM architecture with a write-latency of 1us. In addition to the

DDR4 version for the DRAM architecture, I also used an HBM2 version for the DRAM model. Figure 6-17 shows the SST simulation result of the comparison as a function of the core count, from $2^0=1$ up to $2^9=512$. The x-axis is the core-count and the y-axis is the execution time.

If I look at the STREAM benchmark result for DRAM-DDR4, as the core count increases, the execution time reduces at a constant slope, implying an improvement in performance gained through the higher processing power. However, this trend saturates at around 8 cores, beyond which the simulation time improves at a slower rate. A similar trend exists for the DRAM-HBM implementation as well, with the execution time being lower, but having a similar inflection point which is on the order of the number of memory controllers for the DRAM implementation. For the ReRAM implementation, the execution time much higher due to the higher memory latency but falls at a similar rate as the DRAM implementations. The difference, however, is that ReRAM's inflection point is much higher, above 250-core count.



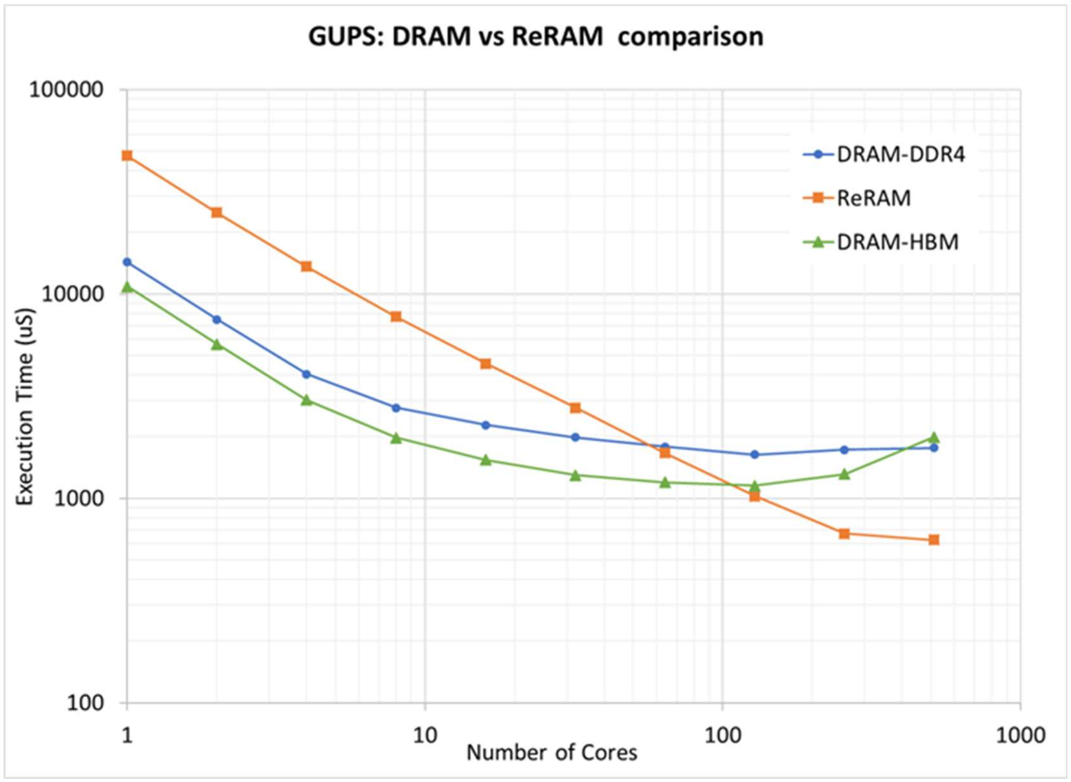
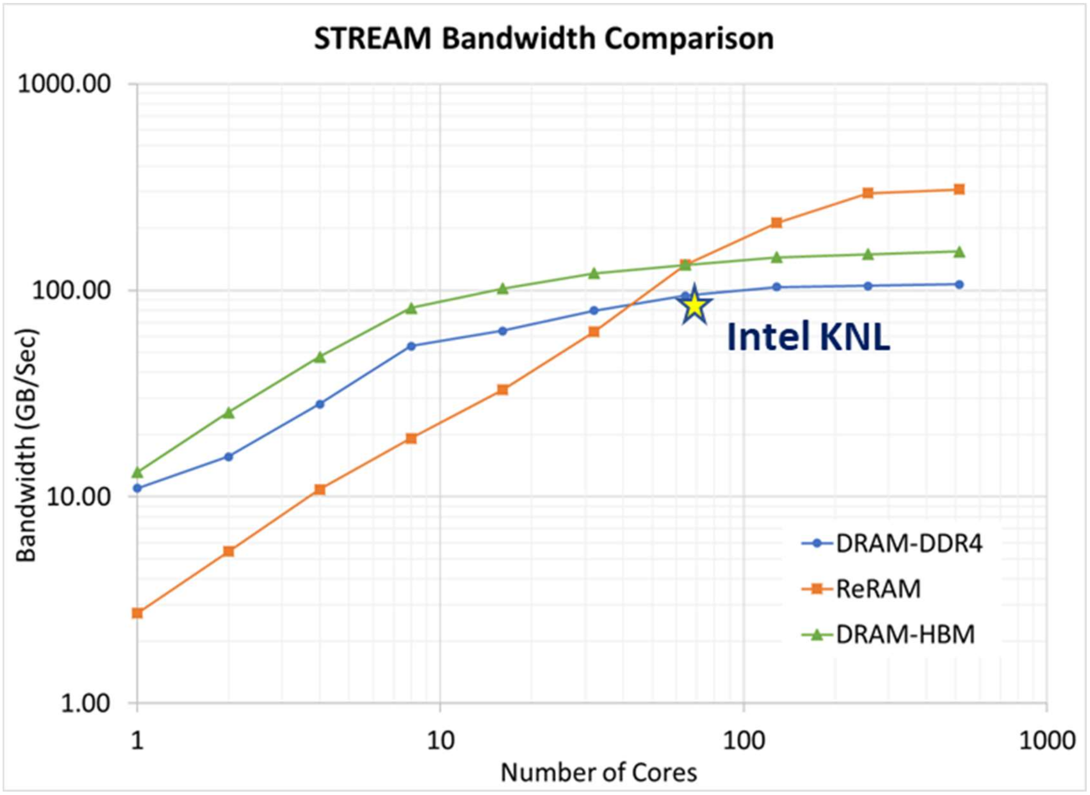


Figure 6-17 - Performance Comparison between DRAM and ReRAM system using STREAM and GUPS benchmarks (note: Log-Scale X & Y axis)

Both DRAM-DDR4 and DRAM-HBM outperform ReRAM in the low core count for both benchmarks. HBM is able to perform slightly better than DRAM due to its higher bandwidth and proximity with the CPU. However, starting at 64 cores, ReRAM begins to outperform both DRAM devices with a low execution time. Figure 6-18 summarizes the bandwidth comparison between ReRAM, DRAM-DDR4, and DRAM-HBM2 architectures. In the STREAM bandwidth plot, the star represents the reported 90+ GB/s number from Intel Knights Landing, which corroborates with my simulation results.



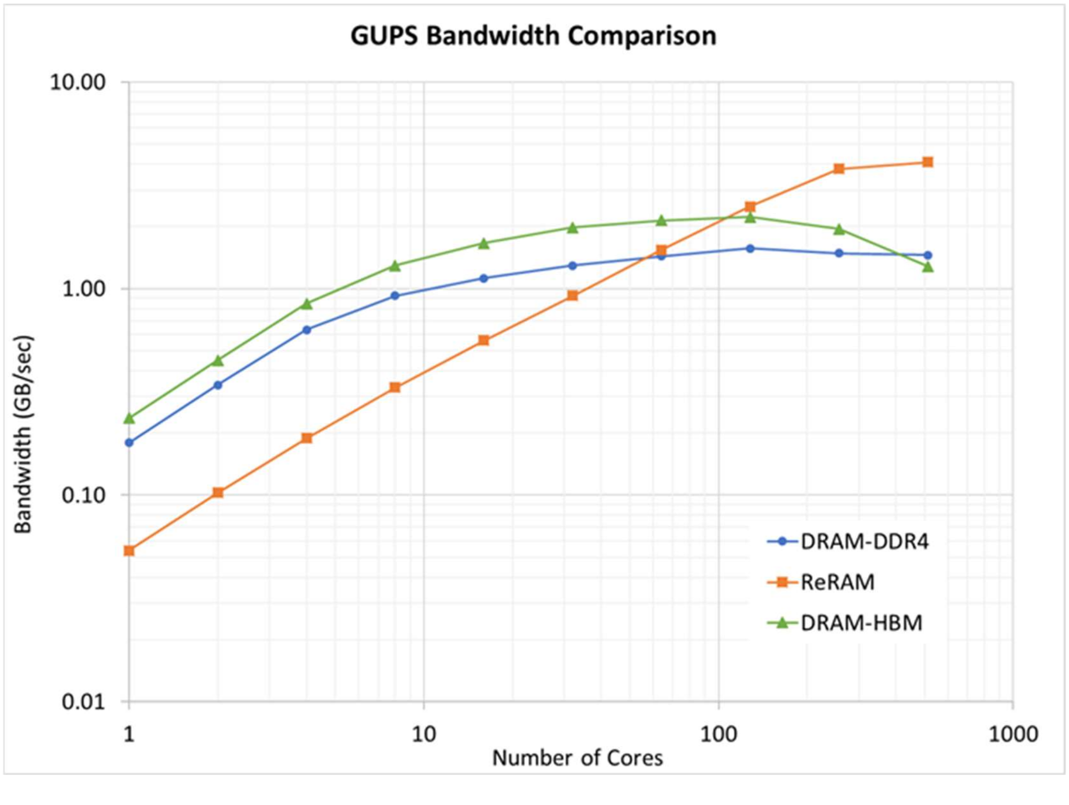


Figure 6-18 - Bandwidth Comparison between DRAM and ReRAM system using STREAM and GUPS benchmarks (note: Log-Scale X & Y axis)

At the inflection point of 64 cores, ReRAM outperforms DRAM-DDR4 by 30% for the STREAM benchmark case and meets the performance of HBM2-based architecture. The results indicate that at very low core counts (less than 64), DRAM outperforms ReRAM due to its much lower inherent access latency for both benchmarks. However, as core count increases, DRAM cannot keep up with the data bandwidth needs, while ReRAM's parallelism compensates for its higher memory latency. This can be further illustrated when I analyze the read latency contribution from the different system components. At the memory level, DRAM latency is 8x faster than ReRAM. Yet, at the CPU level, the overall perceived latency for DRAM is only 2x faster at a core count of 16. This manifests in the DRAM architecture as bottleneck of the memory requests at the miss status holding register (MSHR), the hardware structure for tracking outstanding misses. For ReRAM, due to the high amount of parallelism, the memory requests are processed without having to hold them. At the higher core counts, there is a sufficient amount of access requests to take advantage of the memory parallelism offered by the ReRAM architecture.

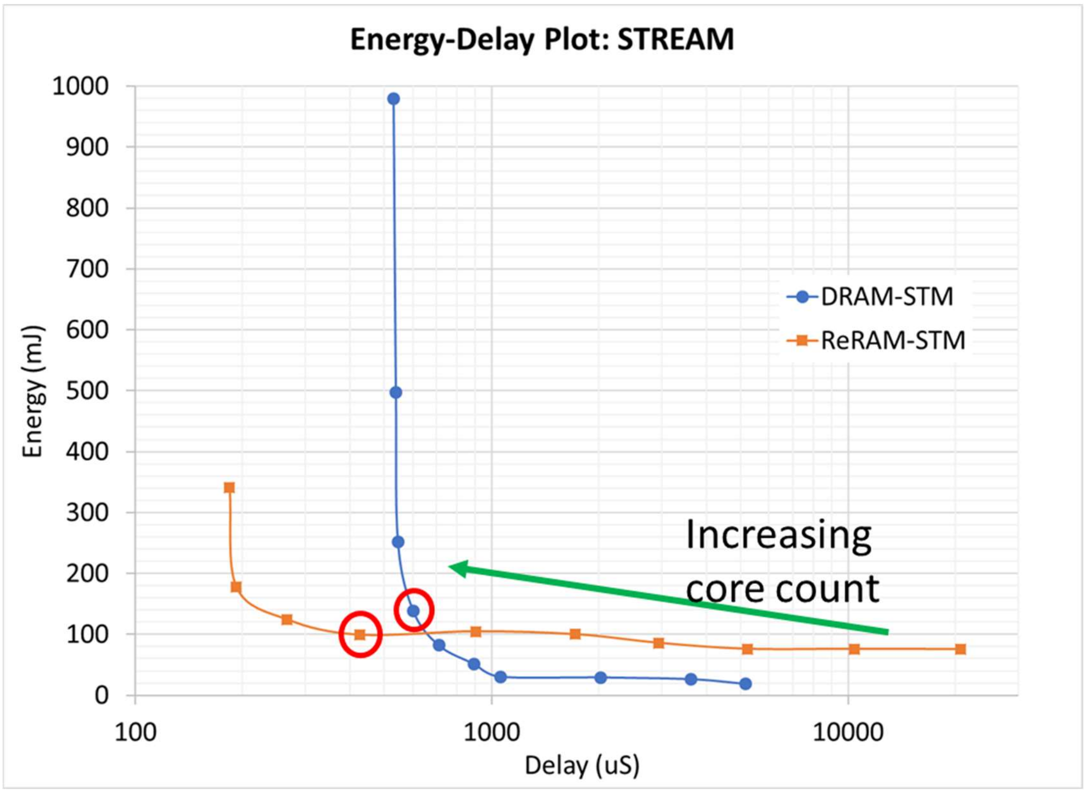
The results confirm that with sufficient processing power, the highly parallel ReRAM with long latencies performs better than high-speed DRAM with limited memory controllers. The cross-over point when ReRAM outperforms is 85GB/s for DRAM-DDR4 and 135GB/s for DRAM-HBM2 device with the STREAM benchmark.

One interesting note is with the slight worsening of performance with HBM DRAM at very high core count of 512. Because of HBM's higher bandwidth interface, the low-access granularity of GUPS suffers with HBM due to stalls from prior access requests.

6.8 Energy Comparison

I also calculated the total energy dissipated for the DRAM-DDR4 system for comparison against ReRAM. For the CPU and network power dissipation, I extrapolated from Intel's Knights Landing power specification. For DRAM, my simulations for the 16GB dual rank DDR4 2400MHz DIMM model reported average energy per bit dissipation of 19.5pJ/bit. For ReRAM, I used energy numbers of 64pJ/bit for write (reported in Crossbar’s whitepaper) and 0.5pJ/bit for read operations assuming a read current of 5uA, a 2V voltage bias, and a cell-sensing time of 50ns. Figure 6-19 shows the energy-delay plot for DRAM and ReRAM architectures for both benchmarks. The x-axis in the plot is the total delay to complete the simulation, while the y-axis is the energy consumed in mJ, as a product of the power (voltage * current) and duration of the power consumption. These points were obtained from the different core counts I used in the previous section. The energy-delay cross product is a metric used to assess the impact that a reduction in delay would provide in terms of energy.

For the STREAM energy-delay plot, for both DRAM and ReRAM, as the core count is increased, resulting in lower delay, I see little impact on the energy consumed. This is because, at these points, increasing the core count reduces the overall duration of power consumption, which is taken up by the higher number of core power. However, I see an inflection point, after which there is very little reduction in delay by increasing core count, but there is a much higher energy penalty. This is the knee of the curve observed, where throwing more processing power does little to provide improvement in performance. This inflection point is at higher core count, similar to the performance plot I saw in the previous section. A similar trend is seen for GUPS benchmark, with its inflection point being much higher for ReRAM, due to the advantage that ReRAM offers in terms of finer access granularity.



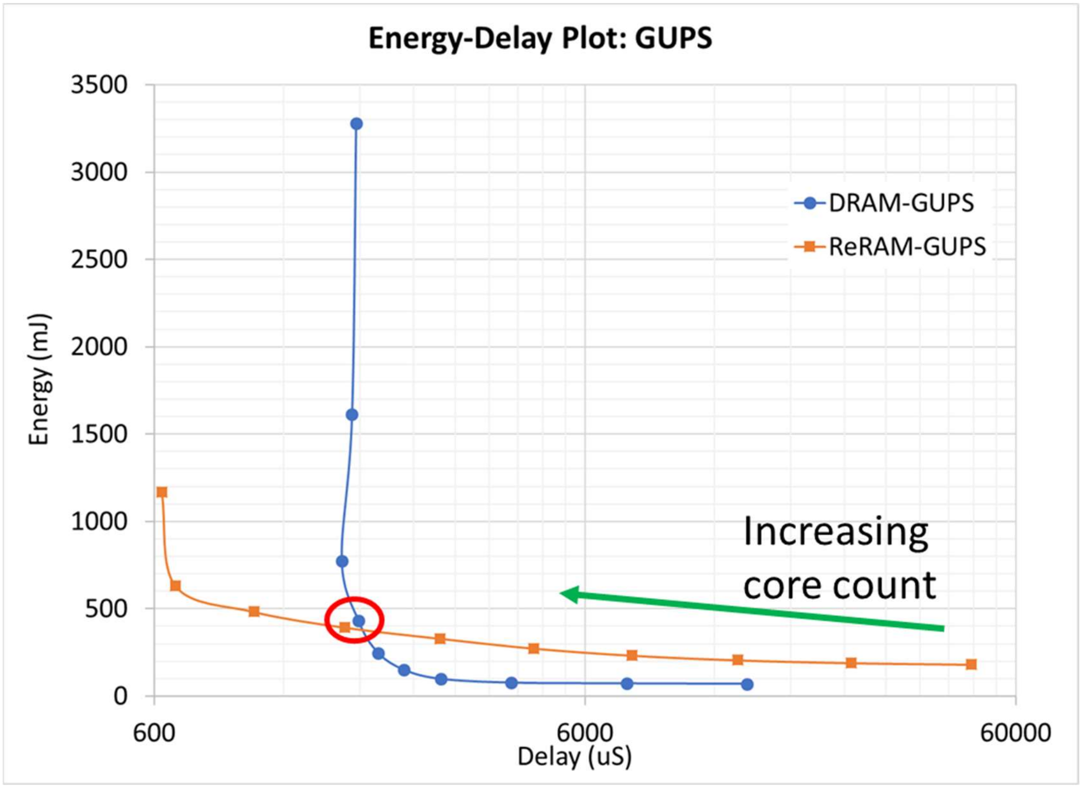


Figure 6-19- Energy-Delay Plot of DRAM-DDR4 and ReRAM system using STREAM and GUPS benchmarks (note: Log-Scale X axis)

The optimal operating point on the energy-delay tradeoff is circled on the figure, indicating that ReRAM performs at or better than DRAM at both benchmarks. I observe that, overall, ReRAM delivers lowest delay for both benchmarks, as seen in the performance comparison. This is achieved at the higher-core counts, where ReRAM as a main memory is able to provide an energy efficient, especially for GUPS where high access granularity of DRAM incurs additional penalty. This energy efficiency comes primarily as a result of faster execution time, which reduces the duration of CPU and NoC power dissipation.

7 NoC Topology Impact

7.1 Motivation

ReRAM based main memory architectures offer advantages in terms of scalability, density, and fine-access granularity. These architectures are capable of delivering high connectivity and low access granularity. To truly exploit the parallelism offered by ReRAM architectures, a robust Network-On-Chip (NoC) topology and optimum scaling of core count is critical to ensure low packet latency while being able to offer the high throughput in communication.

In this chapter, I compare different NoC topologies for a ReRAM based main-memory system and study the effect on speedup as the number of cores scales on-chip. Based on architectural simulation results from SST on streaming and GUPS benchmarks, I observe that fat-tree and torus topologies provide performance gains of 78% and 39%, respectively. I also observed that optimal core and memory controller configuration have a bigger impact at moderate to high number of cores than the topology. Performance comparison of a ReRAM-based main-memory architecture against a conventional DRAM-based architecture indicate a gain of 30% with 64 cores. Power, cost, and performance tradeoff analysis are also presented.

Figure 7-1 (a) shows a conventional CPU chip connected to on-chip DRAM High Bandwidth Memory (HBM) devices through a silicon interposer. Figure 7-1 (b), illustrates an integrated CPU processor with ReRAM main-memory on the same chip, enabling a high-number of connections between the two systems. In contrast, the conventional on-chip DRAM solutions are limited in the number of connections, through memory controller access points, to the on-chip HBM or DDR4 DRAM devices.

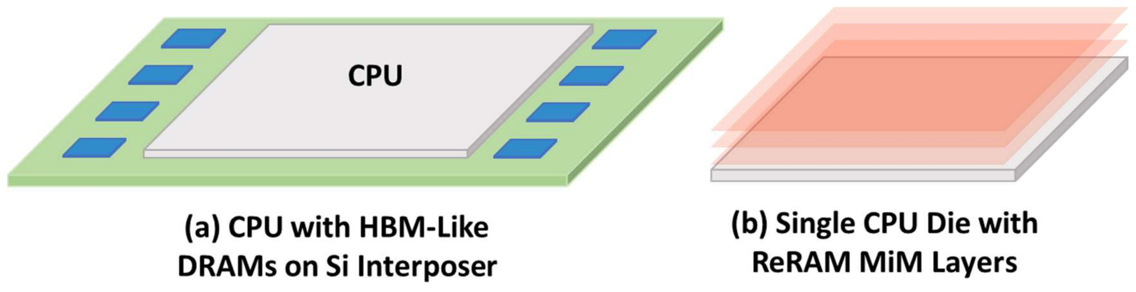

CPU
(a) CPU with HBM-Like DRAMs on Si Interposer
(b) Single CPU Die with ReRAM MiM Layers

Figure 7-1 - Comparison of (a) Conventional off-chip main-memory system with (b) Integrated CPU die with ReRAM layers on-chip

As mentioned before, to support reasonable sustained bandwidth requirements in a system, a high number of these ReRAM arrays need to be accessed in parallel. ReRAM being on-chip allows these connections to occur directly through metal-vias, rather than through an on-chip I/O port, an external interposer for HBM, or large TSVs in the case of 3D-ICs.

The resulting system requires a robust Network-on-Chip (NoC) between the many core and 1000s of memory controller points on the chip. Conventional NoC is based on ring and mesh like topologies and are typically built for 100s of access points. With our highly parallel memory-CPU memory architecture, these topologies may not be able to support the higher network throughput needed, especially for future process nodes. In this paper, I compare performance and power metrics between a DRAM and ReRAM system, look at the effect of different network-on-chip topologies on the system performance, and investigate optimal memory controller configuration for a hybrid ReRAM-DRAM memory system.

The rest of the chapter is organized as follows. Section 2 provides a short background of the NoC topologies I investigated, and the simulation methodology I followed. Section 3 presents the results and discussion of the effect of the NoC topology, and the optimal DRAM Memory controller configuration. Section 4 presents the conclusion.

7.2 Background

Based on previous work [11], a homogenous 2D-Mesh topology for the Network-On-Chip (NoC) is unlikely to keep up with the relatively high communication need of the ReRAM-architecture I envision. In order to support low latency across the chip, I performed a survey of NoC topologies that can support high network capacity for a large number of nodes on-chip. In this section, I perform a brief summary of NoC topologies of interest, and the performance metrics used to measure them. The topologies that have been physically fabricated by other research projects or in the industry are shown in Figure 7-2, implemented as NoC or as datacenter network topologies.

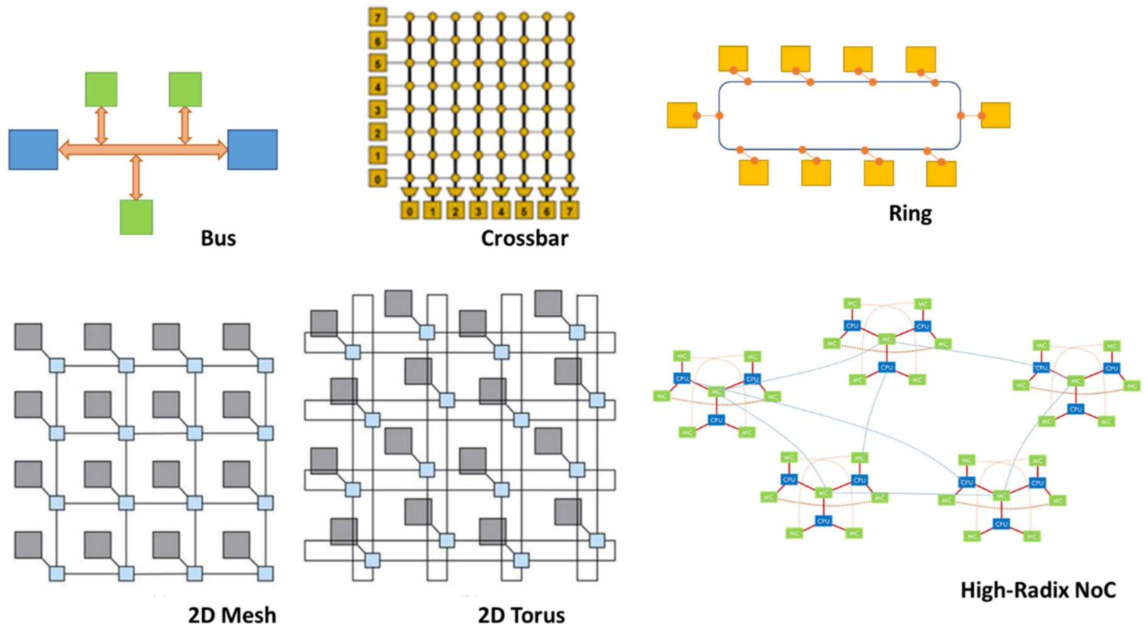


Figure 7-2 Comparison of various NoC topologies

Some of the metrics that are used to compare the network performance are node-degree, diameter, and bisection width. The node-degree of a topology denotes the number of ports connected to each node and reflects the input-output complexity of the network. A high node-degree reduces the average path-delay but increases the complexity of the implementation. The diameter is the worst-case path delay in the network and reflects the maximum shortest path between any two nodes. The bisection width is the minimum number of links that needs to be bisect, or cut, in order to divide the topology into two separate networks. This parameter is used to indicate the parallelism of the network. Ring and bus networks have a fixed bisection width.

1. **Bus**: This topology consists of a common routing channel to which multiple devices connect to communicate with each other. It allows for a simple implementation, and is the paradigm used in older system-on-chip type implementations. However, the single common bus prevents simultaneous communications between devices and requires bus arbitration policies to allocate the resource between the devices. Therefore, this type of topology is not scalable as the number of devices increase.
2. **Crossbar**: The crossbar topology allows for multiple parallel connection between different input and output permutations. The result is a topology that is low latency with higher throughput than the bus topology. The IBM Power5 architecture uses a crossbar topology. However, as the number of nodes increase the matrix expands to an additional row and a column, resulting in a high overhead. Therefore, this topology cannot support a scalable architecture.
3. **Ring**: The ring topology consists of a closed bus with the communication direction restricted to one direction. Each node has two neighbors (degree=2)

and the “first” and “last” nodes are connected to each other. The information packet travels along the ring from the source until the destination is reached. The communication scheme is simpler and requires lower area to implement. The bisection width is 2, and the diameter is n/2, where n is the number of nodes. Architectures that have used the ring topology include the IBM Cell and earlier Intel architectures, such as Knights Ferry.

4. **Mesh**: The mesh topology consists of m rows by n columns to support m*n nodes. At each intersection, a router directs the direction of the packet to take the shortest path to the destination. Higher path diversity makes multiple simultaneous packet transmission possible. The architecture is easy to layout. The bisection width is min(m, n), the diameter is (M+N-2), and the node degree is 5 for the central nodes, 4 for edge nodes and 3 for corner nodes. The Tilera 100-core CMP and the Intel Knights Landing uses this topology in their architecture.

5. **Torus**: This topology is similar to Mesh with the end points in the network being connected to each other. This has the added advantage of allowing for better fairness due to limiting maximum number of hops, while slightly increased complexity and area cost. This leads to better path diversity than mesh and improves the diameter of the network. The bisection width is 2*min(m, n), the

diameter is (m/2)+(n/2), and the node degree is 5 for all nodes. Currently, 3-dimension torus networks are used in some supercomputer networks, such as the CRAY XT3 and IBM BlueGene.

6. **Hoffman-Singleton**: The Hoffman-Singleton is a high-radix symmetric graph. It limits the number of connections between any two nodes to two hops but is more complex to implement on a 2D die. This topology is currently used in large scale datacenters, such as the high-radix CRAY XE6.

Table 7-1 summarizes the relative advantages and disadvantages of the different NoC topologies.

TOPOLOGY	ADVANTAGE	DISADVANTAGE
BUS	Simple implementation	Simultaneous communication between devices not possible
CROSSBAR	Multiple parallel connection possible	Not scalable as number of nodes increase
RING	Simpler communication scheme; lower area to implement	Slower implementation as data packet travels through all nodes
MESH	Higher path diversity; Ease of layout	Higher cost to implement than previous models
TORUS	Reduces worst-case path from Mesh implementation;	Slightly more complexity in wiring than Mesh
HOFFMAN-SINGLETON	Two hops between any two nodes	Complex communication scheme; higher cost for implementation

Table 7-1 Comparison of NoC Topologies

A NoC topology's performance is highly dependent and specific to the application and hardware architecture. I looked into modeling tools that allow us to assess performance impact of various topologies such as mesh, torus, and non-symmetric heterogenous NoC architectures which might be needed to support the Monolithic ReRAM CPU architecture. To support this, I have used the SST (Structural Simulation Toolkit) to model the different heterogeneous NoC architectures.

7.2.1 ReRAM-based Main-Memory Architecture

For ReRAM memories, the read and write latencies are considerably higher than DRAM memory latencies. For ReRAM memories, a trade-off exists between the number of bits accessed from a single word-line and the access latency. In order to limit the latency, per-array bandwidth are typically low as shown in Figure 7-3. The figure shows a ReRAM array with wordline (WL) decoders selecting a single row in the array. A column multiplexer (MUX) circuit at the bottom of the array is used to select a few of the bitcells in the selected wordline to read from. The bitline current from the selected columns is then used by a sense amplifier to differentiate between a logic high and logic low read value. This operation is done for a few cells in an array. In order to provide sufficient bandwidth, several of these mini array banks would need to be accessed in a parallel ganged mode. As shown in the figure, multiple arrays are accessed in parallel together to provide n times the per array bandwidth.

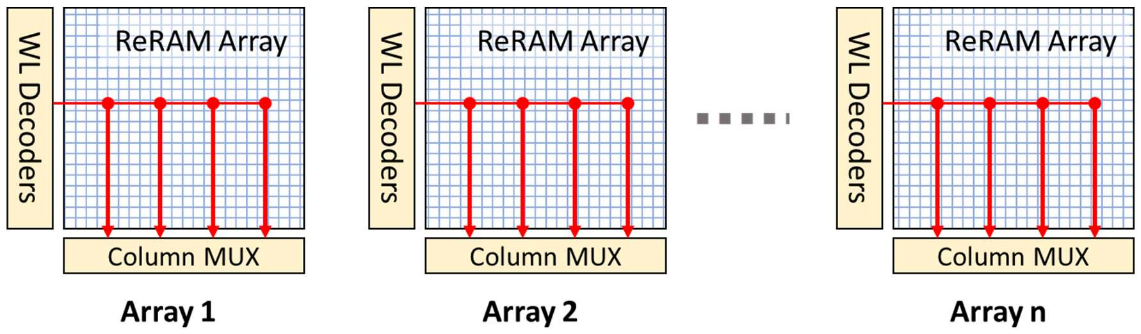


Figure 7-3 - ReRAM Array Access

At the system level, the sustained bandwidth is calculated as a function of the per-array bandwidth, the arrays per bank, the number of banks, and the access latency. The following equation can be used to calculate the number of banks needed on a single chip in order to meet a desired sustained bandwidth target.

$$\text{Bandwidth, BW} = \frac{(\text{BitsPerArray} * \text{ArrayPerBank} * \text{Banks})}{(\text{AccessLatency})}$$

For example, in order to provide a sustained bandwidth (BW) of 16B/ns, assuming an access latency of 200ns, four bits per array, and 16 arrays per bank, 400 banks would be needed.

$$\text{BW} = \frac{4 * 16 * 400}{200\text{ns}} = 16\frac{\text{B}}{\text{ns}}$$

My sustained BW calculation indicates that each core needs to access 400 banks in parallel to deliver 16B/ns, to provide sufficient bandwidth for real-world applications. Note that I chose a per-bank access granularity of 8B, which is desirable for fine-access granularity applications, and therefore have 16 arrays per bank. ReRAM architectures require high amount of parallel memory accesses. The high access time is amortized by accessing multiple arrays at once. Multiple processors would need to access these arrays in order to fully exploit the fine access granularity provided by ReRAM. This necessitates the need for a highly efficient network-on-chip connection topology to service the requests between the multi-processor system and the memory system.

A tiled floorplan, consisting of memory-processor tiles, incurs a relatively high area overhead of 20% caused by the embedding the ReRAM peripheral circuits with the CPU logic [25]. The inefficiency is caused by the CPU logic circuits not having contiguous space for the digital implementation. Additionally, the memory access patterns based on our simulation results also pointed to the fact that they are not dedicated to the memory immediately above it. Therefore, the overall architecture needs to support memory access patterns to any of the memory within the chip. Rather than a tiled floorplan, I selected a centrally located ReRAM type of floorplan, as shown in Figure 7-4.

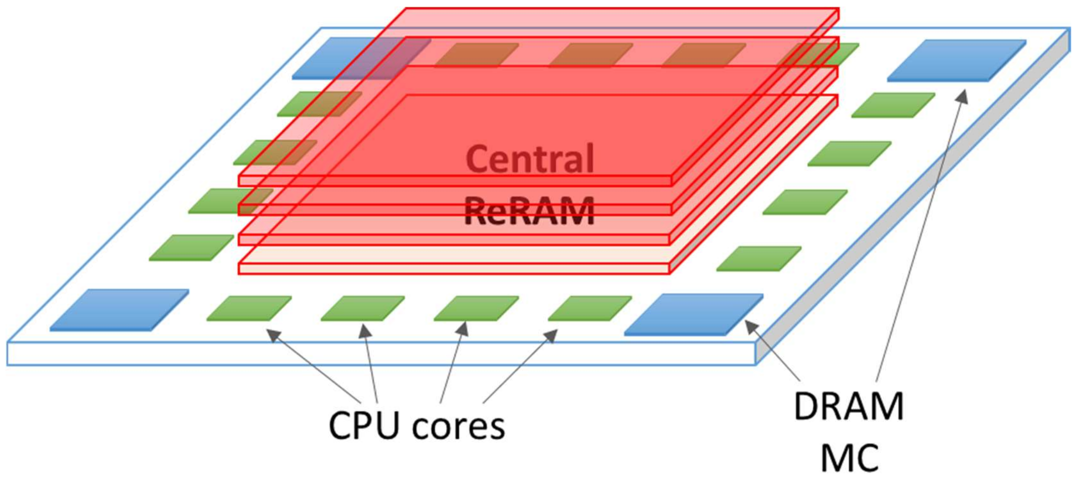


Figure 7-4 - Hybrid ReRAM-DRAM System

This type of central-memory floorplan will allow for the ReRAM to be treated as an embedded unit with their own internal NoC network. Additionally, the CPU cores can be independent of the memory, and be implemented with a contiguous area floorplan. DRAM memory controllers could also be provided for cache, to be selectively used for write-intensive or sequential access type of algorithms.

7.2.2 NoC Topologies of Interest

As mentioned before, a homogeneous 2D-Mesh topology for the Network-On-Chip (NoC) is unlikely to keep up with the relatively high communication need of the ReRAM-architecture I envision. In order to support low latency across the chip, I selected a set of NoC topologies that can support high network capacity for a large number of nodes on-chip.

Based on previous area studies, I calculated that it is feasible to have more than 1000 individual memory banks, each needing to be accessed independently. This would entail over 1000 network endpoints, which maybe fairly large for a simple mesh or torus type of NoC topology. Therefore, I selected topologies, even ones not typical for NoCs, following trends in supercomputers, where 1000s of network endpoints is commonplace. I consider the following four topologies of interest to evaluate my performance comparison: mesh, torus, fattree, and dragonfly. Figure 7-5 illustrates the high-level connections of the different components for each topology. In the figure, the rectangles denote the component to be connected, the small circle denotes the router endpoints, and the lines denote the links and connections between the endpoints. These four topologies can be modeled in the architectural simulator that I had chosen, SST. The next subsection summarizes the simulation details.

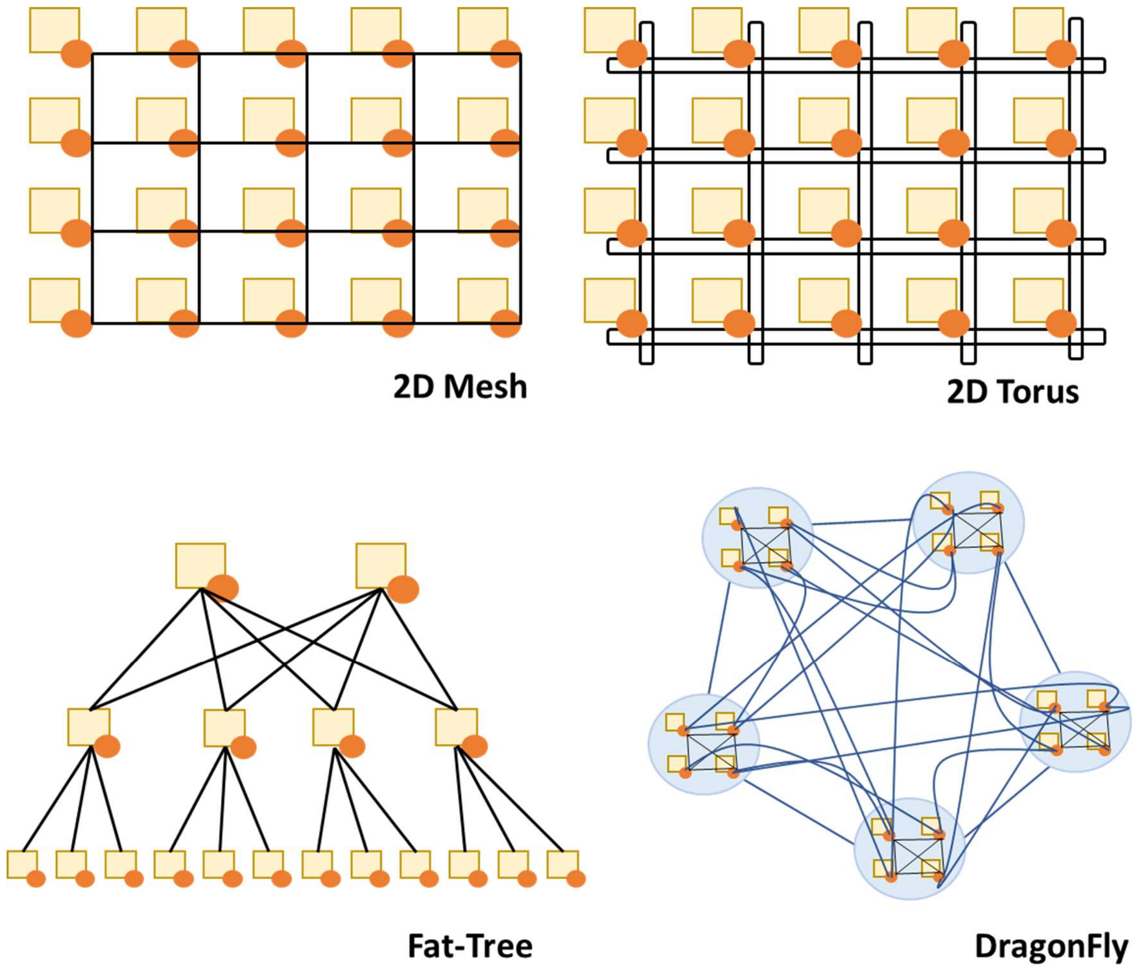


Figure 7-5 - Overview Diagram of NoC Topologies Simulated

1. **2D Mesh**: Mesh is the simplest and most widely used NoC topologies due to its ease of physical layout. For my study, I limited the comparison to a two-dimensional mesh which consists of an array layout. Every network switch is connected to four neighboring switches and one component, which could either be a processor or a memory controller.

2. **2D Torus**: This topology is similar to mesh with the added connection between the endpoints. This has the advantage of allowing for better fairness

due to limiting maximum number of hops, while slightly increased complexity and area cost.

3. **Fat-Tree**: Also known as flattened butterfly, the topology follows a hierarchical layout. The higher-level root nodes have more connections than the leaf nodes [38]. In my study, I use a 3-tier fat-tree Noc topology for my experiments.

4. **Dragonfly**: The dragonfly network is a high-order radix topology that is also hierarchical in nature. Switches are clustered together in groups with high inter-group connections. Intra-group connections between other groups are formed to provide high connectivity. The number of connections between any two nodes is limited three hops (Local-Global-Local) but requires more complex physical implementation [39].

7.2.3 Simulation Methodology

My simulation environment consists of the SST (Structural Simulation Toolkit) to model and evaluate the different memory architectures and NoC topologies. As before, I made use of the following external element libraries to model the different system components.

- **Miranda** - Pattern-based CPU model to model the individual processor on STREAM and GUPS benchmarks

- **Merlin** - Network-On-Chip router model to model the different topologies and specify the connection links between the different components.
- **MemHierarchy** - L1 and L2 cache model
- **DRAMSim3** - DRAM Memory model for DDR4 and HBM2 devices
- **Messier** - ReRAM Memory model with asymmetrical read and write latencies

7.3 Experiment Results & Analysis

In this section, I present the NoC topology comparison simulation results comparing the four topologies: Mesh, Torus, Fattree, and Dragonfly. Additionally, I also studied the impact of higher number of memory controllers to support a hybrid ReRAM-DRAM architecture. Figure 7-6 shows the central ReRAM torus topology, with the access points to the CPU lying on the boundary in darker green color.

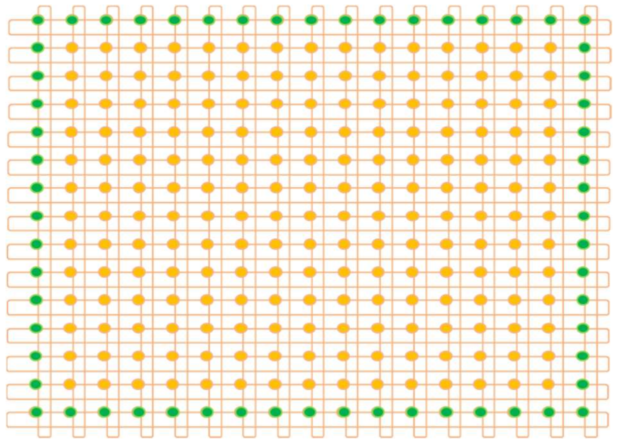


Figure 7-6 - Torus Configuration for Central ReRAM Architecture

7.3.1 NoC Topology Evaluation

The SST architectural simulation result presented in the previous chapter used a Mesh topology for the comparison between DRAM and ReRAM. That work showed the configurations that are optimal to take advantage of ReRAM, which are sparse access patterns and higher core counts. Next, I studied the impact of various NoC topologies presented in Section 2 on the highly parallel ReRAM architecture described in the previous chapter. I simulated the different topologies using a link bandwidth of 1, 2, 4, 8, and 16GB/sec for all topologies. The network parameters for the various topologies are provided in Table 7-2. The input and output buffer sizes were set to 2KB, with 2 virtual channels and a flit size of 16B. The mesh and torus are two-dimensional arrays of 34 rows and 35 columns to meet the maximum number of nodes for the simulation range. The fat-tree network is a 3-level tree, with 2048 hosts.

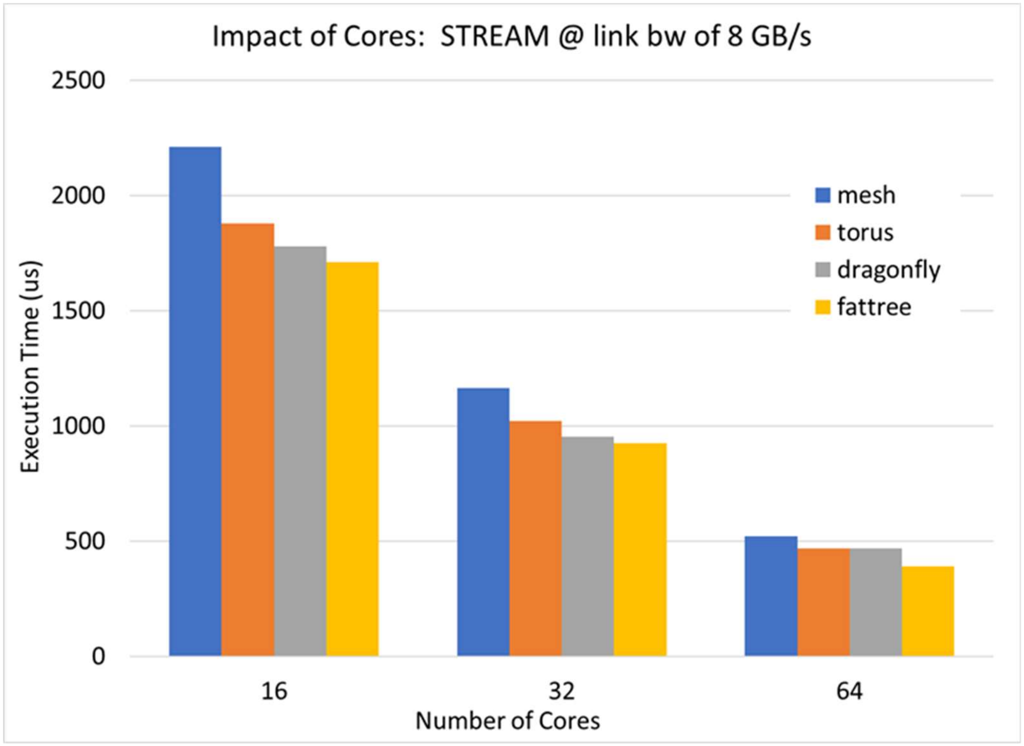
Topology	**Cores < 128**	**Cores ≥ 128**
Mesh	7x7 array	50x50 array
Torus	7x7 array	50x50 array
Fat-Tree	8x8x32	8x8x(32*core/128)
Dragon-Fly		
# of Groups	32	256
Hosts/Router	8	8
Routers/Group	8	8
Inter-group Links	4	4

Table 7-2 – Network Sizing Parameters

At the lowest and middle level, there are 256 routers per level, with 8 upper and 8 lower links each. At the top-level of the fat-tree, there are 64 routers with 32 links. The fattree topology was simulated using a deterministic routing algorithm that only relies on the source and destination address, rather than the current state of the network. The dragonfly network has a high radix of upto 30 links per router at the lowest level. Minimal routing algorithm was used on the dragon fly topology, which selects the route based on the shortest path to the destination. The table also summarizes the total links present in the network for the simulation ranges of my experiment.

Figure 7-7 shows the summary of the NoC comparison simulation results in terms of the raw execution time for a system with 16, 32, and 64 cores for STREAM and GUPS benchmarks. The link bandwidth was kept at 8 GB/s for all simulations. The x-axis is the number of cores and the y-axis is the total execution time. For the STREAM benchmark, I see that at the lowest core count simulated of 16, mesh has a much higher execution time than all of the other three topologies, with the fat-tree providing the best performance. This trend is much more prevalent for the GUPS benchmark, where the MESH is close to 2x slower performance than the fattree topology. Since GUPS requires higher number of separate connections to provide memory access to the CPU, it would have a higher load on the network.

As the core count increases, I see the specific topology having less impact on the performance. This could be due to the contention points being more spread apart, due to more originators of the memory requests. While the memory access points are very high (1000), the number of cores is only 16, this causes the network paths near the CPUs to be more congested. As I increase the number of cores, the contention is alleviated by spreading apart the location of the memory request origins.



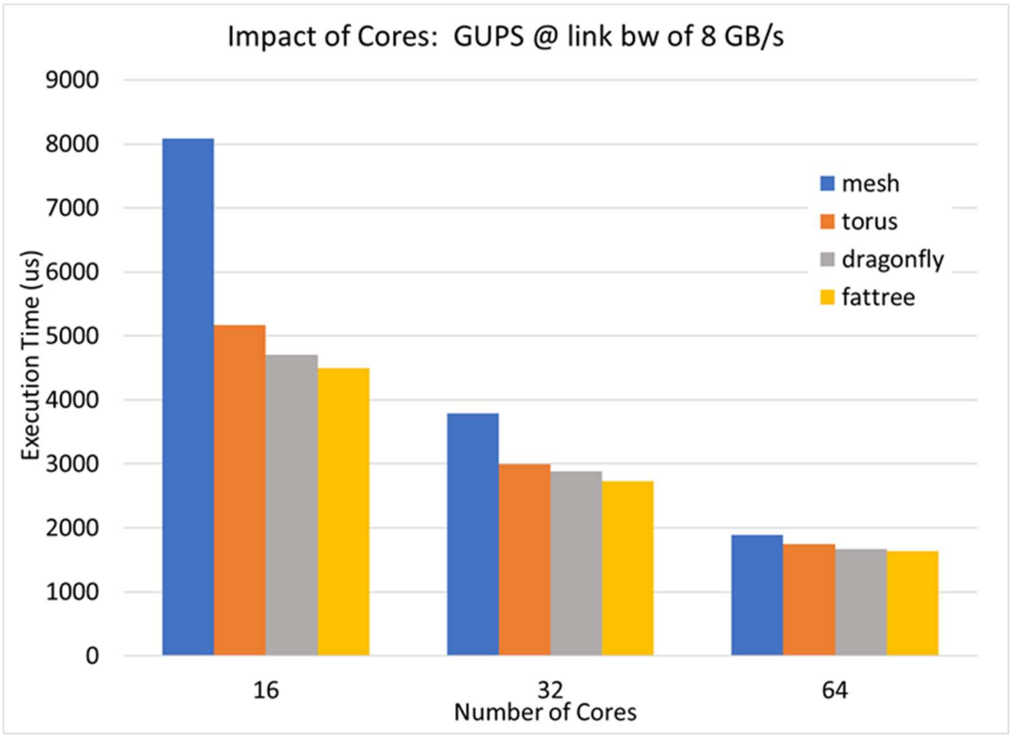


Figure 7-7 - NoC Topology Performance: Impact of Cores

The results show the impact of cores by keeping the link bandwidth constant at 8GB/sec for all topologies. For both benchmarks, I observe a consistent trend in the performance as follows. Mesh topology delivered the highest execution time, followed by torus, then dragonfly, and finally fattree topology. As the number of cores increase, the performance improves, although saturates and the network topology has less impact on the overall performance. This is especially notable for the the GUPS benchmark at the 16-core configuration. Here, mesh had much higher execution time due to network contention owing its sparse finer access granularity.

I next computed the speedup of scaling the performance using the lowest core count, 16, as the baseline. The speedup is used to assess how much faster parallelizing the system improves the system performance, and is calculated by

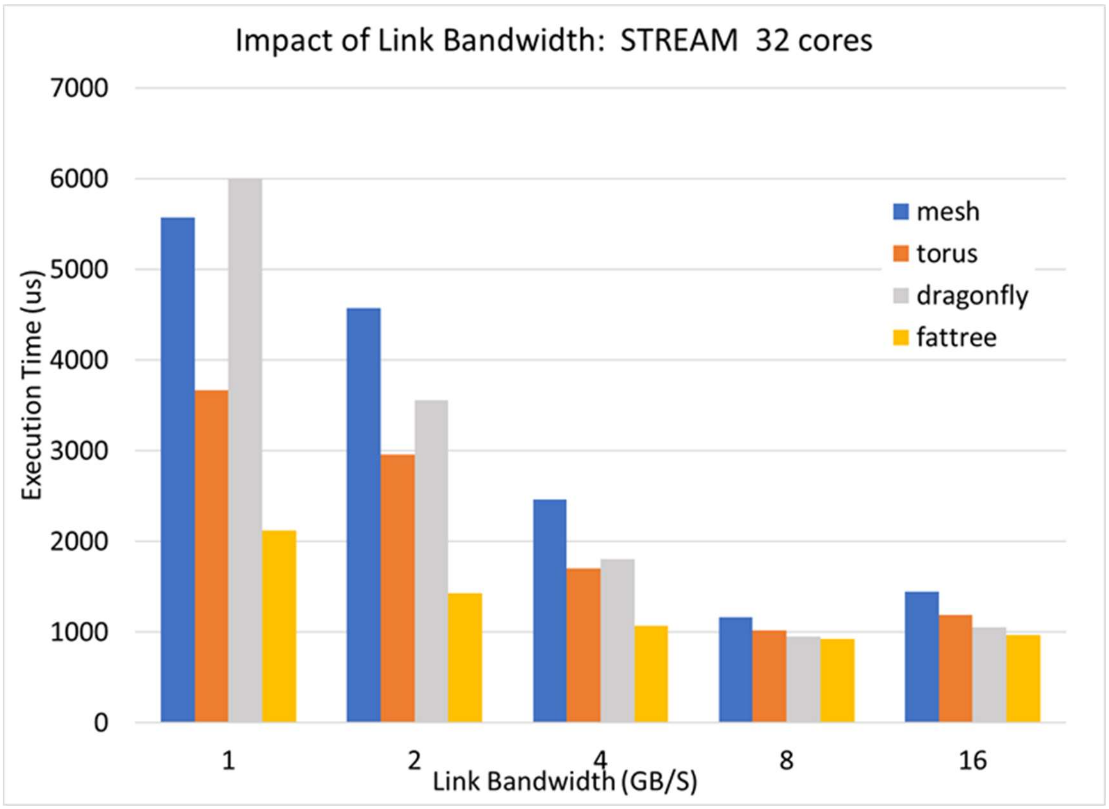
$$\frac{\mathrm{ExecutionTime}_{\mathrm{Corei}}}{\mathrm{ExecutionTime}_{16}}$$

Table 7-3 shows the effectiveness of scaling up for the different cores from a baseline of 16 cores.

# Cores	GUPS 32	GUPS 64	STREAM 32	STREAM 64
dragonfly	1.64	2.83	1.87	3.80
fattree	1.65	2.75	1.85	4.37
mesh	2.13	4.27	1.90	4.25
torus	1.72	2.96	1.84	4.02

Table 7-3 - Speedup for different NoC Topologies (baseline: 16 cores)

Figure 7-8 shows the impact of the link bandwidth on the performance, while keeping the number of cores constant at 32.



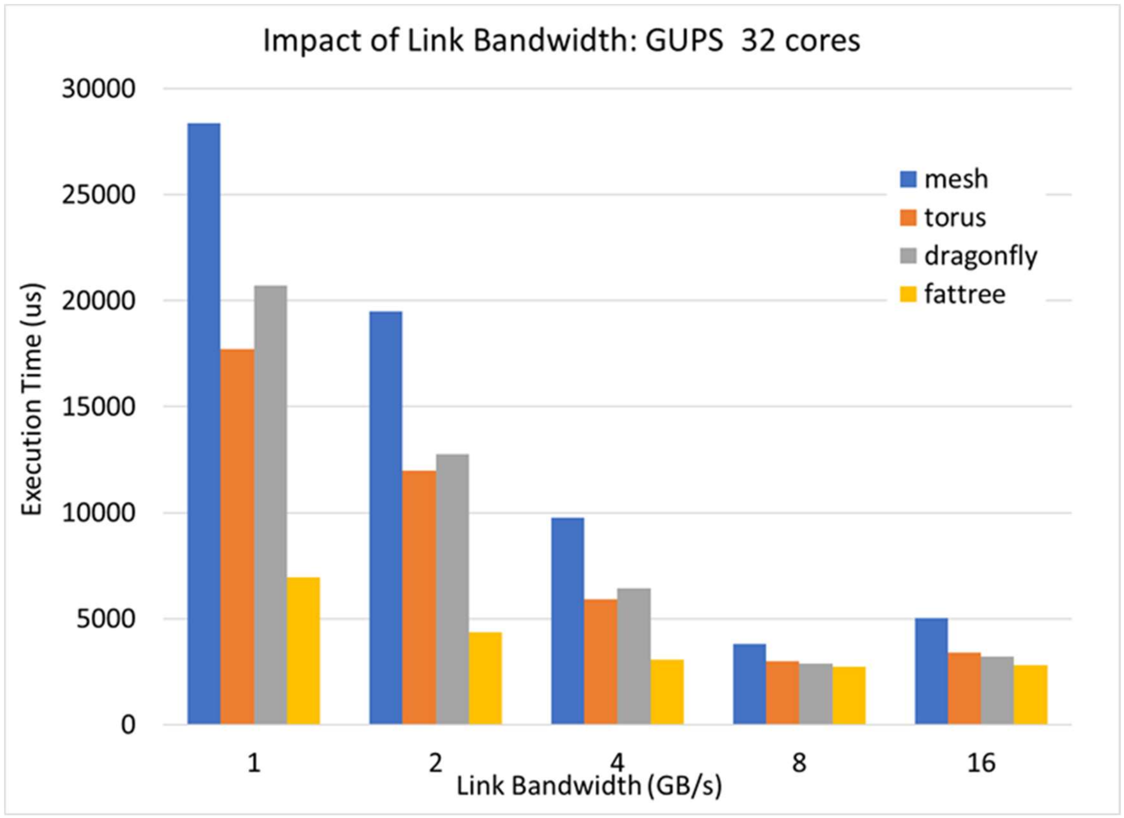


Figure 7-8 - NoC Topology Performance: Impact of Link Bandwidth

While mesh generally performs worst, at very low link bandwidths I see that dragonfly has a degraded performance, worse than both mesh and torus topologies. I looked at two network statistics to explain this anomaly at the 1GB/s link bandwidth scenario. The send-packet-count metric for torus had a higher (5x) number of packets sent overall. This indicates a higher number of hops needed for torus than the higher-order dragonfly, as expected. The output-port-stalls metric, however, showed that the dragonfly topology had a 6-orders of magnitude higher stall count than with dragonfly. This is likely due to Dragonfly using a greedy locally optimized routing algorithm that can cause local link saturation resulting in a bottleneck.

Similar results of poor performance of dragonfly over torus and fat-tree at low message sizes was reported in other works [42, 43]. Torus performed well overall for the benchmarks I simulated, at both reasonable link bandwidths of 4GB/s and above. Although, fat-tree topology had the lowest execution time, the marginal performance improvement seen over torus may not justify the added complexity at these ranges. Using Mesh topology as a baseline, at a link bandwidth of 2GB/s, fattree performed 78% better, while dragonfly and torus performed 35% and 39% better, respectively, for the GUPS benchmark.

The tradeoff between the complexity of the topology and the performance attained in terms of execution time is graphed in Figure 7-9 for the STREAM benchmark.

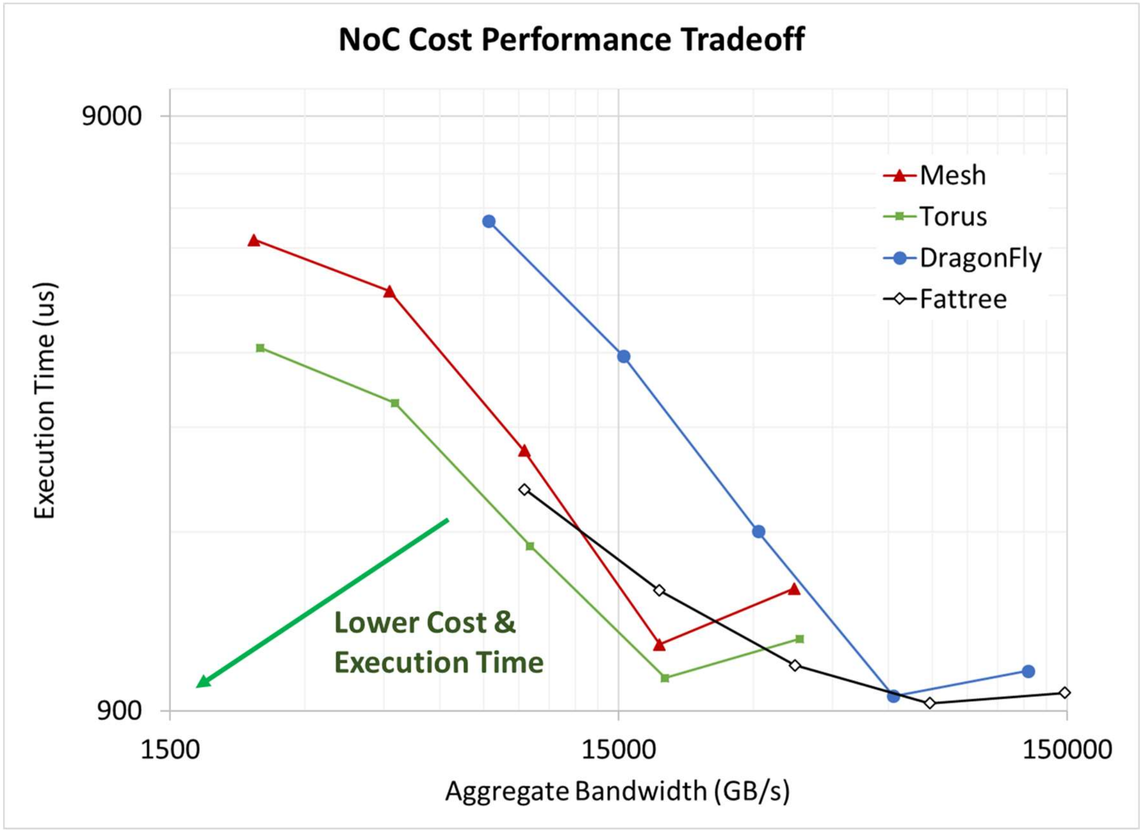


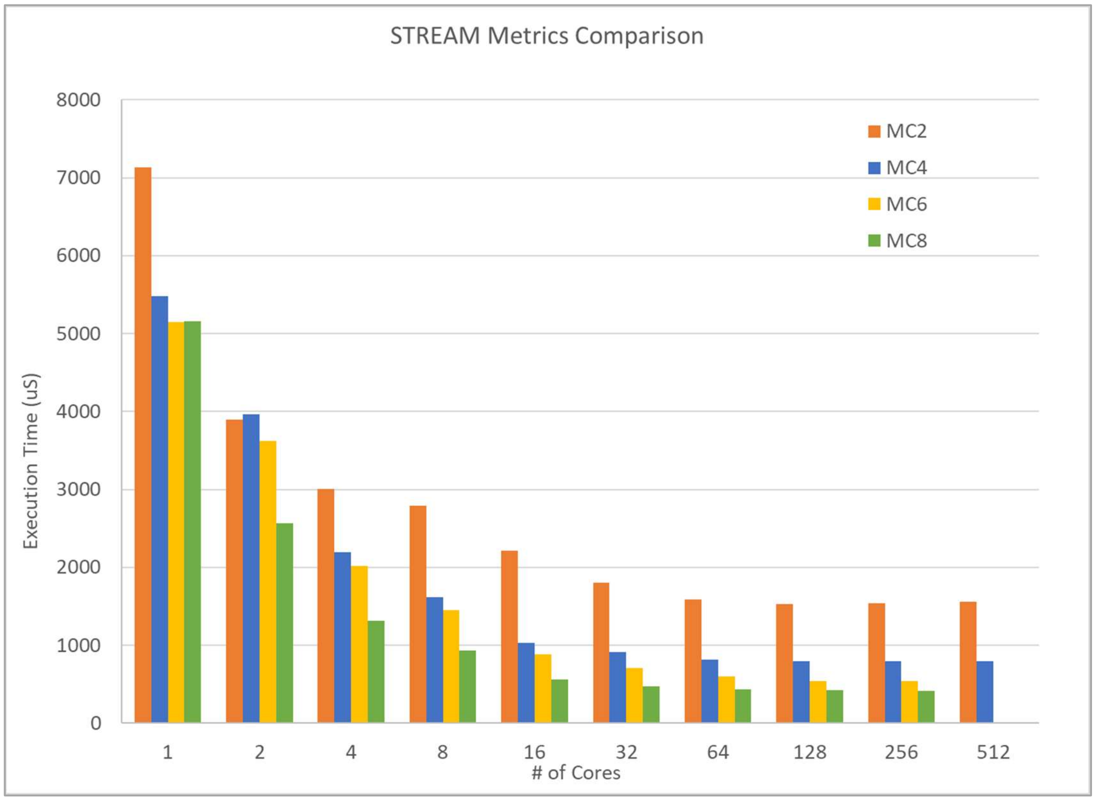
Figure 7-9 - NoC Topology Tradeoff: Execution Time vs Aggregate Bandwidth for STREAM benchmark (Note: Log Scale X & Y axis)

The total concurrent aggregate bandwidth is computed by multiplying the per-link bandwidth by the number of links summarized in Table 7-2. This is a rough proxy for the cost of the topology, in terms of both area, design complexity, and power dissipated by the NoC. Here, Torus topology offers the best trade-off in terms of complexity and delay, while Fattree topology can deliver the lowest execution time, at the cost of higher number of links. Mesh topology also delivered reasonable execution times with low cost. Dragonfly has a poor tradeoff at the lower bandwidth ranges based on the specific configuration that I had chosen. Future work will focus on selecting the optimal configuration of hosts/router and routers per group to reduce link saturation and produce a more balanced dragonfly network. Varying the workload and introducing additional benchmarks suites would yield a more rigorous comparison of the different architectures.

7.3.2 DRAM Memory Controller Optimization

In my final set of experiments, I performed sensitivity analysis on the number of memory controllers (MC) in a conventional DRAM system. The motivation for this final study was to optimize on the number of memory controllers in hybrid DRAM-ReRAM System referred in Figure 7-4. For this study, I used a simple mesh topology and varied the number of memory controllers to 2, 4, 6, and 8 for the full range of cores (upto 512 cores).

Figure 7-10 shows the execution time comparison for STREAM and GUPS benchmark. Again, scaling the cores improves the performance to a point, after which the performance saturates. At lower core counts, a higher number of memory controller cannot be fully utilized. The biggest improvement can be seen from two memory controllers to four for a specific core count.



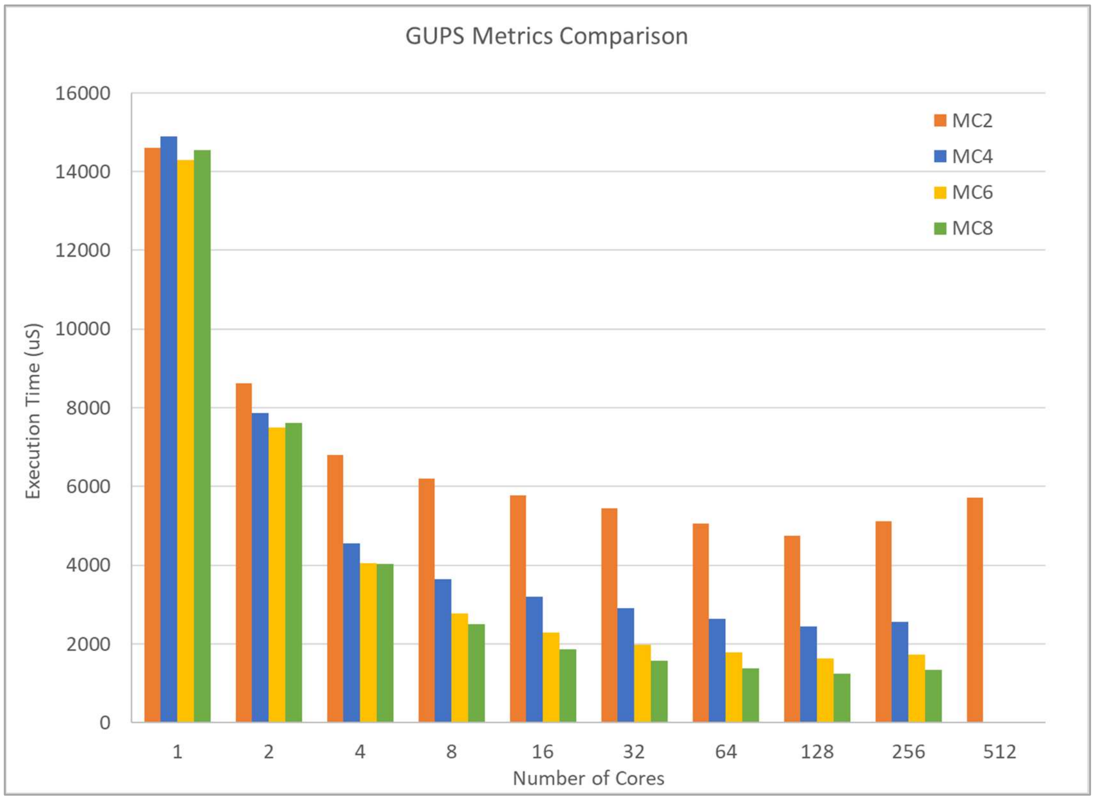
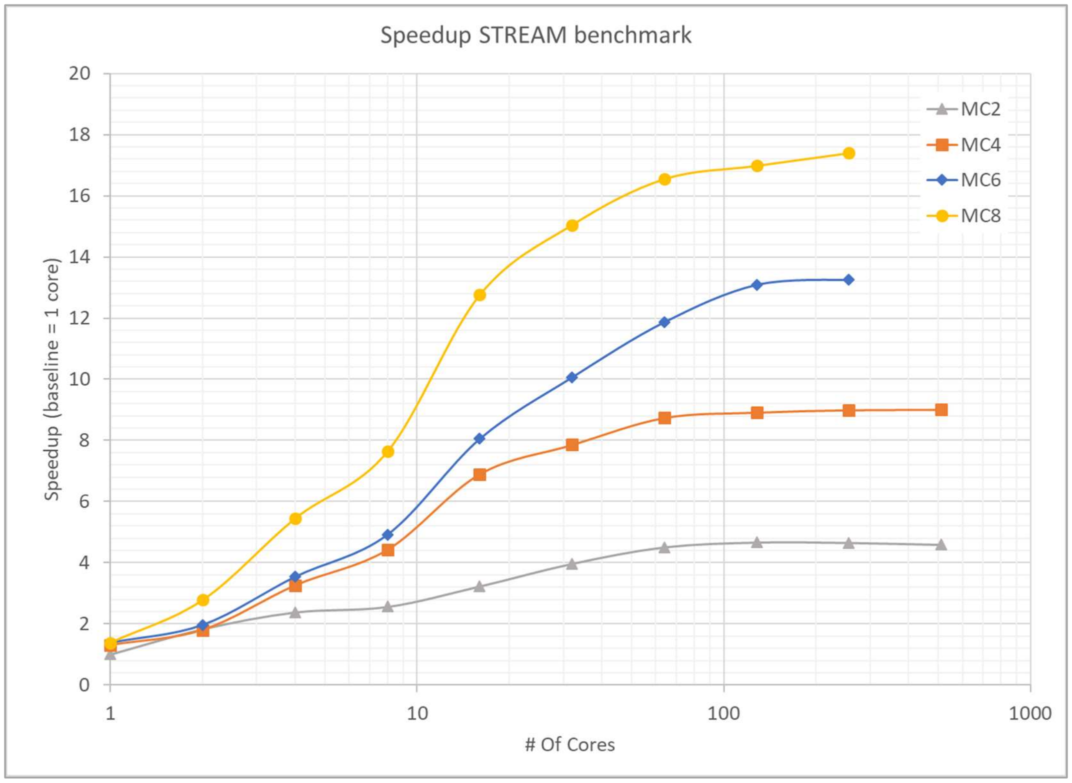


Figure 7-10 - DRAM Performance: Impact of Cores and Memory Controller

Figure 7-11 shows the speedup comparison, using the slowest overall configuration of 1 core and 2 memory controllers as the baseline. The plot shows that for both benchmarks, increasing the memory controllers has a marginal benefit. A configuration of 64 cores and 4 memory controllers seems to be an optimal trade-off between performance and cost. For the hybrid DRAM-ReRAM main-memory solution, having four DRAM Memory controller access points offered the highest performance gain.



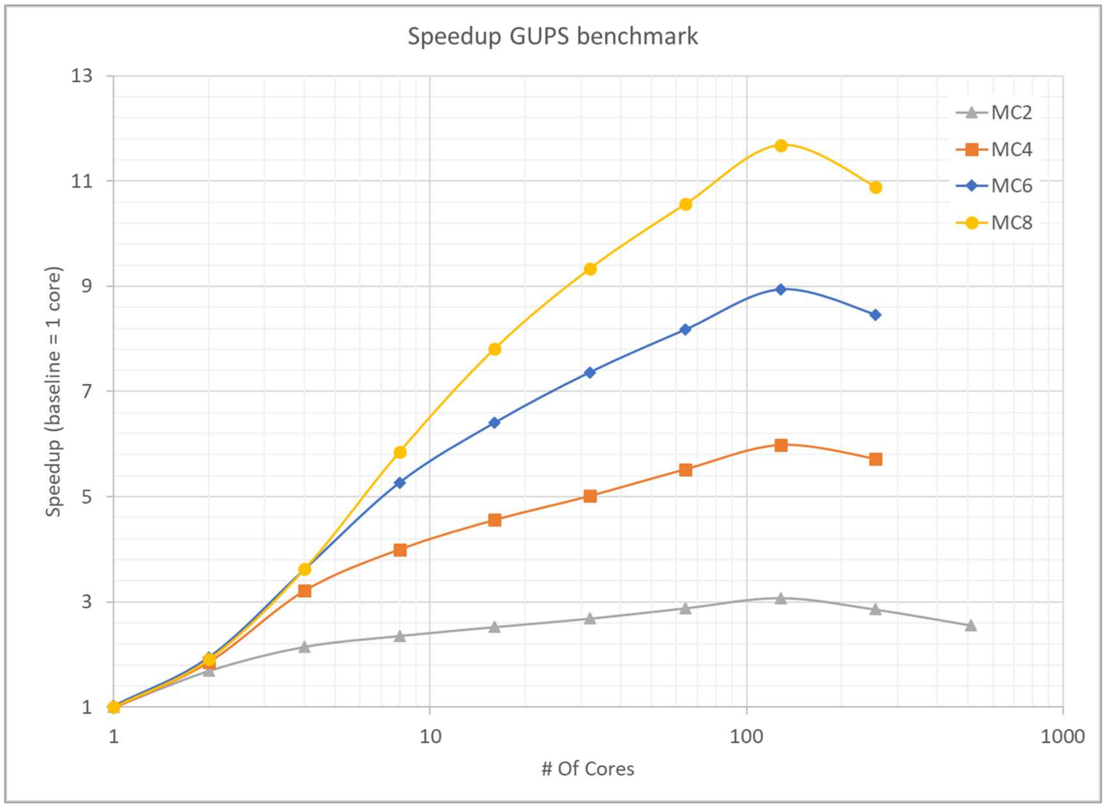


Figure 7-11 - DRAM Speedup: Impact of Cores and Memory Controller (note: Log-Scale X axis)

7.4 Conclusion

ReRAM as a main-memory delivers several advantages over conventional DRAM in terms of scaling, capacity, and performance for sparse-access patterns in support of parallel computations. Power-efficiency is also achieved due to the on-chip data access communication path. At higher core counts, ReRAM is able to surpass DRAM performance and results in lower energy cost. Torus Noc topology performed well in my simulation and might be preferred over fat-tree and dragonfly due to its simpler implementation and lower cost.

8 ReRAM as Trusted On-Chip Main Memory

8.1 Motivation

DRAM as a main-memory is one of the vulnerable points in a hardware system due to it being located off-chip. This opens the system up to snooping on the system bus, side-channel attacks on the memory data through mechanisms like row-hammering attack by malicious devices. Embedded DRAM variations, like eDRAM are limited in capacity and cannot accommodate space needed for real-word application workloads. Additionally, as DRAM faces scaling issues as a high-density memory, emerging memory technologies are being explored as alternatives. One promising alternative for this application is ReRAM, which is scalable, vertically stackable, and because of the possibility of integration with standard logic process, can deliver higher density as a main-memory solution. The key differentiator with this approach involves a ReRAM memory array that integrates directly with a logic processor underneath, eliminating the need to go off-chip.

ReRAM as an on-chip trusted main-memory which is impervious to side-channel attacks, leaves the memory more protected and prevents snooping of the bus. Additionally, by controlling the write energy applied during a program, I can selectively reduce the data-retention time and prevent the cold-boot access, a concern with non-volatile systems. Area studies and measurement results on a fabricated test structure demonstrating the cell relaxation is presented. Architectural performance comparison against a DRAM system shows a 30% improvement.

Secure processor architecture requires addressing both processor and off-chip memory access vulnerabilities. In conventional system architectures, critical data in RAM is typically located off-chip in DRAM and could be comprised due to two major security vulnerabilities. The first is bus-snooping, 1 in Figure 8-1, on the connection between the processor chip and a Main-Memory system that is located off-chip. The second concern is DRAMs vulnerability to Row-Hammer Attacks, 2 in Figure 8-1, whereby accessing a bitcell repeatedly in succession, an adversary is able to introduce data disturbance on a bit in an adjacent column.

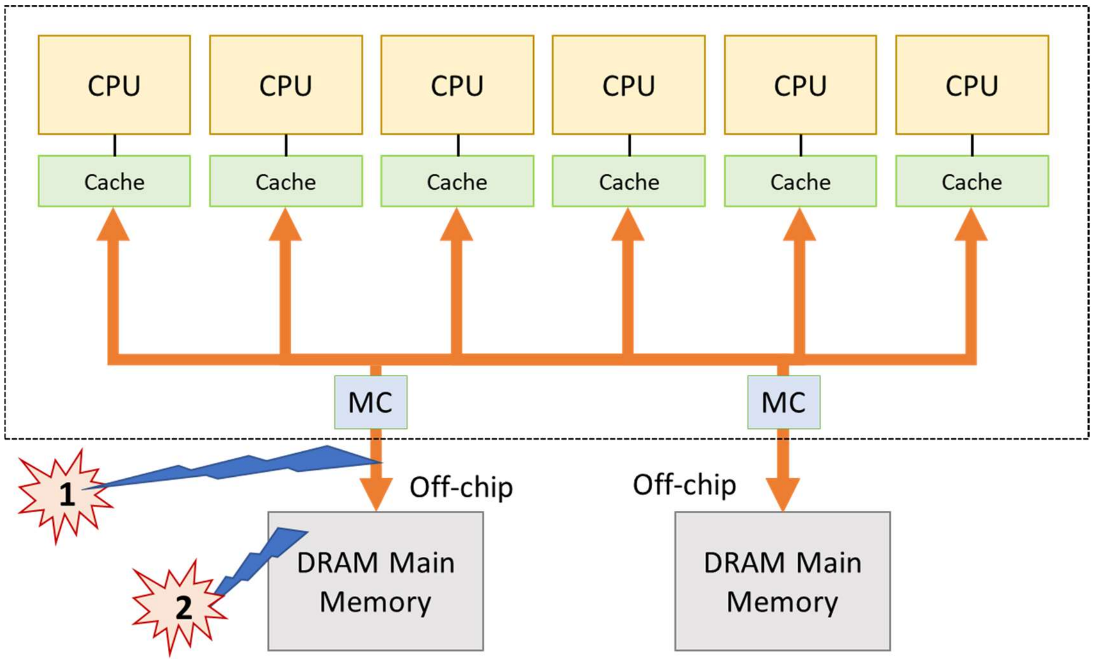


Figure 8-1 - Vulnerabilities in Main Memory

In addition to these security vulnerabilities, DRAM as a high-density memory is reported as facing scaling issues and being vulnerable to failure at advanced technology nodes [16]. Being located off-chip, DRAM has to interface to the processor system through a limited set of memory controller access points. This is especially true for multi-processor systems, as shown in Figure 6-7, where the interface to the main-memory system is through a limited set of memory controllers, often on the order of 4-8 access points per chip. This limited set of connections leads to performance bottlenecks which result to high latencies at the system level, despite low memory latencies. On-chip DRAM options, such as embedded DRAM (eDRAM), are not viable options due to their larger bitcell size and limited capacity. The key advantage of ReRAM, from a system-vulnerability and performance point of view, is that they are

On-Chip, allowing for the processor to be directly connected to the memory. ReRAM functioning as an on-chip main-memory, enhances both the performance and security of the system.

ReRAM, being a non-volatile memory, does have its challenges [11]. Being a non-volatile memory, it is especially vulnerable to cold-boot type of attacks, where data could be recovered from the hard-disk. At the device level, ReRAM's read and write latencies are much longer than DRAM. The write endurance limits are also much lower than what DRAM is able to deliver, which poses an issue for typical applications to be supported. My solution to this is that by controlling the write-energy applied which has an advantage for both performance and security of these ReRAM-based Main-Memory architectures.

8.2 Background

I will go briefly into the bitcell operation mechanism. A conductive filament is grown in the middle layer by applying electrical stress, which allows for the resistance of the device to be modified. Figure 8-2 shows a cross-section of a resistive memory and the resistance modifying behavior. In ReRAM, when the conductive filament is created, the cell is in low-resistance-state (LRS). On the other hand, when the filament is broken by, for example, applying a high-voltage of the opposite polarity, the filament is broken causing the bitcell to be open-circuit, in a high-resistance-state (HRS). The filament is created or broken in the middle dielectric layer(s) by applying a high enough current or voltage causing a dielectric breakdown. These materials are engineered so that the breakdown is not permanent and is reversible, upto a certain number of cycles. The write endurance specifies the number of these write cycles before the bitcell fails and can no longer transition.

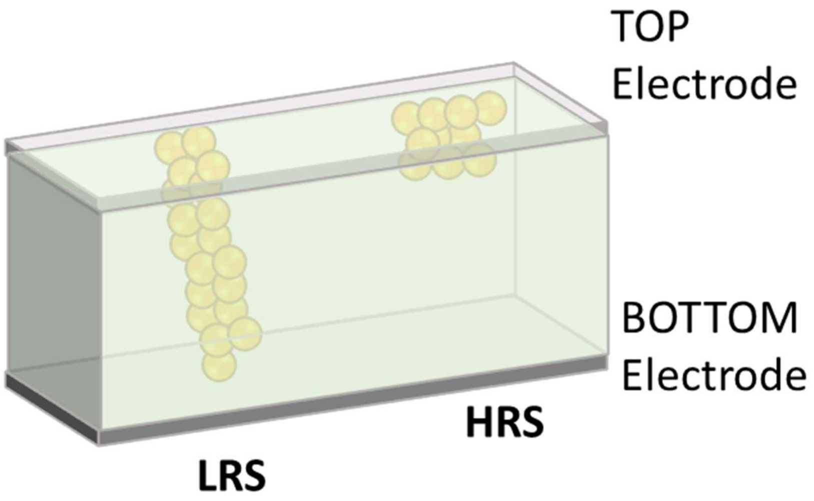


Figure 8-2 - ReRAM Resistance Creation

8.2.1 Integrated Processor-ReRAM Architecture

An integrated Processor-ReRAM architecture layout has the flexibility to be configured in many ways. The data access pattern between the processors and the memory systems for the application space would be a key determinant. Figure 8-3 shows two possible approaches. The tiled configuration consists of individual ReRAM arrays embedded into a larger processor. These processor-ReRAM tiles are ideal for highly local data accesses where each processor computes on workload in the main-memory located over its tile. The central ReRAM configuration shows a high number of individual ReRAM arrays located centrally, surrounded by multiple processors. This configuration is desired for access patterns that are sparse and random, requiring any-to-any connection between the processor and an individual array. In this study, my area analysis focused on the tiled approach.

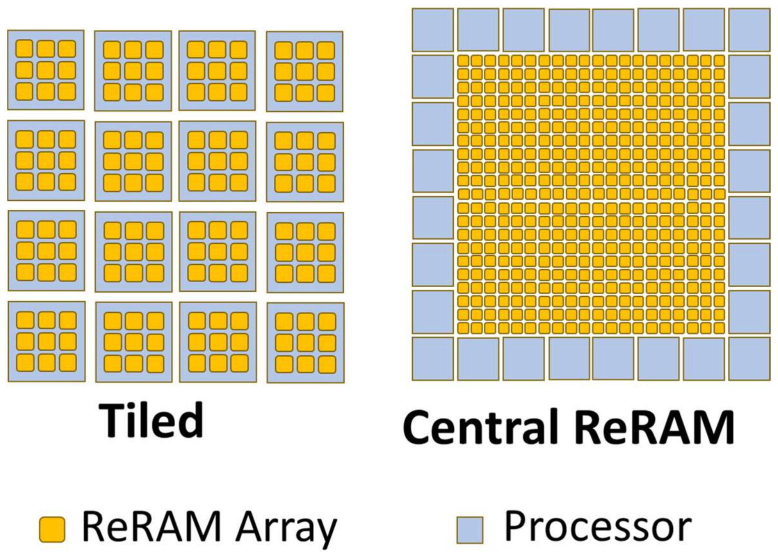


Figure 8-3 - Integrated ReRAM-Configuration

8.2.2 ReRAM Security Implications

From a system architecture point of view, using ReRAM as a main memory that is integrated directly onto a processor enables several security advantages. Figure 8-4 illustrates an on-chip ReRAM based main memory solution, with the connections between the multiple processor subsystem and the ReRAM arrays handled by a Network-On-Chip interconnection. By being on-chip, ReRAM is impervious to side-channel analysis. All of the communication channels between the processor and memory is through on-chip metal vias and thus not available for bus-snooping. Additionally, ReRAM is also not susceptible to the data-disturbance seen in DRAM through the Row-Hammer attack. Memory systems are susceptible to cold-boot type of physical attacks to recover data. In this form of attack, an adversary that has physical access to the hardware performs a memory dump of the RAM in order to obtain encryption keys or other sensitive data. Even without power, DRAM main memory remains stable for a short duration, before the charge on the bitcell is dissipated. This is characterized as the time between refresh cycles, which is often set as 65ms as a conservative specification. In a cold-boot attack, this data recovery time is extended by the adversary lowering the temperature of the memory module. This slows the discharge on the DRAM bitcells capacitor, thereby retaining the data on the bitcell well past the refresh time needed.

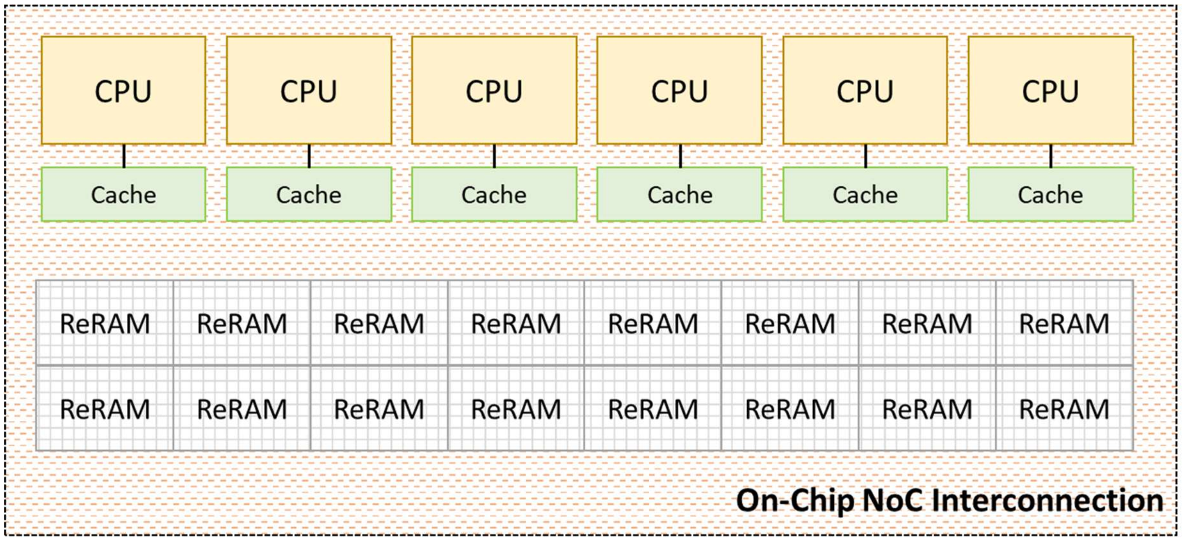


Figure 8-4 - ReRAM-based Main-Memory Solution

Since this attack exploits an intrinsic hardware vulnerability, it poses a serious threat even for trusted platforms [30]. Cold-boot attacks are even more problematic for nonvolatile memories. ReRAM as a non-volatile memory retains data without power, with a typical data-retention time of 10 years. This makes ReRAM-based main memories to be especially susceptible to these types of physical attacks to recover data from a system.

8.3 Proposed Approach

My solution for ReRAM's cold-boot attack problem comes from the insight that ReRAM for main-memory applications do not necessarily require non-volatility of data. Currently, DRAM as a main memory is volatile and extends its data remanence with periodically through a refresh operation from storage. If I are able to selectively control the data-retention time of the ReRAM by applying a lower write-energy, I would be able to reduce the impact of cold-boot type of attacks by preventing the data from being available for long times. Figure 8-5 illustrates ReRAM behavior operating in three different modes based on the electrical stress applied. The electrical stress is indicated in the figure's y-axis by controlling the current compliance limit applied during the program operation. What this would mean is a system where the main memory retains the data for a short time, on the order of a few milliseconds. I can mitigate the data-loss by apply a periodic refresh, which is a manageable solution similar to how DRAM deals with data discharge on its bitcell. Furthermore, studies on the impact of temperature on data retention indicate high stability for ReRAM bitcells [32]. This implies that ReRAM's data retention time may be unaffected by external lowering of the temperature, bolstering it against cold-boot type of attacks.

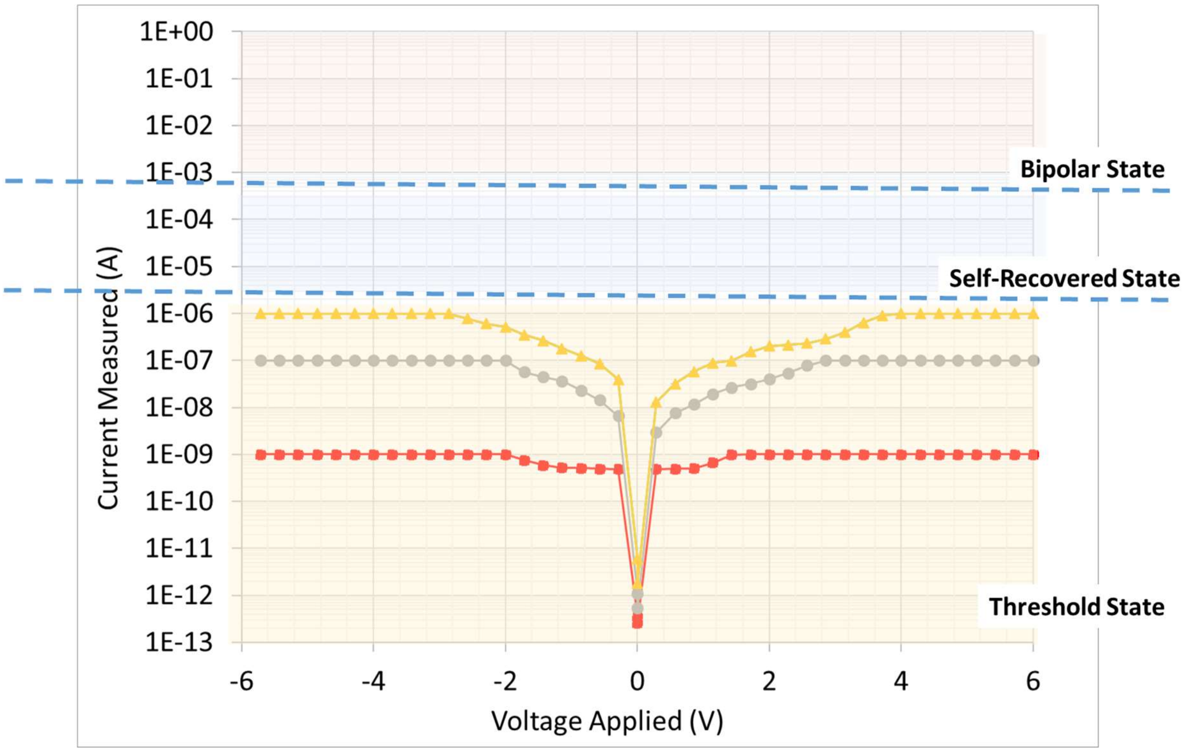


Figure 8-5 - ReRAM Three Modes of Operation

ReRAM's data-retention time is a function of the energy applied to the cell as a function of the voltage and current applied during a Program or a write operation. If a lower write energy is applied, that would result in either a lower program voltage and/or lower write latency, both of which have positive performance implications. Additionally, a RESET cell (LRS) in the digital volatile mode must also be placed in this lower state in order to make it non differentiable to an unpowered read attack.

Additionally, lower write energy would also result in improved write endurance for the cell, which is one of the device challenges with ReRAM technologies [11]. Finally, lower data retention time also helps prevent cold-boot ReRAM data from being accessible by a malicious adversary.

8.4 Analysis and Discussion

In the Chapter 5, I demonstrated that the digital volatile ReRAM behavior of the data is automatically lost after a short duration. I did this by fabricating discrete ReRAM devices in order to observe the programmed cell relaxing from a set to a reset value. I used Physical Vapor Deposition (PVD) to create my ReRAM stack using Platinum and Titanium for the top and bottom electrodes, and Aluminum Oxide and Titanium Oxide for the dielectric layer. The selection of this particular ReRAM stack was based on prior work that had demonstrated short term-time-dependent plasticity (STDP) to mimic neuron behavior [22, 35].

The results presented in the previous chapter, demonstrates the observed cell relaxation after a duration of 10 minutes. From a system security point of view, this relaxation behavior can be exploited to ensure that certain critical information could be programmed in an intermediate region so that the data is lost after a set duration. The effect of temperature on the data-retention, specifically whether cold temperature will extend the data retention time is something to be explored.

As mentioned in section 2 of this chapter, previous studies showed little impact of temperature on the data stability of ReRAM devices [32]. The primary mechanism of resistance creation in TiO2 based resistive memories is through the creation of oxygen vacancies through redox reactions, rather than dominantly from thermally based mechanisms like with Phase Change Memories (PCM). However, since redox reactions could be accelerated by heat, there might be a slow-down in the relaxation time in the case of a lower-temperature. This particular effect was not studied as part of this work and would be a good extension of this research for the future.

ReRAM as a main-memory delivers several advantages over conventional DRAM in terms of scaling, capacity, and performance for sparse-access patterns in support of parallel computations. Power-efficiency is also achieved due to the on-chip data access communication path. In addition to these performance benefits, on-chip ReRAM main memory can be a trusted hardware resource. There is no off-chip system bus snooping and no vulnerability to row-hammer hardware attacks.

In this paper, I presented the opportunity for ReRAM to be leveraged as mixed volatility main memory based on the electrical stress applied. The low data-retention time avoids Cold-Boot physical attack on the system by clearing the data over short time. There would also be a tradeoff of lower write energy leading to improved write endurance which is an effect to be studied in future work. Additionally, the experimental study could be repeated at lower and higher temperature in order to evaluate the effect of temperature on the data retention and the behavior in the cold-boot scenario.

9 Conclusion

A monolithic processor that integrates ReRAM memory and processor requires optimum configuration of the Core, NoC Topology, and memory controller at the architecture level to fully exploit the advantages. The core/CPU needs to be able to issue multiple non-blocking memory requests per cycle. This can be achieved through superscalar or multi-threading processors with SIMD flexible scatter-gather memory requests [11]. The Network-On-Chip needs to support high-throughput, which can be realized by incorporating higher-dimensional alternative NoC topologies. The ratio of Memory Controllers to cores need to be optimized to balance the area incurred against the need for parallelism. A summary of the key contribution is presented in the table below.

In this work, I have demonstrated a method for evaluating integrated ReRAM-Processor type of architectures using standard EDA tools. I also presented an overview of Crossbar ReRAM technology that has been demonstrated in fabricated silicon chips that allow this novel on-chip main memory architecture. My layout results indicate that

I can integrate a cluster of ReRAM mat arrays with a processor logic underneath and incur an area penalty of 18% and an overall area efficiency of 50%. Based on the memory access patterns, however, I noted that a central ReRAM approach would allow for independent development of the ReRAM and Processor logic. The area under the ReRAM array could be used to support SRAM cache array and the NoC interconnect logic.

Knowledge Area	Contribution
Physical Design	Digital Implementation of Integrated ReRAM-CPU solution
	Area analysis showed a 20% penalty with 50% area efficiency
	Floorplan of controller circuits underneath Central ReRAM block
Device Level	Fabrication of Resistive Test Structure by milling aluminum mask and performing PVD of test structure
	Demonstrated digital volatile cell behavior and confirmed bipolar program switching operation
	100x lower write energy per write possible in digital volatile state with similar lower write endurance
Architecture Level	ReRAM performs favorably with higher parallel requests
	Queue depth impacted dense memory access pattern
	Reducing ReRAM Write latency from 200ns to 1us improved the bandwidth by 25% for GUPS benchmark
	Fat-tree and torus NoC topologies provide performance gains of 78% and 39%, at bandwidth constrained scenario
	Torus NoC topology performed well across varying core count and link bandwidths

Table 9-1 - Summary of Key Contributions

The device-level research work I performed demonstrated that ReRAM could be used in a mixed-volatile state where a SET programmed cell could be placed to lose its’ value over time. This has several advantages such as lower write energy, which translates to lower write current and/or lower write latency, and improved write endurance due to the lower write energy applied. Additionally, controlling the volatility of the data in this manner, opens the memory technology to be used in many ways to selectively retain data for security or data persistence.

Finally, architectural simulations comparing ReRAM and DRAM based architectures showed that ReRAM-based main memory architectures outperform at higher core counts, where their high amount of memory parallelism can be sufficiently utilized. My simulations showed the cross-over point where ReRAM outperforms DRAM-DDR4 to be at 64 cores for the STREAM benchmark. My NoC topology comparison indicated both Fat-Tree and Torus topologies to have good performance for my configuration, with torus being an optimal choice due to its simplicity of implementation.

ReRAM as a main-memory delivers several advantages over conventional DRAM in terms of scaling, capacity, and performance for sparse-access patterns in support of parallel computations. Power-efficiency is also achieved due to the on-chip data access communication path. In addition to these performance benefits, on-chip ReRAM main memory can be a trusted hardware resource. There is no off-chip system bus snooping and no vulnerability to row-hammer hardware attacks. In this paper, I presented the opportunity for ReRAM to be leveraged as mixed volatility main memory based on the electrical stress applied. The low data-retention time avoids Cold-Boot physical attack on the system by clearing the data over short time. There would also be a tradeoff of lower write energy leading to improved write endurance which is an effect to be studied in future work.

10 Future Work

In this chapter, I summarize future work of the different research aspects that were investigated in terms of physical-design, device-level, and architecture-level.

Physical Design: With the increased complexity of integrating two distinct full-chip like designs, floor-planning placement of the blocks, their orientation, and location of the I/O ports will be critical in minimizing routing congestion. The next future work can explore floor-planning and digital implementation of the central ReRAM approach with multiple surrounding processor cores, NoC router circuitry, and any additional hardware accelerators for optimal performance of graphical processing applications. This will allow to quantitively evaluate placement options for the multi-core central ReRAM fabric to maximize I/O bandwidth to individual tiles and the intra-tile communication network needed. Thermal dissipation of the underlying logic circuits through the ReRAM BEOL layers is a possible concern that needs to be looked at.

The current area study was limited to a simple RISC-V processor in an academic 45nm technology. The next course of study can include more complex and divergent processors to stress the connectivity to the memory bank network. Additionally, extending to a more advanced process nodes with a process design kit (PDK) from a foundry such as TSMC, GlobalFoundries, SEMS, would make the diversity of standard-cell logic more accurate in the area estimations.

Device-Level: Several points of observation merit a closer look. Volume data on the observed volatile state is critical to provide more data points which would lead to better averaging of the program current compliance and the expected rate of the relaxation. As mentioned in Ch. 5, volume data for the characterization results would be useful to isolate noise and model the cell retention relationship more robustly. This requires fabricating a full array, with more than 1000 bitcells so that statistical analysis could be performed to more completely characterize the bitcell behavior.

Further analysis on the behavior for very high resistance bitcells and the increase in resistance after a delay needs to be analyzed. These can be further mapped to a model of the current compliance applied and the sequence of preceding program pulses. As for the dimensions of the cell, future work can try to make the dimensions smaller, towards the target dimensions seen in the intended application in order to minimize the creation of parallel filaments. Future work on the effect of oxygen partial pressure can be analyzed to identify any impurities created during the fabrication process. In regard to the trusted memory application, the effect of temperature on the data retention is critical in assessing the cold-boot attack approach discussed in chapter 8.

Architecture-Level: The impact of core count showed a inflection point for both ReRAM and DRAM based architecture where the system performance saturates at a point near the number of memory controllers. While this point was above the number of memory controllers for DRAM, it was well below that for ReRAM. Future work can explore the reason for this difference. One possible reason could be the differing memory models used. ReRAM used a simple Messier memory model from SST to model the latencies as a constant value. While for the DRAM, DRAMSIM was used to more accurately model the effect of reordering and stalls from pending requests.

On the NoC topology studied, further work is needed on optimizing the design configurations of the different topologies. For the dragonfly configuration, this would be the ratio of the number of groups, hosts, and routers. For mesh and torus topologies, a more optimal approach would be to scale the router array for each core count. Finally, an architectural simulation of the hybrid ReRAM-DRAM approach would be valuable to investigate a solution where the best points of each technology is exploited.

11 References

1. Y. Chen, C. Petti, “ReRAM technology evolution for storage class memory application,” 2016 46th European Solid-State Device Research Conference (ESSDERC), Lausanne, 2016, pp. 432-435.

2. Sung Hyun Jo, Kuk-Hwan Kim, and Ii Lu, “High-Density Crossbar Arrays Based on a Si Memristive System,” Nano Letters, 2009, Vol. 9 No (2), pp. 870-874

3. G. C. Adam, B. D. Hoskins, M. Prezioso, F. M. Bayat, B. Chakrabarti and D. B. Strukov, "Highly-uniform multi-layer ReRAM crossbar circuits," 2016 46th European Solid-State Device Research Conference (ESSDERC), Lausanne, 2016, pp. 436-439.

4. Sung Hyun Jo, T. Kumar, S. Narayanan, W. D. Lu and H. Nazarian, "3D-stackable crossbar resistive memory based on Field Assisted Superlinear

Threshold (FAST) selector," 2014 IEEE International Electron Devices Meeting, San Francisco, CA, 2014, pp. 6.7.1-6.7.4.

5. I. Bhati, M. T. Chang, Z. Chishti, S. L. Lu and B. Jacob, "DRAM Refresh Mechanisms, Penalties, and Trade-Offs," in IEEE Transactions on Computers, 2016, vol. 65, no. 1, pp. 108-121.

6. M. R. Guthaus, J. E. Stine, S. Ataei, B. Chen, B. Wu, M. Sarwar, "OpenRAM: An Open-Source Memory Compiler," Proceedings of the 35th International Conference on Computer-Aided Design (ICCAD), 2016.

7. T. Y. Liu et al., "A 130.7mm2 2-layer 32Gb ReRAM memory device in 24nm technology," 2013 IEEE International Solid-State Circuits Conference Digest of Technical Papers, San Francisco, CA, 2013, pp. 210-211.

8. R. Fackenthal et al., "19.7 A 16Gb ReRAM with 200MB/s write and 1GB/s read in 27nm technology," 2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC), San Francisco, CA, 2014, pp. 338-339.

9. A. Fumarola et al., "Accelerating machine learning with Non-Volatile Memory: Exploring device and circuit tradeoffs," 2016 IEEE International Conference on Rebooting Computing (ICRC), San Diego, CA, 2016, pp. 1-8.

10. MRAM-info. (2016, August). STT-MRAM: Introduction and market status. Retrieved from MRAM-info: https://www.mram-info.com/stt-mram

11. Meenatchi Jagasivamani, Candace Walden, Devesh Singh, Luyi Kang, Shang Li, Mehdi Asnaashari, Sylvain Dubois, Bruce Jacob, and Donald Yeung. "Memory Systems Challenges in Realizing Monolithic Computers." In Proceedings of the 4th International Symposium on Memory Systems (MEMSYS-IV). National Harbor, MD. October 2018.

12. Emerging Technology and Architecture for Big-data Analytics, by Anupam Chattopadhyay, Chip Hong Chang, Hao Yu (Chapter 4: Compute-in-Memory Architecture for Data-Intensive Kernels)

13. Shrunk-2-D: A Physical Design Methodology to Build Commercial-Quality Monolithic 3-D ICs, by Shreepad Panth, Kambiz Samad, Yang Du, and Sung Kyu Lim

14. Circuit design for beyond von Neumann applications using emerging memory: From nonvolatile logics to neuromorphic computing by Ii-Hao Chen; Win-San Khwa ; Jun-Yi Li ; Ii-Yu Lin ; Huan-Ting Lin ; Yongpan Liu ; Yu Wang ; Huaqiang Wu ; Huazhong Yang ; Meng-Fan Chang

15. A 16Mb dual-mode ReRAM macro with sub-14ns computing-in-memory and memory functions enabled by self-write termination scheme by Ii-Hao Chen;

In-Jang Lin; Li-Ya Lai; Shuangchen Li; Chien-Hua Hsu; Huan-Ting Lin; Heng-Yuan Lee; Jian-Ii Su; Yuan Xie; Shyh-Shyuan Sheu; Meng-Fan Chang

16. ITRS Roadmap. International technology roadmap for semiconductors. Semiconductor Industry Association, 2017.
17. Y. Li, P. Yuan, L. Fu, R. Li, X. Gao, C. Tao, " Coexistence of diode-like volatile and multilevel nonvolatile resistive switching in a ZrO 2 /TiO 2 stack structure ", Nanotechnology, vol. 26, no. 39, pp. 391001, Sep. 2015.
18. M. Prezioso, F. Merrikh, B. Hoskins, K. Likharev and D. Strukov, “Self-adaptive spike-time-dependent plasticity of metal-oxide memristors”, 2015, arXiv preprint arXiv:1505.05549
19. C. Xu, D. Niu, N. Muralimanohar, R. Balasubramonian, T. Zhang, S. Yu, Y. Xie, "Overcoming the challenges of crossbar resistive memory architectures", High Performance Computer Architecture (HPCA) 2015 IEEE 21st International Symposium on. IEEE, pp. 476-488, 2015.
20. H. Zhang, N. Xiao, F. Liu, Z. Chen, "Leader: Accelerating ReRAM-based main memory by leveraging access latency discrepancy in crossbar arrays", DATE, pp. 756-761, 2016.

21. Y. Shi, C. Pan, V. Chen, N. Raghavan, et. Al, “Coexistence of volatile and nonvolatile resistive switching in 2D h-bn based electronic synapses,” IEDM pp.119-122, 2017.

22. W Banerjee, Q Liu, H Lv, S Long, M Liu, “Electronic imitation of behavioral and psychological synaptic activities using TiO x/Al 2 O 3-based memristor devices,” Nanoscale 9 (38), 14442-14450.

23. J. Jeffers, J. Reinders, and A. Sodani, “Knights landing overview, ”Intel Xeon Phi Processor High Performance Programming, pp. 15–24, 2016.

24. Katam, N. K., Mukhanov, O. A., & Pedram, M. (2018). Superconducting Magnetic Field Programmable Gate Array. *IEEE Transactions on Applied Superconductivity*, *28*(2), 1–12. doi: 10.1109/tasc.2018.2797262.

25. M. Jagasivamani, C. Walden, D. Singh, L. Kang, S. Li, M. Asnaashari, S. Dubois, D. Yeung, B. Jacob. “Design for ReRAM-based Main-Memory Architectures." In Proceedings of the International Symposium on Memory Systems, Washington D.C., October 2019.

26. Lei Wang, CiHui Yang, Jing In, and Shan Gai, Emerging Nonvolatile Memories to Go Beyond Scaling Limits of Conventional CMOS Nanodevices, Journal of Nanomaterials, vol. 2014, Article ID 927696, 10 pages, 2014.

27. Ielmini, D. (2016). Resistive switching memories based on metal oxides: Mechanisms, reliability and scaling. Semiconductor Science and Technology,31(6), 063002. doi:10.1088/0268- 1242/31/6/063002

28. ReRAM Memory — Crossbar. (n.d.). Retrieved from https://crossbar-inc.com/en/

29. Jakub Szefer, Principles of Secure Processor Architecture Design, in Synthesis Lectures on Computer Architecture, Morgan Claypool Publishers, October 2018.

30. J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten.2009. Lest I remember: cold-boot attacks on encryption keys. Commun. ACM 52, 5 (May 2009), 91-98. DOI: https://doi.org/10.1145/1506409.1506429.

31. M. Jagasivamani, C. Walden, D. Singh, L. Kang, S. Li, M. Asnaashari, S. Dubois, B. Jacob, and D. Yeung. "Analyzing the Monolithic Integration of a ReRAM-based Main Memory into a CPU's Die.", in IEEE Micro (Special Issue on Monolithic 3D Architectures), November/December 2019.

32. Ambrosi, E., Bricalli, A., Laudato, M., Ielmini, D. (2019). Impact of oxide and electrode materials on the switching characteristics of oxide ReRAM devices. Faraday Discussions, 213, 8798. doi: 10.1039/c8fd00106e

33. Nail, C., Molas, G., Blaise, P., Piccolboni, G., Sklenard, B., Cagli, C., … Perniola, L. (2016). Understanding RRAM endurance, retention and window margin trade-off using experimental results and simulations. 2016 IEEE International Electron Devices Meeting (IEDM). doi: 10.1109/iedm.2016.7838346

34. Zhang, Y., Feng, D., Liu, J., Tong, W., Wu, B., Fang, C. (2017). A Novel ReRAM-based Main Memory Structure for Optimizing Access Latency and Reliability. Proceedings of the 54th Annual Design Automation Conference 2017 on – DAC 17. doi:10.1145/3061639.3062191.

35. Prezioso, M., Bayat, F. M., Hoskins, B., Likharev, K., Strukov, D. (2016). Self-Adaptive Spike-Time-Dependent Plasticity of Metal-Oxide Memristors. Scientific Reports,6(1). doi:10.1038/srep21331

36. Ge, J., & Chaker, M. (2017). Oxygen Vacancies Control Transition of Resistive Switching Mode in Single-Crystal TiO2 Memory Device. ACS Applied Materials & Interfaces,9(19), 16327-16334. doi:10.1021/acsami.7b03527.

37. B. Akin, C. Chou, J. Park, C. J. Hughes, and R. Agarwal, “Dynamic fine-grained sparse memory accesses," in Proceedings of the International Symposium on Memory Systems, MEMSYS '18, (New York, NY, USA), pp. 85-97, ACM, 2018.

38. N. Moussa, F. Nasri, and R. Tourki, “Noc architecture comparison with network simulator ns2," International Journal of Engineering Trends and Technology, vol. 13, no. 7, pp. 340-346, 2014.

39. J. Kim, W. J. Dally, S. Scott, and D. Abts, “Technology-driven, highly-scalable dragonfly topology,"2008 International Symposium on Computer Architecture, 2008.

40. A. F. Rodrigues, R. C. Murphy, P. Kogge, and K. D. Underwood, “Poster reception|the structural simulation toolkit," Proceedings of the 2006 ACM/IEEE conference on Supercomputing - SC 06, 2006.

41. Crossbar Inc., “Crossbar ReRAM Technology White Paper." 2017.

42. N. Jain, A. Bhatele, S. White, T. Gamblin, and L. V. Kale, “Evaluating hpc networks via simulation of parallel workloads," SC16: International Conference for High Performance Computing, Networking, Storage and Analysis, 2016.

43. A. Bhatele, N. Jain, Y. Livnat, V. Pascucci, and P.-T. Bremer, “Analyzing network health and congestion in dragonfly-based supercomputers," 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2016.

44. J. Seo and B. Kim, “Read margin analysis in an reram crossbar array," 2016 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS), 2016.

45. S. V and N. Chiplunkar, “Design and implementation of mesh and torus for network on chip based system," 2015 International Conference on Trends in Automation, Communications and Computing Technology (I-TACT-15), 2015.

46. M. M. Kim, J. D. Davis, M. Oskin, and T. Austin, “Polymorphic on-chip networks," 2008 International Symposium on Computer Architecture, 2008.

47. M. M. Kim, M. Mehrara, M. Oskin, and T. Austin, “Architectural implications of brick and mortar silicon manufacturing," Proceedings of the 34th annual international symposium on Computer architecture – ISCA 07, 2007.

48. X. Liu, S. Mohanraj, M. Pioro, and D. Medhi, “Multipath routing from a trac engineering perspective: How beneficial is it?," pp. 143-154, 10 2014.

49. R. Marculescu, U. Y. Ogras, L.-S. Peh, N. E. Jerger, and Y. Hoskote, “Outstanding research problems in noc design: System, microarchitecture, and circuit perspectives," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 28, no. 1, pp. 3-21, 2009.

50. K. S. Solnushkin, “Automated design of two-layer fat-tree networks." http://arxiv.org/abs/1301.6179, January 2013. arXiv:1301.6179.

51. K. S. Solnushkin, “Automated design of torus networks." http://arxiv.org/abs/1301.6180, January 2013.arXiv:1301.6180.

52. F. J. Andujar, S. Coll, M. Alonso, P. Lopez, and J.-M. Martinez, “Powar," vol. 15, pp. 1-22, 2019.

53. Z. Wang and S. Ma, Networks-on-chip: from implementations to programming paradigms. Morgan Kaufmann, 2015.

54. Kannan, S., Karimi, N., Sinanoglu, O., Karri, R. (2015). Security Vulnerabilities of Emerging Nonvolatile Main Memories and Countermeasures. IEEE Transactions on Computer- Aided Design of Integrated Circuits and Systems,34(1), 2-15. doi:10.1109/tcad.2014.2369741

55. Shewmon, P. (1989). *Diffusion in solids*. Warrendale, PA: Minerals, Metals & Materials Society.

56. Bertaud, T., Sowinska, M., Walczyk, D., Walczyk, C., Kubotsch, S., Wenger, C., & Schroeder, T. (2012). Resistive switching of Ti/HfO2-based memory devices: impact of the atmosphere and the oxygen partial pressure. *IOP Conference Series: Materials Science and Engineering*, *41*, 012018.

Appendix A: Cadence Encounter Command File

The following is the final Cadence Encounter command file script used to implement the four-cluster blockage regions and perform the APR to generate the final layout.

```
##########################################
# Cadence Encounter Command File
# Design: Integrated ReRAM with Scaled VSCALE Processor
# Meenatchi Jagasivamani 2018

set_global _enable_mmmc_by_default_flow      $CTE::mmmc_default
suppressMessage ENCEXT-2799
getVersion
win
set ::TimeLib::tsgMarkCellLatchConstructFlag 1
set conf_in_tran_delay {120.0ps}
# allow for random io placement

set defHierChar {/}
set distributed_client_message_echo {1}
set init_assign_buffer {0}

set init_gnd_net {VSS}
set init_pwr_net {VDD}

# Read in Libraries
set init_lef_file
{/homes/mjagasiv/research/Nangate/NangateOpenCellLibrary_PDKv1_3_v2010_
12/Back_End/lef/NangateOpenCellLibrary.lef}

# Read in design netlist
set init_top_cell {vscale_core256}
set init_verilog {vscale_core256.vh}


set lsgOCPGainMult 1.000000
set opt_buf_footprint {buf}
set opt_delay_footprint {buf}
set opt_inv_footprint {inv}
set pegDefaultResScaleFactor 1.000000
set pegDetailResScaleFactor 1.000000
set timing_library_float_precision_tol 0.000010
set timing_library_load_pin_cap_indices {}
set tso_post_client_restore_command {update_timing ;
write_eco_opt_db ;}

set init_mmmc_file vscale_core256.view

####### Begin Initializing Design
init_design
getIoFlowFlag
setIoFlowFlag 0

# Specify Floorplan
floorPlan -site FreePDK45_38x28_10R_NP_162NW_34O -s 1000 1000  20 20 20
20

# Specify blockage for standard-cell placement and Metal Layers
# Four separate clusters
createPlaceBlockage -box        289.5   628     507.5   655
createPlaceBlockage -box        385     532.5   412     750.5
createRouteBlk -box     291.5   630     505.5   653      -layer 1 2 3
4 5 6 7 8 9 10
createRouteBlk -box     387     534.5   410     748.5    -layer 1 2 3
4 5 6 7 8 9 10

createPlaceBlockage -box        532.5   628     750.5   655
createPlaceBlockage -box        628     532.5   655     750.5
createRouteBlk -box     534.5   630     748.5   653      -layer 1 2 3
4 5 6 7 8 9 10
createRouteBlk -box     630     534.5   653     748.5    -layer 1 2 3
4 5 6 7 8 9 10

createPlaceBlockage -box        289.5   385     507.5   412
createPlaceBlockage -box        385     289.5   412     507.5
createRouteBlk -box     291.5   387     505.5   410      -layer 1 2 3
4 5 6 7 8 9 10
createRouteBlk -box     387     291.5   410     505.5    -layer 1 2 3
4 5 6 7 8 9 10

createPlaceBlockage -box        532.5   385     750.5   412
createPlaceBlockage -box        628     289.5   655     507.5
```

```
createRouteBlk -box     534.5   387     748.5   410      -layer 1 2 3
4 5 6 7 8 9 10
createRouteBlk -box     630     291.5   653     505.5    -layer 1 2 3
4 5 6 7 8 9 10

# Global VDD and VSS nets
clearGlobalNets
globalNetConnect VDD -type pgpin -pin VDD -inst * -all -override
globalNetConnect VDD -type tiehi -inst * -all -override
globalNetConnect VSS -type pgpin -pin VSS -inst * -all -override
globalNetConnect VSS -type tielo -inst * -all -override

set sprCreateIeRingNets {}
set sprCreateIeRingLayers {}
set sprCreateIeRingWidth 1.0
set sprCreateIeRingSpacing 1.0
set sprCreateIeRingOffset 1.0
set sprCreateIeRingThreshold 1.0
set sprCreateIeRingJogDistance 1.0

# Create power & ground rings
addRing -skip_via_on_wire_shape Noshape -skip_via_on_pin Standardcell -
center 1 -stacked_via_top_layer metal10 -type core_rin
gs -jog_distance 0.8 -threshold 0.8 -nets {VSS VDD} -follow core -
stacked_via_bottom_layer metal1 -layer {bottom metal9 top m
etal9 right metal10 left metal10} -width 5 -spacing 5 -offset 0.8

set sprCreateIeStripeNets {}
set sprCreateIeStripeLayers {}
set sprCreateIeStripeWidth 10.0
set sprCreateIeStripeSpacing 2.0
set sprCreateIeStripeThreshold 1.0

# Create power grid
addStripe -skip_via_on_wire_shape Noshape -block_ring_top_layer_limit
metal10 -max_same_layer_jog_length 1.6 -padcore_ring_bo
ttom_layer_limit metal9 -number_of_sets 3 -skip_via_on_pin Standardcell
-stacked_via_top_layer metal10 -padcore_ring_top_laye
r_limit metal10 -spacing 2 -merge_stripes_value 0.095 -layer metal10 -
block_ring_bottom_layer_limit metal10 -width 2 -nets {V
DD VSS} -stacked_via_bottom_layer metal9

# Connect Power and Ground nets
createPGPin -net VDD VDD
createPGPin -net VSS VSS
sroute -connect { blockPin padPin padRing corePin floatingStripe } -
layerChangeRange { metal1 metal10 } -blockPinTarget { nea
restTarget } -padPinPortConnect { allPort oneGeom } -padPinTarget
{ nearestTarget } -corePinTarget { firstAfterRowEnd } -floa
tingStripeTarget { blockring padring ring stripe ringpin blockpin
followpin } -allowJogging 1 -crossoverViaLayerRange { metal
1 metal10 } -nets { VDD VSS } -allowLayerChange 1 -blockPin useLef -
targetViaLayerRange { metal1 metal10 }

# Place standard cells
setPlaceMode -fp false
placeDesign
checkPlace
checkPinAssignment

# Route signals
trialRoute -maxRouteLayer 8 -highEffort

# Apply timing constraints
####### timing route -- preCTS
create_constraint_mode -name SDCvscale_core256 -sdc_files
{vscale_core256.sdc}
create_analysis_view -name typ -constraint_mode {SDCvscale_core256} -
delay_corner {default}
set_analysis_view -setup {typ} -hold {typ}
timeDesign -preCTS -idealClock -pathReports -drvReports -slackReports -
numPaths 50 -prefix v_core256_preCTS -outDir timingRep
orts
setEndCapMode -reset
setEndCapMode -boundary_tap false

######### clock tree synthesis
createClockTreeSpec -bufferList {CLKBUF_X1 CLKBUF_X2 CLKBUF_X3 BUF_X1
BUF_X16 BUF_X2 BUF_X32 BUF_X4 BUF_X8 } -output clock.sp
ec -routeClkNet
specifyClockTree -clkfile clock.spec
setCTSMode -engine ck
clockDesign -specFile clock.spec -outDir clock_report -
fixedInstBeforeCTS

######## RC extraction
setDrawView place
timeDesign -reportOnly -pathReports -drvReports -slackReports -numPaths
50 -prefix v_core256 -outDir timingReports
setNanoRouteMode -quiet -timingEngine {}
setNanoRouteMode -quiet -routeWithTimingDriven 1
setNanoRouteMode -quiet -routeWithSiDriven 1
setNanoRouteMode -quiet -routeWithSiPostRouteFix 0
setNanoRouteMode -quiet -drouteStartIteration default
setNanoRouteMode -quiet -routeTopRoutingLayer default
setNanoRouteMode -quiet -routeBottomRoutingLayer default
setNanoRouteMode -quiet -drouteEndIteration default
setNanoRouteMode -quiet -routeWithTimingDriven true
setNanoRouteMode -quiet -routeWithSiDriven true
#setNanoRouteMode routeWithECO true
```

```
snapFPlan -guide -block -stdCell -pinBlk -ptnCore -placeBlk -macroPin -
pin
globalDetailRoute

# Add filler cells and perform parasitic extraction
getFillerMode -quiet
addFiller -cell FILLCELL_X2 FILLCELL_X4 FILLCELL_X8 FILLCELL_X16
FILLCELL_X32 -prefix FILL_
extractRC
timeDesign -postRoute -pathReports -drvReports -slackReports -numPaths
50 -prefix v_core256_postRoute -outDir timingReports

########## power measurement
set_power_analysis_mode -reset
set_power_analysis_mode -method static -corner max -create_binary_db
true -write_static_currents true -honor_negative_energy
true -ignore_control_signals true
set_power_output_dir -reset
set_power_output_dir ./
set_default_switching_activity -reset
set_default_switching_activity -input_activity 0.2 -period 10.0
read_activity_file -reset
set_power -reset
set_powerup_analysis -reset
set_dynamic_power_simulation -reset
report_power -rail_analysis_format VS -outfile .//v_core256.rpt

############## verify connectivity (LVS)
verifyConnectivity -type all -noAntenna -error 1000 -warning 50

##########  verify geometry (DRC)
setVerifyGeometryMode -area { 0 0 0 0 } -minWidth true -minSpacing true
-minArea true -sameNet true -short true -overlap true
 -offRGrid false -offMGrid true -mergedMGridCheck true -minHole true -
implantCheck true -minimumCut true -minStep true -viaEn
closure true -antenna false -insuffMetalOverlap true -pinInBlkg false -
diffCellViol true -sameCellViol false -padFillerCellsO
verlap true -routingBlkgPinOverlap false -routingCellBlkgOverlap true -
regRoutingOnly false -stackedViasOnRegNet false -wireE
xt true -useNonDefaultSpacing false -maxWidth true -maxNonPrefLength -1
-error 1000

verifyGeometry

######## Rerun detailed route with ECO mode
setNanoRouteMode -quiet -routeWithEco 1
setNanoRouteMode -quiet -drouteStartIteration default
setNanoRouteMode -quiet -routeTopRoutingLayer default
setNanoRouteMode -quiet -routeBottomRoutingLayer default
setNanoRouteMode -quiet -drouteEndIteration default
setNanoRouteMode -quiet -routeWithTimingDriven true
setNanoRouteMode -quiet -routeWithSiDriven true
routeDesign -globalDetail
verifyGeometry
setLayerPreference violation -isVisible 1
```

```
violationBrowser -all -no_display_false
verifyConnectivity -type all -noAntenna -error 1000 -warning 50

######### Timing report
timeDesign -reportOnly -pathReports -drvReports -slackReports -numPaths
50 -prefix v_core256 -outDir timingReports

###### Stream out and save design
streamOut v_core256.gds -
mapFile ../FreePDK45/osu_soc/lib/files/gds2_encounter.map -libName
libJuly5 -structureName v_core256
 -stripes 1 -units 1000 -mode ALL
saveNetlist v_core256_PR.v
saveDesign v_core256.enc
summaryReport -noHtml -outfile summaryReport.rpt
reportGateCount -level 5 -limit 100 -outfile v_core256.gateCount
reportNetStat
##########################################
```

Appendix B: MAC Javascript Source Code

The following is the Javascript source code for the Monolithic Architecture Calculator (MAC) mentioned in Section 4.6.

```
<html>
<head>
   <title>Monolithic Architecture Calculator</title>
<style>
p {
    font-family: Tahoma, Geneva, sans-serif;
}

table, th, td {
    margin-left:15px;
    border: 1px solid black;
    border-collapse: collapse;
}
th, td {
    padding: 10px;
}
* {
    box-sizing: border-box;
     font-family: Tahoma, Geneva, sans-serif;
}

.row {
    display: flex;
    border-style: double;
    border-width: thick;
}
.input_form {
    padding:0px;
    margin:0px;
    margin-left: 25px;
}
```

```
/* Create two equal columns that sits next to each other */
.column {
    flex: 50%;
    padding: 10px;
    border-style: double;
    border-width: thin;

}
.slidecontainer {
    margin-left: 25px;
}

.slider {
    -webkit-appearance: none;
    width: 50%;
    height: 10px;
    //border-radius: 5px;
    background: #d3d3d3;
    outline: none;
    opacity: 0.7;
    -webkit-transition: .2s;
    transition: opacity .2s;
}

.slider:hover {
    opacity: 1;
}

.slider::-webkit-slider-thumb {
    -webkit-appearance: none;
    appearance: none;
    width: 25px;
    height: 25px;
   // border-radius: 50%;
    background: #537d9b;
    cursor: pointer;
}

.slider::-moz-range-thumb {
    width: 25px;
    height: 25px;
 //   border-radius: 50%;
    background: #537d9b;
    cursor: pointer;
}

</style>
<script language=javascript type="text/javascript">
```

```
function round(value, precision) {
    var multiplier = Math.pow(10, precision || 0);
    return Math.round(value * multiplier) / multiplier;
}
function calculate(){
    var memSelected = document.getElementById('memory_type');
    var networkSelected = document.getElementsByName('networkT');
    var procSelected = document.getElementsByName('procT');
    var procT = document.getElementById('procT');
    var diesize = document.getElementById("diesize");
    var numMCs = document.getElementById("Input_numMC").value;
    var slideCol = document.getElementById("cache_ratio");
    var cache_ratio = slideCol.value;
    var l2l3_ratio = slideCol2.value;

      //var cache_ratio = document.getElementById("cache_ratio").value;

    var availArea=diesize.value;
      //calculate available area based on memory type selected
    var CPoverhead = 0;
      if (memSelected.value == "NVR") {
        //charge pump % area -> typical flash ratio
        //Reference:  7% of 16nm Micron NAND flash diesize of 173 mm^2
        CPOverhead=0.07*173;
        availArea = availArea-CPoverhead;
      } else if (memSelected.value == "dram") {
        //no charge pump, but external Memory controller interface logic
        // Typical 10% area
        //Reference:  10% of 16nm Micron NAND flash diesize of 173 mm^2
        CPOverhead=0.1*173;
        availArea = availArea-CPoverhead;
      } else if (memSelected.value == "VR") {
        //charge pump % area -> half of typical flash ratio
        //Reference:  3.5% of 16nm Micron NAND flash diesize of 173 mm^2
        CPOverhead=0.035*173;
        availArea = availArea-CPoverhead;
      }

    //calculate NoC overhead
      if (networkSelected[0].checked) {
        //Mesh -- upto 4 links per node
        //TODO placeholder 10% overhead
        availArea = availArea*0.9;

      } else if (networkSelected[1].checked) {
        //Hoffman-Singleton -- upto 3 links per node
        //TODO placeholder 7% overhead
        availArea = availArea*0.93;
      }

    //Calculate available SRAM Cache area
      //Reference:  IEDM 2013, TSMC 6T SRAM bit cell area for 16nm: 0.07
sq. micron
      //Using typical value of 70% array efficiency for overhead circuitry
      var SRAMcellsize=0.07;

      var SRAMarea=(1-cache_ratio)*availArea*1000000*0.7;
      var SRAMsize = SRAMarea/(SRAMcellsize*1024*1024*8);
      document.getElementById("Total_Cache").innerHTML              =
round(SRAMsize,1)+" MB";
      document.getElementById("L2").innerHTML          =          "L2:
"+round(SRAMsize*l2l3_ratio,1)+" MB";
      document.getElementById("L3").innerHTML          =          "L3:
"+round(SRAMsize*(1-l2l3_ratio),1)+" MB";

      //Fixed  32kB  size  for  Queuing  buffer  of  Memory  controller  --
supports 512 depth of 512 bits
      var MCBuffer_area=(SRAMcellsize*32*1024*8/1000000)/0.7;

    //set processor area
    var procSize = 0.0284077;
    if (procSelected[0].checked) {
        //Raven-3 RISC-V: Scaled 0.46x from 28nm node
        procSize = 0.46*1.19; //mm^2
      } else if (procSelected[1].checked) {
        //Fujitsu Sparc64 X+: Scaled 0.46x from 28nm node
        procSize = 0.46*10.907; //mm^2
      } else if (procSelected[2].checked) {
        //Intel Skylake-X CPU core (14nm)
        procSize = 16.9; //mm^2
      } else if (procSelected[3].checked) {
        //Intel Xeon-Phi core -- knights landing (14nm)
        procSize = 3.13; //mm^2
      }

      var Procarea= cache_ratio*availArea;
    var numProcessors = Procarea/procSize; //initial value ignoring ReRAM
congestion penalty
```

```
    //calculate Main-Memory bandwidth, latency
    //for ReRAM (default), bw=8*4 bits per memory controller
    var bw = 32*numMCs;
    var rLatency = 200;
    var wLatency = 1000;

      //Power numbers derived based on McPAT memory access model
      //Crossbar for ReRAM cell energy number
      var rPwr=4.24; //W
      var wPwr=41.86; //W

   //Calculate # of Mem Controllers under SRAM
    if (memSelected.value == "dram") {
        //charge pump % area -> typical flash ratio
        //Reference:  7% of 16nm Micron NAND flash diesize of 173 mm^2
        document.getElementById("Total_ReRAM").innerHTML = "External MM
";
        document.getElementById("Total_ReRAM2").innerHTML = " ";
        document.getElementById("Total_ReRAM8").innerHTML = " ";

            //128 bits per memory controller
        bw = 128*numMCs;
        rLatency = 55;
        wLatency = 55;
            rPwr=0; //W
            wPwr=0; //W

      }  else if (numMCs>0) {

            //calculate ReRAM storage amount
            //Assumptions:  50% array efficiency
            //Cell-size at 16nm: 0.011236 sq-um at 45nm --> scales down
by 10x at 16nm
            //Reference: Crossbar ReRAM
            // Avail area =  with 25% periphery overhead +  congestion
penalty

            var MMArea = availArea*0.5;  //50% array efficiency
            var ReRAMcellsize=0.011236/10;
            var
ReRAMsize=MMArea*1000000/(ReRAMcellsize*1024*1024*1024*8);
            //check if numMemory Controllers entered is invalid
            var MMGranularity=ReRAMsize*2*1024/numMCs;

            /////////////Calculat           congestion           penalty
////////////////////
            var congestion = 0.12;  //congestion penalty is function of
# of MCs

            //remove RERAM periphery circuit area -- applies to Proc and
Cache
```

```
            availArea = availArea*(1-0.5*0.25);
            //remove Memory-Controller queuing buffer area
            availArea = availArea-(numMCs*MCBuffer_area);

            if (numMCs <= (availArea/procSize)) {
                  congestion = 0.12*(numMCs/(availArea*0.88/procSize));
//numMCs/numProcTiles (approx)
            } else {
                  congestion                                          =
congestion+0.04*Math.log2((numMCs/(availArea/procSize)));
//numMCs/numProcTiles (approx)
                  var   tmpArea      =   availArea*(1-congestion);   //not
available for either proc or sram
                  congestion                                          =
congestion+0.04*Math.log2((numMCs/(tmpArea/procSize)));
//numMCs/numProcTiles (approx)
            }
            if                                              (congestion<0)
document.getElementById("debug").innerHTML = " **** ERROR:    INVALID
Congestion Penalty parameter **** " ;
            //document.getElementById("debug").innerHTML       =       "
numproctiles="+(availArea*0.88/procSize)+" MMarea="+MMArea;
            //document.getElementById("debug").innerHTML       =       "
MCBuffer_area="+MCBuffer_area*numMCs;


            //recalculate after removing wasted area for congestion
            availArea  =  availArea*(1-congestion);  //not  available  for
either proc or sram
            ReRAMsize=MMArea*1000000/(ReRAMcellsize*1024*1024*1024*8);
            MMGranularity=ReRAMsize*2*1024/numMCs;

            if (MMGranularity > 1) {
                  document.getElementById("Total_ReRAM").innerHTML    =
round(ReRAMsize,1)+" GB ";//congestion="+congestion;
                  document.getElementById("Total_ReRAM2").innerHTML   =
round(ReRAMsize*2,1)+" GB for 2-layer stack ";
                  document.getElementById("Total_ReRAM8").innerHTML   =
round(ReRAMsize*8,1)+" GB for 8-layer stack";
            } else {
                  document.getElementById("Total_ReRAM").innerHTML    =
"ERROR -- too high # of Memory Controllers";
                  document.getElementById("Total_ReRAM2").innerHTML = "
";
                  document.getElementById("Total_ReRAM8").innerHTML = "
";
                  bw=0;
                  rLatency=0;
```

```
                 wrLatency=0;
            }

            //update num processors after considering reram congestion
penalty
            Procarea= cache_ratio*availArea;
            numProcessors = Procarea/procSize;
    } else {
        document.getElementById("Total_ReRAM").innerHTML = "0 GB ";
        document.getElementById("Total_ReRAM2").innerHTML = "0 GB ";
        document.getElementById("Total_ReRAM8").innerHTML = "0 GB ";
            bw=0;
            rLatency=0;
            wLatency=0;
      }

      if (availArea <= 0) {
            document.getElementById("NumMC").innerHTML = "ERROR --
Invalid Configuration";
            document.getElementById("NumProc").innerHTML = " ";
      } else {
            document.getElementById("NumMC").innerHTML =
round(numMCs,0);
            document.getElementById("NumProc").innerHTML =
round(numProcessors,0);
      }
    //calculate number of NoC controllers: 1 per tile
    var numNoCs = numMCs;
      if (networkSelected[0].checked) {
        //Mesh -- upto 4 links per node
        //TODO placeholder 10% overhead
        numNoCs = numMCs;
      } else if (networkSelected[1].checked) {
        //Hoffman-Singleton -- upto 3 links per node
        //TODO placeholder 7% overhead
        numNoCs = numMCs*0.75;
      }
    // document.getElementById("NumNoc").innerHTML = round(numNoCs,0);


    //add NoC Latency
      if (networkSelected[0].checked) {
        //Mesh -- upto 4 links per node
        //TODO placeholder 10% overhead
        //rLatency = rLatency * 1.1;
        //wLatency = wLatency * 1.1;
      } else if (networkSelected[1].checked) {
        //Hoffman-Singleton -- upto 3 links per node
        //TODO placeholder 7% overhead
        //rLatency = rLatency * 1.07;
        //wLatency = wLatency * 1.07;
      }
    //convert to bytes
    bw=round(bw,0)
    bw = bw/8;
    document.getElementById("bw").innerHTML = round(bw,2);
    document.getElementById("latency").innerHTML = round(rLatency,2)+"
ns & "+round(wLatency,2);

    //calculate power consumption for Processor, MM, SRAM
    pwr = 0; // unit mW/Hz
    performance = 0;
    //for Processor:
    if (procSelected[0].checked) {
        //RISC-V Ref IEEE Micro 2016: 961MHz, 173mW, 34GFLOPS/w
        //pwr = 0.173*numProcessors/961; //w/MHz
        pwr = 0.173*numProcessors; //w
        performance = 34; //GFLOPS/W
      } else if (procSelected[1].checked) {
        //Sparc Ref: Fujitsu Sparc64 X+ wikipedia 392W, 448GFLOPS, 3.5GHz
        //pwr = 392*(numProcessors/16)/3500;  //w/MHz
        pwr = 392*(numProcessors/16);  //w
        performance = 448/392; //GFLOPS/W at 1.4GHz
      } else if (procSelected[2].checked) {
        //Intel Skylake-X CPU core (14nm) spec: 165W, 1152 GFLOPS at
4.4GHz (turbo)
        //pwr = 165*(numProcessors/18)/4400; //w/MHz
        pwr = 165*(numProcessors/18); //w
        performance = 1152/165; //GFLOPS/W at 1GHz
      } else if (procSelected[3].checked) {
        //Intel Knights Landing/Mill Xeon Phi  (14nm) spec: 260W, 3456
GFLOPS at 1.7GHz (turbo)
        //pwr = 260*(numProcessors/72)/1700;  //w/MHz
        pwr = 260*(numProcessors/72);  //w
        performance = 3456/260; //GFLOPS/W at 1GHz
      }
    document.getElementById("pwr").innerHTML      =      "Processor:
"+round(pwr,3)+" w";
      if (memSelected.value == "dram")
            document.getElementById("pwr_MM").innerHTML = " ";
      else
            document.getElementById("pwr_MM").innerHTML = "Main-Memory:
Read: "+round(rPwr,2)+" W & Write: "+round(wPwr,2)+" W";
```

```
    document.getElementById("performance").innerHTML              =
round(performance,2)+" GFLOPS/W ";

}
</script>

</head>

<body>
<h2>Monolithic Architecture Calculator (MAC)</h2>

<b>Purpose:</b>  Preliminary estimates on what can "fit" in a given chip
dimension and assess architectural tradeoffs with various design options
on a Memory-Processors System.

<br> Process Node: 16nm<br><br>


<div class="row" >

<div class="column"  >
<form name="calc" action="post">
<p>
1) Main Memory Type:
<select id="memory_type">
    <option value=NVR>Non-Volatile ReRAM </option>
    <option value=dram>DRAM</option>
    <option value=VR>Volatile ReRAM</option>
</select>
<br>
<br>

2) Die-Size [mm^2]: <input type=text id="diesize" size=10 value="686">
<br>
  <div class="input_form"></div>

3) Select Processor Core Type:
  <div class="input_form">
    <input type="radio" id="riscv"
     name="procT" value="riscv">
    <label for="riscv">Raven-3 RISC-V w/56kB L1 per core </label><br>

    <input type="radio" id="sparc"
     name="procT" value="sparc">
    <label for="sparc">Sparc64 XII w/128kB L1 per core </label><br>

    <input type="radio" id="gpu"
     name="procT" value="gpu">
    <label for="gpu">Intel Skylake-X w/64kB L1 per core  </label><br>

    <input type="radio" id="knl"
     name="procT" value="knl" checked="checked">
    <label for="gpu">Intel Xeon Phi (Knights Landing) w/32kB L1 per core
</label><br>
  </div>
  <br>
```

```
4) Core-to-Cache Area ratio for SRAM Cache on Chip (0 to 1):<br>
  (0:all SRAM; 1: all Processor, no SRAM): <br>
<div class="slidecontainer">
   <input  type="range"  min="0"  max="1"  step="0.05"  value="0.85"
id="cache_ratio" class="slider">
  Value: <span id="f" style="font-weight:bold;"> </span>
<script>
var slideCol = document.getElementById("cache_ratio");
var y = document.getElementById("f");
y.innerHTML = slideCol.value;

slideCol.oninput = function() {
    y.innerHTML = this.value;
}

</script>

</div>
<br>

5)  Number  of  Memory  Controllers:  <input  type=text  id="Input_numMC"
size=10 value="131"> <br>
  <div class="input_form"></div>
  <br>


6) Network Topology:
  <div class="input_form">

    <input type="radio" id="mesh"
     name="networkT" value="mesh" checked="checked">
    <label for="mesh">Mesh network</label><br>
     <!---
    <input type="radio" id="singleton"
     name="networkT" value="singleton" disabled>
    <label for="singleton">Hoffman-Singleton Graph</label><br>
      -->
  </div>
  <br>

-------------------------------------------------------<p
align="center">
<input type=button value="CALCULATE" onClick="calculate()"></p>
</form>
</div>
<div class="column" >
 Final Architecture Configuration for given die-size and components:<br>
<a id="debug"></a>
<table>
  <tr>
      <th>Parameter</th>
      <th>Value</th>
  </tr>
  <tr>
      <td>Total Main Memory Storage Size </td>
      <td><a  id="Total_ReRAM"></a><br><a  id="Total_ReRAM2"></a>  <br><a
id="Total_ReRAM8"></a> <br> </td>
  </tr>
  <tr>
      <td>Total SRAM Cache Size
            <p>L2-to-L3 ratio: <p>
<div class="slidecontainer">
   <input type="range" min="0" max="1" step="0.05" value="1" id="l2_l3"
class="slider"> <p>
    <span id="l2l3" style="font-weight:bold;">  </span>
<script>
var slideCol2 = document.getElementById("l2_l3");
var y2 = document.getElementById("l2l3");
y2.innerHTML = slideCol2.value;

slideCol2.oninput = function() {
    y2.innerHTML = this.value;
}

</script>

</div>
      </td>
      <td>
      <a id="Total_Cache"></a> <br>
      <a id="L2"></a> <br>
      <a id="L3"></a>

</td>
  </tr>
  <tr>
      <td>Number of Processors <br>
    Number of Memory Controllers <br>
  <!--  Number of NoC Controllers <br> -->
    </td>
      <td><a id="NumProc"></a> <br>
        <a id="NumMC"></a> <br>
   <!--     <a id="NumNoc"></a> <br> -->
    </td>
  </tr>
  <tr>
      <td>Main Memory (MM) Bandwidth </td>
      <td><a id="bw"></a> bytes per access   </td>
  </tr>
  <tr>
      <td>MM Read & Write Latency </td>
      <td><a id="latency"></a> ns  </td>
  </tr>
  <tr>
      <td>Power Consumption </td>
      <td><a id="pwr"></a>   <br>
      <a id="pwr_MM"></a>    <br>
      </td>
  </tr>
  <tr>
      <td>Energy Efficiency </td>
      <td><a id="performance"></a>  </td>
  </tr>
</table>
</div>


</div>


<hr>

<div align="right"><small>
   University of Maryland, College Park, MD<br>
   Electrical and Computer Engineering<br>
   Memory Systems Research Group<br> <i>
  <br>Posted by: Meenatchi Jagasivamani
  <br><a href="mjagasiv@terpmail.umd.edu">
  mjagasiv@terpmail.umd.edu</a>.</i>
</small></div>

</body>

</html>
```