TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies
Department of Computer Systems

IASM02/15

Ameer Shalabi 177359IASM

# PERFORMANCE AND ENERGY EVALUATION OF NVM-BASED CIM AND MEMORY HIERARCHY

Master's Thesis

|  |  |
|---|---|
| Supervisor: | Jaan Raik |
|  | Professor |
| Co-Supervisor: | Kolin Paul |
|  | Adjunct Professor |

Tallinn 2019

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Arvutisüsteemide instituut

IASM02/15

Ameer Shalabi 177359IASM

# JÕUDLUSE JA ENERGIATARBE HINDAMINE MITTELENDUVAL MÄLUL PÕHINEVAL MÄLUSISESEL ARVUTUSEL JA MÄLUHIERARHIAL

Magistritöö

| | |
|---|---|
| Juhendaja: | Jaan Raik |
| | Professor |
| Kaasjuhendaja: | Kolin Paul |
| | Adjunct Professor |

Tallinn 2019

## Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Ameer Shalabi

2019-05-06

# Abstract

Non-Volatile Memory technologies are rapidly rising as the most promising candidate for universal memory technologies. NVMs are characterized with low-power leakage, high-density storage, and fast access time. NVMs offer solutions for the high standby mode power consumption that contemporary memory technologies suffer from. Furthermore, such NVMs can be programmed and designed to create NVM-based arrays capable of completing complex logical operations as Computation in Memory.

In this thesis, the Level-1 instructions cache is augmented with a Non-Volatile Scratch Pad that stores instructions that cause highest number of I-cache misses. When implementing the NV-SP using Magnetic RAM, it improved the performance for all the simulated applications at different sizes. Performance improvement reached up to 22% for applications with the highest miss rate. The MRAM NV-SP has been shown to improve the total access energy and total power leakage of the I-cache. However, when such NV-SP is implemented using PCRAM, it showed that it can cause performance degradation for application with low miss latency at large NV-SP sizes. It is also showed that PCRAM had reduced the total access energy of the I-cache for all the applications at all sizes, but increased the power leakage of the I-cache at larger NV-SP sizes. Furthermore, this thesis also presents an implementation of a memristor-based AES S-Box. Such implementation uses memristor-based *AND*, *OR*, and $XOR$ logic gates that perform Computation in Memory.

This thesis is written in English and is 78 pages long, including 5 chapters, 22 figures, and 7 tables.

# Annotatsioon

Mittelenduva mälu tehnoloogiad on kiiresti muutumas populaarseimateks universaalse mälutehnoloogia kandidaatideks. Mittelenduvat mälu iseloomustab madal võimsusleke, kõrge salvestustihedus ja kiire pöördusaeg. Ta pakub lahenduse kõrgele võimsustarbele ooterežiimis, mille all kannatavad moodsad mälutehnoloogiad. Samuti saab mittemuutlikke mälusid programmeerida ja projekteerida moodustamaks massiive, mis on võimelised mälusiseselt läbi viima keerukaid operatsioone.

Käesolevas lõputöös on 1. taseme käsu-vahemälu mugandatud mittelenduvaks märkmikuks (ingl. k. scratch pad), mis salvestab käske, mis puuduvad sagedimini vahemälust. Kui mittelenduv märkmik oli realiseeritud magneetilisel muutmälul (MRAM), siis jõudlus kasvas kõigi simuleeritud eri suurusega rakenduste puhul. Jõudluse kasv saavutas 22% rakenduse jaoks, millel oli kõrgeim arv käske, mis vahemälust puudusid. On näidatud, et MRAM mittelenduv märkmik parandab kogu pöörduseks kuluvat energiat ja kogu võimsusleket käsu-vahemälus. Samas osutus, et kui sarnane mittelenduv märkmik on realiseeritud kasutades PCRAM tehnoloogiat, siis väiksema vahemälu poole pöördumise ajakuluga rakenduste jõudlus langes suurte märkmiku mahtude puhul. Veelenam, käesolev töö esitab ka memristoridel põhineva AES-algoritmi S-box'i realisatsiooni. Realisatsioon kasutab memristoridel põhinevaid $JA$, $V\tilde{O}I$, ja $V\ddot{A}LISTAV - V\tilde{O}I$ loogikaelemente, mis võimaldavad mälusisest arvutust.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 78 leheküljel, 5 peatükki, 22 joonist, 7 tabelit.

# List of abbreviations and terms

| | |
|---|---|
| **AES** | Advanced Encryption Standard |
| **ARCFOUR** | Rivest Cipher 4 |
| **CiM** | Computation in Memory |
| **CISC** | Complex Instruction Set Computer |
| **CPU** | Central Processing Unit |
| **D-Cache** | Data Cache |
| **DES** | DATA Encryption Standard |
| **DRAM** | Dynamic Random Access Memory |
| **I-Cache** | Instruction Cache |
| **ICs** | Integrated Circuits |
| **ICT** | Information and communication technologies |
| **ISA** | Instruction Set Architecture |
| **L1 Caches** | First Level Caches |
| **MRAM** | Magnetic Random Access Memory |
| **MTJ** | Magnetic Tunnel Junction |
| **NV-SP** | Non-Volatile Scratch Pad |
| **NVM** | Non-Volatile Memory |
| **OS** | Operating System |
| **PCM** | Phase-Changing Memory |
| **PCRAM** | Phase-Change Random Access Memory |
| **RAM** | Random Access Memory |
| **ROT13** | Rotate by 13 places Cipher |
| **Sbox** | Substitution Box |

**SoC** System-on-Chip

**SRAM** Static Random Access Memory

**TAS** Thermally Assisted Switching Magnetic Random Access Memory

**TMR** Tunnelling MagnetoResistance

# Contents

# List of Figures

# List of Tables

# 1 Introduction

The discovery of semi-conductive materials had a monumental impact on our society. They are found in devices we use every day in the form of diodes, transistors, and everything that is computerized. The semiconductor revolutionized the world and we have found a use for it in everything around us. Information and communication technologies (ICT) are indispensable for many industries and form the backbone of modern society. At the core of ICT is the semiconductor that occupies the main position in areas of storage, working memories, and computer logic .

One of the most important uses of the semiconductor is the design of Integrated Circuits (ICs). ICs are used in the design and fabrication of every component of a computer system. Most notably, ICs live at the heart of the Central Processing Unit (CPU) and it's supporting systems. ICs made components smaller, cheaper, more reliable, and easier to repair. ICs made computerization possible.

Modern ICs, however, face major issues such as high power leakage [1] and performance degradation under high utilization [2]. Increased device variability and process complexity only inflate these issues further. In an attempt to combat these issues, designers put forth two goals in mind: systems should achieve fast process switching at low power when used in dynamic mode and low leakage power when systems are idle or in static mode.

Current computer systems employ different types of semiconductor-based memory to store information. Inside the processor, registers are made from Flip-Flops that have smaller capacity but are accessed at very high speeds. Outside the processor are the Static Random Access Memory (SRAM) caches. The caches have higher capacity that the registers but they are slower. The Dynamic Random Access Memory (DRAM) main memory, or Random Access Memory (RAM), is larger than the caches, but it is far slower. These volatile devices differ in speeds of access, but they all lose information that is stored in them when powered off.

Furthermore, there are two key issues in the current semiconductor-based memory: firstly, the rapid increase in density causes larger amounts of power consumption. The major factor here is that the standby power to retain the information compared to the operating power to perform information processing is high. This is due to the characteristic of transistors when miniaturized. This issue not only causes large amounts of power consumption, but it is also becoming an obstruction for further miniaturization of transistors [3]. Secondly, the speed gap between different levels of the memory hierarchy, shown in Figure 1 (a), cause bottlenecks at each level. For example, DRAM will always have several clock cycles of latency due to its operating principle while both the CPU and SRAM caches can operate at a frequency of 3 GHz [4].

To combat the first issue, designers were forced to implement circuit design techniques, such as power and clock gating [5], to reduce standby mode power consumption. However, a more concrete solution to this issue of power leakage can be the introduction of Non-Volatile devices into the system. Non-Volatile devices can hold the information stored in them even when the system is powered off or when power failures occur. More specifically, integrating Non-Volatile Memory (NVM) into the memory hierarchy allows information and logic state retention in case of a complete power down of the system and significantly reduces power usage and power leakage of the memory system and, in some cases, improves the overall performance of the system.

Since the memory systems' speed is not increasing as fast as the increase in processor speeds, processor execution rates are limited by the latency of accessing instructions and data in the memory. The on-chip caches offer a solution: a memory device inside the processor that has higher capacity than the registers and faster than the caches outside the CPU. Higher capacity meant a slower cache, but this issue of speed was addressed by introducing specialized caches known as the First Level (L1) Caches. The L1 Caches supported instruction and data fetch bandwidths of modern processors [6] which, in turn, helped minimize the overall latency by using an Instruction Cache (I-Cache) to store instructions before being executed by the processor and a Data Cache (D-Cache) to store data needed during the execution flow. For embedded System-on-Chip (SoC), a scratch pad memory was introduced to store smaller amounts of information that are constantly needed. Scratch Pad memories, unlike caches, hold information (either instructions or data) permanently and are manually configured. Scratch Pads do not have eviction policies, thus the information stored in them is never changed automatically and can be configured depending on the execution nature of the workload.

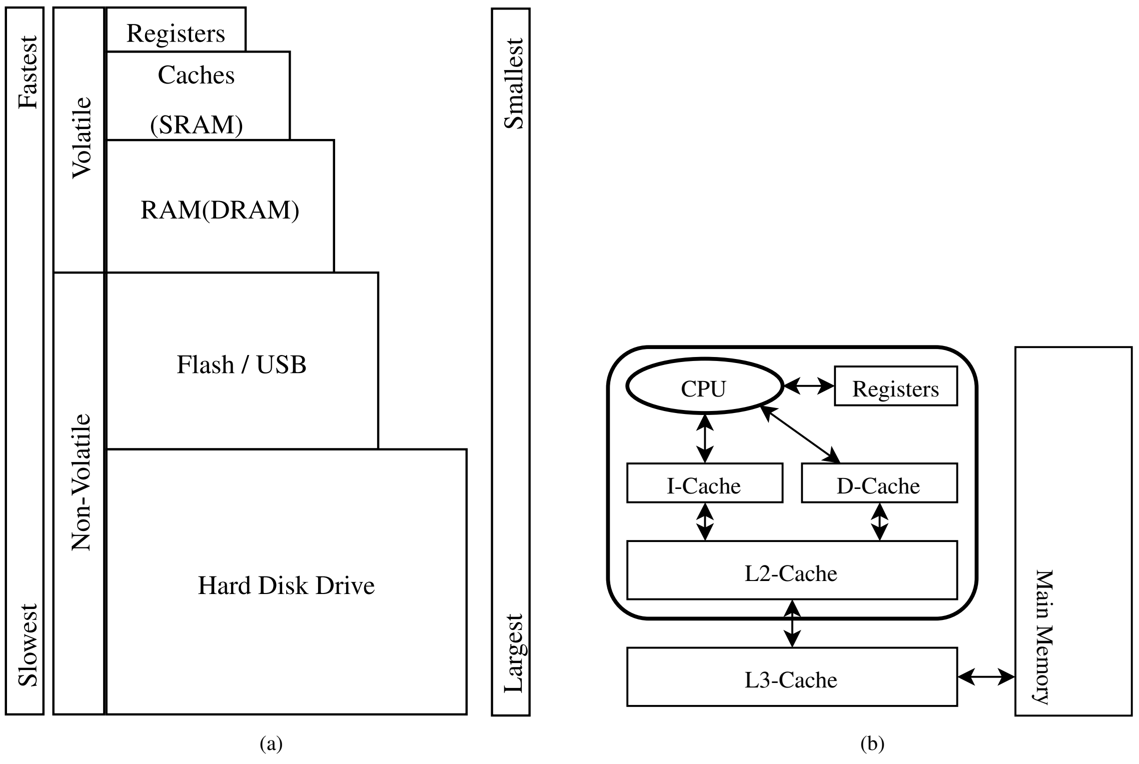


Figure 1. Memory hierarchy: (a) Memory types and their properties, (b) Typical memory hierarchy with two-level cache system inside the processor connected to a third-level cache outside the processor.

## 1.1 Motivation and problem formulation

In the past years, a universal memory which has low-power leakage, high-density storage, and fast access time has been becoming possible largely thanks to the increasing popularity of Non-Volatile Memory (NVM) technologies. Magnetic Random Access Memory (MRAM) and Phase-Change Random Access Memory (PCRAM) are examples of NVM technologies that offer the aforementioned characteristics. They could potentially be used to create high-density, low-power, relatively fast, and non-volatile memories across the different layers of the memory hierarchy. NVM technologies could potentially offer a candidate that covers the wide spectrum of memory devices from highly optimized microprocessors caches with low latency and power leakage to highly optimized permanent storage with high density and little to no data loss over time.

In this thesis, an L1 Non-Volatile Scratch Pad (NV-SP) is evaluated for performance, energy, and power leakage. Performance is evaluated on a Single-Core x86 processor where the regular L1 cache is augmented with a configurable Scratch Pad made of Phase Change Memories (PCM) or Magnetic RAM (MRAM). Energy is evaluated using the combination of energy estimations obtained from specialized simulator and performance measures. Applications are profiled where the top *k* instructions which suffer the largest number of cache misses are moved into the Scratch Pad. Using this L1 Scratch Pad reduces the energy consumption and leakage and increases the performance of the system.

## 1.2 Contributions of the thesis

The focus of this thesis will be directed towards the L1 caches of the cache system: more specifically, on-chip Instruction cache. I-Caches have a higher frequency of Read accesses compared to their Write accesses frequency. This is largely due to the fact that instructions are reused during the execution of an application, which means that there are fewer instructions being written to the I-cache from the higher memory hierarchy. In this thesis, a framework implementation is developed for evaluating the potential use of Non-Volatile memory technologies in improving the performance, energy, and security of low-level memory hierarchy. More specifically, an NV-SP is inserted into the L1 of the memory hierarchy to store instructions that cause a high number of misses. To achieve the required framework, the following tasks must be completed.

- TASK 1: Set up the necessary environment and tools for the experiment as follows:
  - Design a working x86 processor system that includes an L1 I/D-caches, L2 cache, and the main memory.
  - Configure the tools needed to complete the necessary evaluations.
  - Create applications' binaries with no compilation optimization to obtain the exact instructions executed during the simulation.
- TASK 2: Create a method to determine the performance of the I-cache and extract the needed data as follows:
  - Obtain information regarding the number of instructions and how many I-cache misses each instruction causes, the number of I-cache hits, the number of overall accesses to the I-cache, and the overall execution time.
  - Determining the combined latency of all the I-cache misses and the impact these misses had on the overall execution of the application.
- TASK 3: Create a method to determine the performance of the NV-SP as follows:

  - Obtain information on the time and energy performance of the different non-volatile memory technologies under evaluation using a specialized simulator.
  - Determining the performance, access energy, and power leakage impact of using the NV-SP to store instructions that cause a high number of misses by comparing those numbers to the performance of the I-Cache.
- TASK 4: Conduct an analysis of the results obtained from the experiments.
- TASK 5: Use the analysis to refine the current configuration of the cache and scratch pad to find the optimal cache and scratch pad configuration for maximum performance, energy, and security gains and repeat the above tasks.

This framework is designed to test the effectiveness, advantages, and disadvantages of incorporating an NV-SP memory on the x86 processor system under evaluation. This framework will also give insights on other potential Non-Volatile technology applications of in-memory computing. These insights will inspire future work in the subject.

## 1.3 Thesis organization

This section shows the content and organization of this thesis. In Chapter 2, descriptions of previous works related to the topic and methods of this thesis are presented. In Chapter 3, the technologies and tools used in this thesis are discussed. A closer look is taken at the Non-volatile memory technologies evaluated as well as the simulators used to evaluate the performance and energy of these technologies. In Chapter 4, NVM-based logic and Computation in Memory are discussed and implementation of NVM-based AES S-box is presented. In Chapter 5, the methodology and implementation of the framework of this thesis' experiment is discussed. In Chapter 6, results from this thesis' experiment are presented and analyzed. In Chapter 7, conclusions are drawn based on the results and data obtained in the course of this thesis work.

# 2 Related work

In recent years, the research community has been paying more attention to NVMs and their potential to offer a universal memory device that can be used across the different layers of the memory hierarchy. From the very first theorization of the memristors [7] in 1971 to the first working prototype [8] 37 years later, NVMs has been seen as the potential solution for many issues arising in nanoscale electronics.

Khvalkovskiy et al. [9] showed that STT-MRAM, an implementation of MRAM, can be used in memory arrays and proposed different designs by reordering the cell components, but no actual implementation of such arrays into a full memory device was proposed. Similarly, Hamann et al. [27] showed that phase-changing material can be used to create a low-energy PCRAM cell with high performance. However, Hamann et al. only showed proof of principle, with no actual propositions for implementation in memory devices.

Zhu and Park [10] showed that Magnetic Tunnel Junction (MTJ) properties can be utilized to realize a fully working MRAM cell that stores information at low energy cost and offered simple implementations for different MTJ application in memory devices. As for PCRAM, Burr et al. [29] proposed PCRAM arrays that can be used to create a memory device. However, both [10] and [29] did not evaluate the memory devices proposed as part of a larger system.

Senni et al. [11] tested potential applications for the different MRAM implementations in the memory system. Namely, STT-MRAM and TAS-MRAM were evaluated for performance and energy consumption for L2 cache as well as evaluating the STT-RAM for the L1 caches in different scenarios – Where I-cache and D-cache are both STT-RAM, only D-cache is STT-RAM, and only I-cache is STT-RAM. While [11] evaluated both the I-cache and D-cache based on STT-RAM in different scenarios, it did not evaluate the STT-MRAM in a context where other NVM technologies are evaluated for comparison purposes. Furthermore, the evaluation in [11] was an implementation of a traditional von-Neumann processor system that did not fully take advantage of the potential low Read and energy cost of the STT-MRAM. Using NVMs to replace SRAM in von-Neumann system surely gave insight into the overall performance of MRAM when used in the memory system. However, it did not show how MRAM can assist, rather than replace, contemporary memory technologies to improve the performance and energy of the memory system.

The evaluation framework used in this thesis was largely inspired by the evaluation framework used by Senni et al. [11]. However, the introduction of the NV-SP to the memory system focuses on exploiting any performance gains offered by STT-MRAM. Thus, a more focused framework was developed to include the evaluation of the NV-SP. Furthermore, this framework, unlike the framework in [11], was also designed to evaluate PCRAM alongside MRAM. Using a stand-alone NVM-based device can shed some light on the effectiveness of using NVMs to create processor systems that fall outside the traditional von-Neumann architecture.

While NVMs are being evaluated for their capabilities as memory devices, Kolodny et al. [16] showed that memristor-based logical operations are possible. Kolodny et al. presented functional *AND* and *OR* gates with prepositions for *NAND* and *NOR* gates. However, the *NAND* and *NOR* were designed with CMOS inverters, which in turns shows that such gates cannot be fully designed only using memristors. This was overcome by Kvatinsky et al. [17] as it implemented working *NAND* and *NOR* that are fully made from memristors. Both [16] and [17], however, did not implement such memristor-based logic operation in circuits that can perform complex logical operations.

Using previously proposed *AND* and *OR* gates, Wang et al. [15] designed an *XOR* gate and showed how it can be used to implement a fully functional memristor-based Full Adder circuit. Following on the footsteps of Wang et al, this thesis proposes a memristor-based AES S-Box implementation using the aforementioned *XOR* gate.

# 3 Background and Preliminaries

In this chapter, the technologies and tools utilized in this thesis are discussed. The first section discusses the different NVM technologies evaluated. The second and third sections will have short introductions to gem5, the simulator used in the framework implementation, and NVSIM, the tool used to acquire estimations on performance, access energy, and power leakage of the different NVM technologies.

## 3.1 Non-Volatile Technologies

Today's computer systems suffer from an increase in power consumption due to the increasingly problematic current leakage in modern technology nodes. Much of this power consumption happens in the memory system, especially in the volatile memory areas of System-on-chip (SoC) components [11]. Since the high-density integration of CMOS-based memory is increasingly taking more space on-chip, it significantly increases the power consumption per area due to power leakage. This forces the use of controllers that switch off parts of the chip while other parts are switched on, potentially affecting the overall performance of the system [5], [22]. The issue of power leakage enforces limitations of clock frequency due to heat dissipation, thus hindering performance improvements [5], [23]. Non-volatile memories, such as MRAM and PCRAM, offer a solution to this problem. Using NVM, power consumption can be decreased significantly since NVM offers scalable, high-density memory using less space as well as its non-volatile nature allows the use of the NVM with very low power leakage.

Furthermore, NVM offers an array of solutions to other complicated issues. Most important of these solutions is finding an alternative solution to the volatile nature of the volatile memories i.e. registers and various levels of cache space both on-chip and off-chip memory systems in the memory hierarchy of computer architectures. It offers a solution for long wake-up times and system status restore. Since CMOS-based memory is volatile, all information stored in it is lost when power is turned off. By introducing NVM, the current state of execution (processor state and memory state) can be stored during power shut down, allowing a faster reboot of the system during power failure (this can be done in several ways depending on the architecture) and faster wake up time in restoring execution status. This can be done by storing the current state of execution in non-volatile registers (NVReg)[11] and non-volatile partitions (NVMs) of the various cache systems.

While there is a lot to explore in terms of the non-volatile nature of NVM technologies, in this thesis, the focus will be directed to the low leakage and performance aspects of NVMs. More specifically the gains and losses of using two NVM technologies, Spin Torque Transfer Random Access Memory (STT-MRAM) and Phase-Change Random Access Memory (PCRAM) to introduce a Scratch Pad memory into the first memory level of the memory hierarchy.

In this section, characteristics of MRAM and PCRAM are further explained and explored.

### 3.1.1 Magnetic Random Access Memory (MRAM)

Magnetic Random Access Memory (MRAM) is a candidate to replace current memory. MRAM is a non-volatile memory that offers high scalability, high density, low latency, and low leakage and can provide a solution to the power efficiency issue since it can be completely shut down without loss of data stored in the MRAM [11]. These attractive features put MRAM on the path of becoming a universal memory.

MRAM is largely based on Magnetic Tunnel Junction (MTJ). In MTJ, two conducting electrodes are separated by a tunnel barrier causing electrical conduction due to electrons tunneling through the barrier. The tunnel barrier, made of a dielectric layer, ranges from a couple of hundred pico-meters to a few nanometres in thickness. This tunneling phenomenon is caused by the wave nature of electrons while the conductance is caused by the electron wave function within the barrier [9].

To realize this behavior, MRAM cells are made of two ferromagnetic layers separated by a barrier. The top electrode layer called the Free Layer (FL), also called the storage layer, has a changing magnetic orientation that changes based on the voltage pulse delivered by electric contact. The bottom electrode layer called the Reference Layer (RF), also called the Fixed Layer, has a fixed magnetic orientation used as a reference for reading from and writing to the MTJ cell, and the barrier is made of an insulator. The MTJ layers are illustrated in Figure 2 (a). Figure 2 (b) and (c) show illustration and schematic of an MRAM respectively.

In the MRAM cell, the information is stored as the resistance of the MTJ caused by a phenomenon called the Tunnelling MagnetoResistance (TMR). The resistance is dependent on the orientation of the ferromagnetic layers, thickness, and the type of material of the

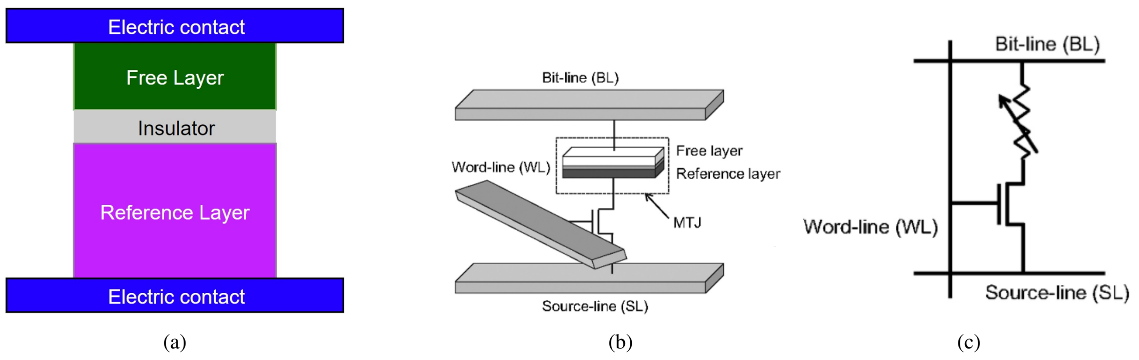


Figure 2. MTJ layers in an MRAM cell: (a) Schematic showing the two electrode layers in a Magnetic Tunnel Junction separated by an insulator, (b) An MRAM cell [24], (c) A schematic view of an MRAM cell [24].

Free Layer, the Reference Layer, and the barrier layer. However, the major factor is the magnetic orientation, and since the Free Layer is the only layer with changing magnetic orientation, there are two states that the cell can have as shown in Figure 3 (a) and Figure 3 (b):

- The first state happens when the magnetic orientation of the Free Layer is parallel to the magnetic orientation of the Reference Layer. This means that the Resistance of the MTJ cell is low donating a Logic Zero "0" as shown in Figure 3 (a).
- The second state happens when the magnetic orientation of the Free Layer is anti-parallel to the magnetic orientation of the Reference Layer. This means that the Resistance of the MTJ cell is High donating a Logic One "1" as shown in Figure 3 (b).

The process of reorienting the Free Layer is implemented in three ways:

- Spin Transfer Torque (STT-MRAM): This method uses the spin-transfer torque effect to reorient the Free Layer. When a highly polarized current is flowing through the MTJ cell, a torque is created. This happens when the electrons spin on the magnetization of the Free Layer causing it to slowly reorient[10]. STT-MRAM is unique from the other MRAM technologies due to its many potential applications [25].
- Toggle: this method of reorientation uses a sequence of voltage pulses designed to rotate a specially made Free Layer called the Synthetic AntiFerromagnet (SAF)

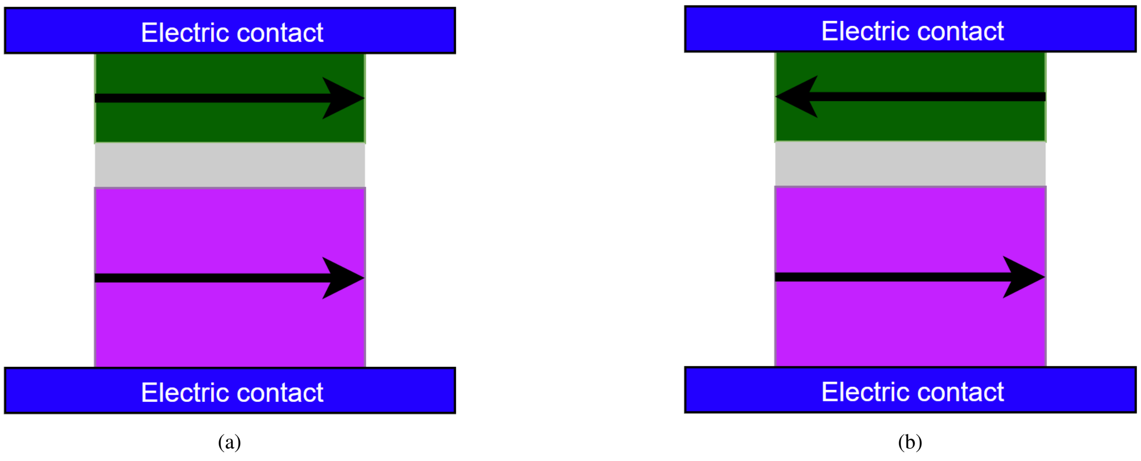


Figure 3. Demonstrations of MTJ: (a) The magnetic orientation of the Free Layer is parallel to the magnetic orientation of the Reference Layer. In this case, cell resistance is low, (b) The magnetic orientation of the Free Layer is anti-parallel to the magnetic orientation of the Reference Layer. In this case, cell resistance is High.

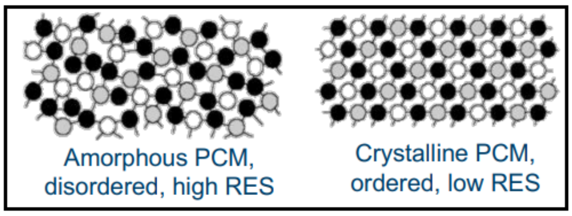
that is programmed by applying a series of voltage pulses causing the Free Layer to toggle from one state to another [13].

- Thermally Assisted Switching (TAS-MRAM): TAS is similar to Toggle, however, TAS utilizes an extra layer that prevents the Free Layer from switching when it is under a threshold temperature. When a current is applied to a Select Transistor, it heats up the MTJ cell to a higher temperature than the threshold temperature causing the magnetic orientation of the Free Layer to change [14].

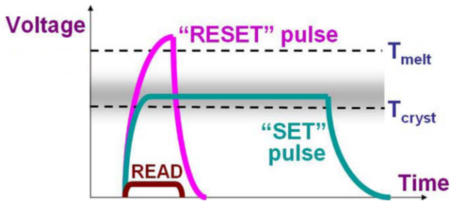
For this thesis, the STT-MRAM implementation is used. Among these reorientation implementations, STT-MRAM showed the most promise. It is faster than Toggle and TAS and is more academically researched and much more developed and understood compared to the other two implementations [11]. STT-MRAM will be used in the evaluation of the NV-SP. The result of this evaluation will be compared to the results of PCRAM and traditional I-cache.

### 3.1.2 Phase-Changing Random Access Memory (PCRAM)

Phase-Changing Memory (PCM) is among the rapidly advancing NVM technologies out there [26]. It is composed of a nano-metric volume of phase-changing materials that exhibit a significant change in resistance when it changes from one state to another. This material is then put between two electrodes to make PCM cell [27]. When different voltage pulses are applied to the material, the material changes from an amorphous state to a crystalline state. Figure 4 (a) shows the amorphous and crystalline states of a phase-changing material.


Amorphous PCM, disordered, high RES
Crystalline PCM, ordered, low RES

(a)



(b)

Figure 4. States of a PCM cell: (a) The amorphous and crystalline states of a phase-changing material [28], (b) Voltage/temperature-Time relationship during RESET, SET, and READ showing the $T_{melt}$ and $T_{cryst}$ [29].

By applying a suitable electrical pulse, the material heats up to a certain temperature ($T_{melt}$) causing the phase-changing material to be altered. Such pulses that cause a volume of amorphous material to quench are called *RESET* pulses. This RESET operation leads to an increase in the resistance of the material. On the other hand, longer pulses that cause the material to heat into a slightly lower temperature ($T_{cryst}$) increasing the crystallization of the material are called *SET* pulses. These SET operations lead to a decrease in the resistance of the material. These two operations (RESET and SET) constitute the *WRITE* operation on the PCM cell – similar to electrically charging the material. To constitute a *READ* operation on the PCM cell a low electrical field is applied to the phase-changing material allowing the electrical charge to flow freely through the material into the bottom electrode – similar to electrically discharging the material [30]. Figure 4 (b) demonstrates the voltage/temperature-time relationship of a PCM cell.

Depending on the amplitude of the electrical pulse, the resistance of the material changes. Measurement of this resistance varies depending on the electrical pulse, this means that we can translate the resistance to store different sets of data inside the material. Different material can be configured to suitable pulses achieving a continuum of resistivity creating a high-capacity low-power analog device for storing information [28]. Figure 5 shows a possible continuum of resistivity with a PCRAM cell. Figure 6 (a) and (b) show illustration and schematic of an PCRAM respectively.

*GeTe*-$Sb_2Te_3$ (GST) and $GeTe_2$-$Sb_2Te_5$ (GST-225) [29] are among the most commonly used phase-changing material with PCMs. Perfect RAM or Phase-changing RAM (PCRAM) is the most notable application of these types of memories. PCRAM offers

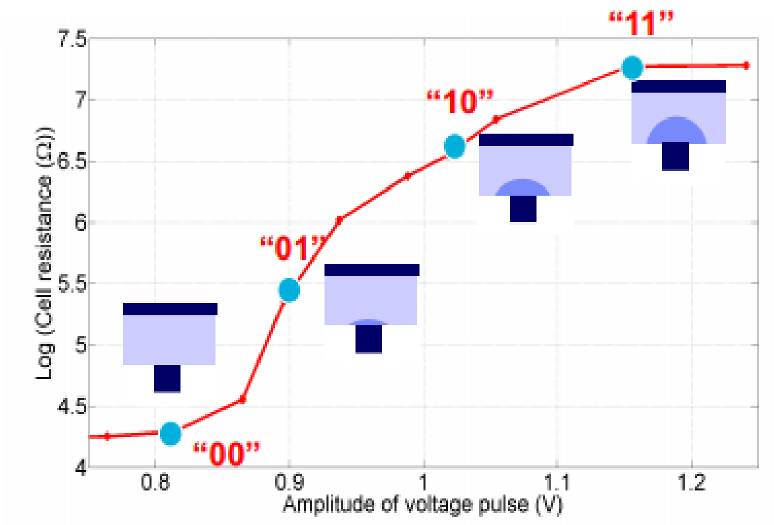


Figure 5. The continuum of resistivity that can be achieved when using suitable pulses with a phase-changing material [28].

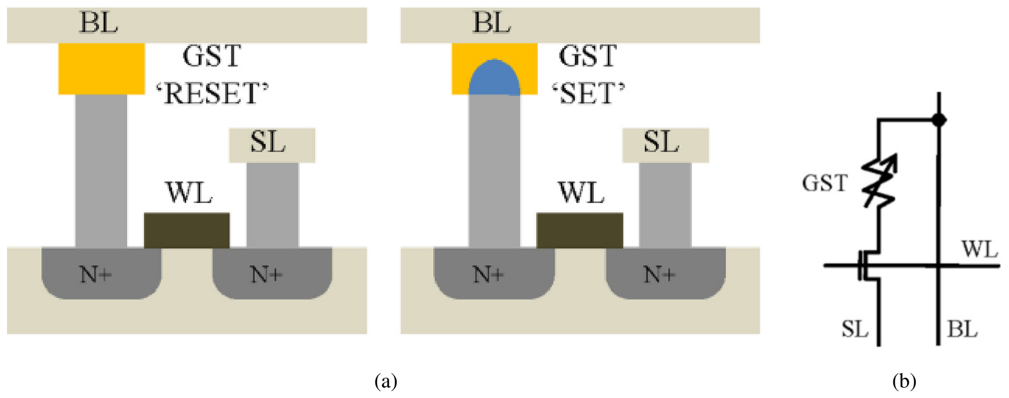


Figure 6. Demonstrations of PCRAM: (a) A PCRAM cell, (b) A schematic view of a PCRAM cell [24].

two notable advantages. The first is a multi-level storage property due to the ability to configure it with a continuum of electrical pulses. The second is the dynamic behavior of the PCRAM units that applies a complex system of feedback interconnection of electrical, thermal, and structural dynamics as shown in Figure 7.

In this thesis, PCRAM is used to evaluate NV-SP memory. The result of this evaluation will be compared to the results of STT-MRAM and traditional I-cache.

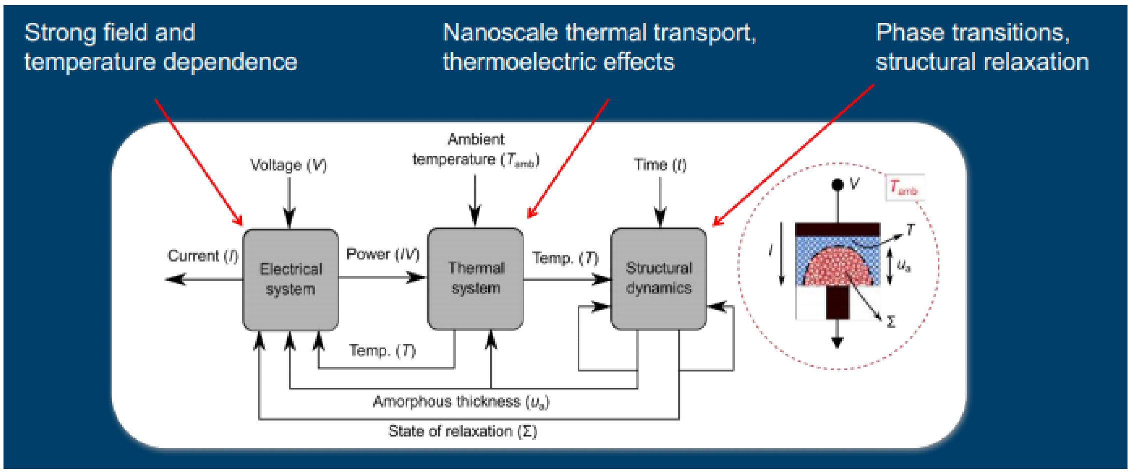

Strong field and temperature dependence
Nanoscale thermal transport, thermoelectric effects
Phase transitions, structural relaxation
Voltage (V)
Ambient temperature ($T_{amb}$)
Time (t)
Current (I)
Electrical system
Power (IV)
Thermal system
Temp. (T)
Structural dynamics
Temp. (T)
Amorphous thickness ($u_a$)
State of relaxation (Σ)
V
$T_{amb}$
T
I
$u_a$
Σ

Figure 7. The dynamic behaviour of a PCRAM unit complex system of feedback [28].

## 3.2 gem5

The gem5 simulator is a product of collaboration between academic and industrial institutions [31]. gem5 is widely used by the research community to simulate processor architecture. It offers a highly configurable simulation framework with diverse CPU models and multiple Instruction Set Architectures (ISAs). Furthermore, gem5 features a detailed and flexible memory system that supports many interconnect models and cache coherence protocols. ISAs such as x86 and ARM are two of the many ISAs supported by the simulator.

There are two execution modes of gem5:

- System-call Emulation (SE): In this mode, gem5 emulates the most common system calls [31]. Whenever an application is executed, the application issues a system call, gem5 traps these calls and emulates them.
- Full-System (FS): In this mode, gem5 simulates a bare-metal environment suitable for running an OS. As the name suggests, a full system is emulated including I/O devices, exceptions, interrupts, etc.

While Full-System mode has higher accuracy and can handle a wider variate of workloads compared to System-call Emulation mode, in this thesis, System-call Emulation will be utilized due to the simplicity of the processor architecture studied in this thesis. gem5 is used because it allows defining the overall processor system architecture. This includes the memory hierarchy specifications such as the different levels of the cache system, cache sizes, latencies of the caches and the main memory etc. Memory traces can show the propagation of data within the memory. They also show each memory's read and write accesses and the cache misses and hits and their respective rates. By modifying these memory traces, determining which instruction was executed at what time becomes possible. Trace files can show the overall execution time needed for an application to be executed as well as statistics regarding the execution flow.

## 3.3 NVSim

NVSim is a circuit-level model for NVM performance, energy, and area estimations, which supports various NVM technologies, including STT-RAM, PCRAM, ReRAM, and legacy NAND Flash. NVSim also supports volatile memory technologies such as SRAM and DRAM [24]. Depending on a given configuration, NVSim estimates the access time, access energy and area of the NVM technology chip. These estimates are then used to explore the optimal NVM chip organization. This helps in finding the optimal NVM chip design space for achieving the best performance, area, or energy.

NVSim is used in this thesis to find the optimal performance and energy estimations of STT-MRAM and PCRAM memory cells as well as the performance and energy estimations of a 4 kB SRAM cache. These estimations are later used to evaluate the performance of a Scratch Pad memory made with each of the NVM technologies.

# 4 Methodology

In this chapter, the methodology used in this thesis and the framework implementation described in section 1.2 are discussed in detail. Section 1 discusses the architecture used in the simulation. Section 2 shows the different workloads simulated in the architecture. Sections 3 and 4 describe the configuration files of gem5 and NVSim. Section 5 will present the framework implementation and the bash scripts used to obtain the simulation results before the analysis of the data.

## 4.1 Architecture

The x86 architecture is an instruction set architecture (ISA) based on the Complex Instruction Set Computer (CISC) architecture. In CISC processors, instructions can execute several low-level operations, such as a load, arithmetic, and store operations in a single instruction [38]. x86 defines how a processor handles and executes instructions passed to the processor by the Operating System (OS). Developed by Intel Corporation, x86 architecture was used in multiple processor chipsets, most notable are Intel's 8008, 80188, Pentium, and Xeon chipsets as well as AMD's K7, K8, and K10 chipsets [39]. The popularity of the x86 architecture and its wide use makes it an obvious choice for processor-related simulations for research and development purposes.

In this thesis, an x86 ISA is simulated with an I-cache, a D-cache, an L2 cache, and the Main memory. Table 1 shows the system components' configuration used in this thesis. The objective selecting of this architecture is to focus on the direct interaction between the processor and the L1 I-cache. This interaction can be exploited or used to develop high-efficiency systems to reduce the bottleneck issue that may arise from the ever-increasing speed gap between CPUs and caches. Thus, only a Single-Core is used since the applications will not be executed on multiple cores.

Using a Single-Core will give insight into the direct impact of adding a scratch pad to the L1 memory hierarchy. To that end, the L1 caches were separated as opposed to a unified L1 cache. The sizes of the L1 I/D-caches, at 4-kilobytes each, was used to accommodate the small size of the applications simulated. Since only a Single-Core is used, applications used in the simulations (discussed in the following section) had to be small in size for the purpose of convenience. Since the size of L2 cache was irrelevant to

Table 1. System components' Configuration.

| System Component | Configuration |
|---|---|
| Processor | Single-Core, 1 GHz, 32-bit CISC x86 |
| L1 I/D-cache | Private, 4kB, 2-way associative, 64B cache line |
| L2 cache | Shared, 64kB, 4-way associative, 64B cache line |
| Main memory | DRAM, DDR3, 100-cycle latency |

the overall outcome of the thesis, the size of 64-kilobytes is set as to allow the entirety of the applications and their inputs/necessary files for execution to be fully loaded to the L2 avoiding any overall performance degradation. The default Main Memory and memory controller in gem5 (the simulator) is used. Figure 8 shows an illustration of the x86 architecture used in this thesis.

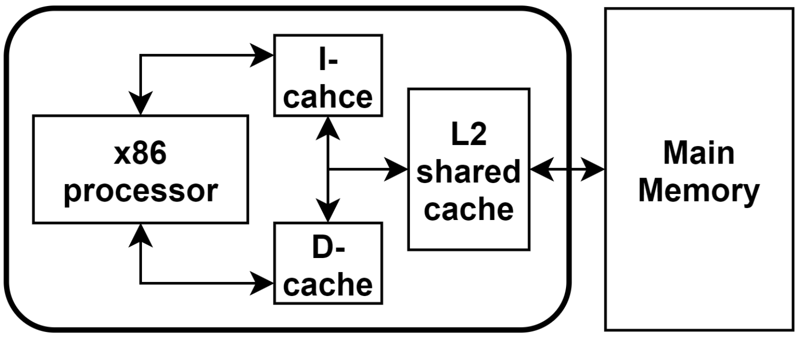


Figure 8. Illustration of the x86 architecture used in this thesis.

As for the NV-SP, three sizes of 1, 2, and 4-kiloBytes are used to evaluate the impact of different sizes of NV-SP on the performance of the architecture. Since this architecture executes 32-bit instructions, each instruction will be 4 Bytes of size. Each of the NV-SP sizes of 1 kB, 2 kB, and 4 kB will store 250, 500, and 1000 instructions respectively. These sizes were used to accommodate the small size of the applications simulated as well as to give insight on the impact of different sizes of the NV-SP can have on the final results. Figure 9 shows the incorporation of NV-SP to the x86 architecture used in this thesis.

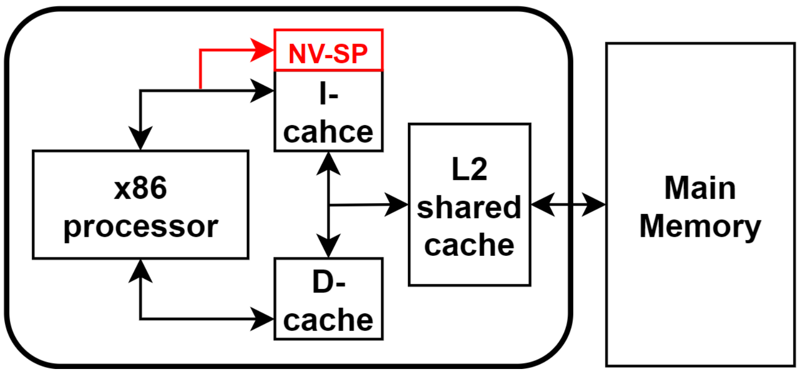


Figure 9. Illustration of the x86 architecture with incorporation of NV-SP.

## 4.2 Architecture Workloads

In this section, the applications – referred to as workloads – used in this thesis' experiment are introduced and discussed.

### 4.2.1 Applications

The applications chosen for this thesis are based on their simple implementation and small size. This allowed for easier editing of the code to accommodate the constraints set forth by the architecture's 32-bit compatibility and the SE mode of gem5 used in the simulation. For this thesis, two types of applications were simulated during the experiment.

#### Cryptographic Algorithms

The first type of applications used in this thesis is a collection of cryptographic algorithms based on an open source implementation of cryptography algorithms [40]. The original implementations in [40] were simple and easily editable to accommodate the system constraints. Furthermore, all the implementations had test benches designed by the author. Those test benches are the ones used to execute the algorithms during the simulations. The cryptographic algorithms are the following:

- **Advanced Encryption Standard (AES)**
  The Advanced Encryption Standard (AES), originally known as Rijndael cipher, is a symmetric-key algorithm. It is the standard specification for electronic data encryption adopted by the National Institute of Standards and Technology (NIST)

in 2001 [34]. AES features a block size of 128 bits and key lengths of 128, 192, and 256 bits.

- **Data Encryption Standard (DES)**
  Was the predecessor of AES as the standard specification for electronic data encryption. Was developed in the early 1970s [41], it became the standard 1976. DES has been shown to be vulnerable to cryptographic attacks, however, in 1999, another version of the cipher, Triple DES (or FIPS-46-3), was reaffirmed as a safe cipher. DES was withdrawn by the NIST in 2005. In this thesis, we used the Triple DES implementation of the cipher. DES has a block size of 64 bits and 64-bit key length.

- **Rivest Cipher 4 (ARCFOUR)**
  Was designed by cryptographer Ron Rivest in 1987 while working at RSA Security [42]. The algorithm became public in 1994 when it was leaked on online mailing lists. RSA Security never officially released the algorithm, but in 2014, ARCFOUR creator Ron Rivest has confirmed the code of the algorithm [42]. ARCFOUR was tested against cryptographic attacks many times, but there is no evidence showing ARCFOUR has been broken. ARCFOUR is a streaming cipher with a key length of 40 to 2048 bits.

- **Twofish cipher**
  Twofish is a symmetric key block cipher that was among the five finalists in the Advanced Encryption Standard contest held by NIST between 1997 and 2001 [43]. However, Rijndael cipher eventually won the contest. With a block size of 128 bits and key sizes up to 256 bits, Twofish cipher is never demonstrated to be broken or vulnerable to cryptographic attacks. Twofish is related to the earlier block cipher Blowfish.

- **Blowfish cipher**
  Blowfish is a symmetric-key block cipher, designed in 1993 by Bruce Schneier as an alternative to the then increasingly vulnerable DES [44]. With a 64-bit block size and a variable key length from 32 bits up to 448 bits, was shown to be vulnerable to a type of cryptographic attacks called birthday attacks. Since then, the creator of Blowfish has recommended using its more secure successor, Twofish.

- **Rotate by 13 places Cipher (ROT13)**
  ROT13 is a very simple substitution cipher that replaces a letter with the 13th letter after it in the alphabet. ROT13 is similar to the Roman Ceasar Cipher. It provides no

cryptographic security whatsoever. The inclusion of ROT13 in this thesis is solely to understand the performance of the SP-NV with simple ciphers.

Table 2 shows the comparison in key length, key lengths used in the simulation, block sizes, and vulnerability of the cryptographic algorithms used in this thesis. These cryptographic algorithms were not compiled and did not include a build script. A compilation command was to be devised and used in order to create executable binaries from these algorithms. The command is described in section 4.2.2 is used to create such binaries. The use of cryptographic algorithms is aimed at the understanding of the performance of Non-Volatile Memory technologies when used for cryptographic purposes. The results obtained will be a reflection of how well does the NVMs perform when applied in fields that require encryption and decryption of data that utilizes commonly used cryptographic algorithms. The results will also shed light on the potential use of NVMs in designing non-volatile devices aimed to implement these algorithms on the hardware level. The design of such devices is one future work that will be further explored.

Table 2. Cryptographic Algorithms' key lengths, block sizes, and vulnerability comparison.

| **Cryptographic Algorithms** | **Key Length (used in simulation)** | **Block Size** | **Vulnerability** |
|---|---|---|---|
| AES | 128, 192, and 256 bits (256) | 128 bits | Very Secure |
| DES | 64 bits (53) | 64 bits | Insecure |
| ARCFOUR | 40 to 2048 bits (256) | None | Secure |
| Twofish | 256 bits (256) | 128 bits | Secure |
| Blowfish | 32 to 448 bits (256) | 64 bits | Insecure |
| ROT13 | None | None | Very insecure |

## Non-Cryptographic Algorithms

The second type of applications used in this thesis is a collection of non-cryptographic algorithms. These algorithms were chosen based on their computational nature. The JPEG workload will show the performance of the NV-SP when tasked with a stream-like behavior of instructions. The encoding and decoding of an image require the same set of instructions to run many times with different data. The bzip2 algorithm is a classic compression algorithm that will show the performance of the NV-SP when tasked with data compression workloads. The specrand workload will show the performance of the NV-SP when tasked with random number generators that require using random sets of instructions to generate random numbers.

- **JPEG image compression and decompression**
  JPEG is a standardized compression method for continuous-tone still images, both full-color and gray-scale images. JPEG is designed to compress real-world-like images using Discrete Cosine Transform (DCT) compressions algorithm [45]. JPEG is lossy, meaning that the output image is not exactly identical to the input image. In this thesis, we use the MediaBench Consortium implementation [46]. The MediaBench implementation of JPEG is a package of two demo programs:
  - **djpeg**
    Djpeg is for decoding jpg files to a variety of graphic file formats [47].
  - **cjpeg**
    Cjpeg encodes a file to a jpg file from different graphic formats [47].
- **bzip2**
  bzip2 is an open-source single-file compression program first published in 1996 [48]. bzip2 is based on the burrows-Wheeler block-sorting text compression algorithm, and Huffman coding [49]. bzip2 is often used in benchmark suites such as SPEC CPU2006 Benchmark suite [50]. The version of bzip2 used in the experiment is bzip2-1.0.6 [51]. Both compression, (bzip2-c) and decompression (bzip2-d) modes are used.
- **specrand**
  specrand is a seeded generator of a sequence of pseudorandom numbers and part of the SPEC CPU2006 Benchmark [52]. It is based on the "Random Number Generators: Good Ones Are Hard To Find" [53] algorithm.

### 4.2.2 Compilation

Since the x86 architecture is a 32-bit architecture simulated in the SE mode of gem5, the compilation process of the applications used as workloads must accommodate these constraints. First, the applications are slightly altered before they are compiled. These changes assure that the workloads are not too big for the simulations and remove any code that could cause system calls that are unsupported by the SE mode. Since this is not possible for already-compiled workloads or for workloads that are too large to edited manually, smaller applications are targeted, edited, and compiled. Second, the workloads must be compiled as 32-bit statically linked binaries. To achieve this, all the applications source code is written in C Language. The GNU GCC 5.5.0 [54] compiler is used to compile the applications into binaries. The following command is issued to compile the applications:

```
gcc app.c app-test.c app.h -m32 -static -O0 -o app
```

- The compiler and the inputs:
  - The `gcc` command invokes the GNU GCC compiler. The GCC compiler is one of, if not the most, important piece of open source software in the world [55]. Developed by the Free Software Foundation, GCC offers support for a wide array of programming languages. It offers online command options, supports mixing multiple languages to build applications, and features a debugger, an automatic configuration utility, and many other useful utility programs [55].
  - The `app.c` input specifies the main source code for the application being compiled. All the applications edited and compiled for this experiment are written in the C language.
  - The `app-test.c` input specifies any test bench source codes for testing the `app.c` code. This option is only used when the applications themselves do not accept any input parameters or input files after the compilation process and can only be tested by executing the code on a test bench.
  - The `app.h` input specifies any header files associated with the `app.c` code or dependencies needed for the compilation of the `app.c` code.
- Compiler option flags:
  - The `-m32` option compiles the code into binaries that can run on any i386 system [54]. `-m32` forces data types, such as `int`, `long`, and `pointers`, to be set to 32 bits allowing these binaries to be executed on a 32 bit processor.
  - The `-static` option forces the GCC compiler to statically link the binaries on systems that support dynamic linking [54]. It prevents the `app.c` code

from linking with the shared libraries. The system where this command is executed supports dynamic linking, thus the importance of using this option.

– The `-O0` option reduces the compilation time and make debugging produce the expected results [54]. Although this is a default option in GCC, the use of `-m32` flag invokes some optimization during compilation. The `-O0` option makes sure that any optimization invoked by the `-m32` command are ignored. By using this command and preventing optimizations, if any, the exact instructions are compiled and executed by the binaries. This way, instructions can be profiled and information about them can be extracted during the experiments.

– The `-o app` option specifies the output binary files form the compilation process.

This compilation command is used primarily with the cryptographic algorithms since the original code in [40] did not include any build scripts. The non-cryptographic algorithms included build scripts with the original source code. These building scripts were subsequently edited to include the options mentioned above.

## 4.3 gem5 Configuration

In gem5, the configuration files were created manually without using the included configurations that came with the simulator. Since the SE mode was used in the simulations, a new configuration file was written to describe the architecture. The caches' configuration files were also written manually. After the architecture and the caches were configured, two new trace flags were created, in addition to trace flags built into the simulators, to trace the behavior of the caches as well as to record the exact instructions being processed by the CPU.

### 4.3.1 Configuration Files

Two files were created for the configuration of the architecture and the caches. The first file caches.py contains the specifications of the L1 I/D caches and the L2 shared cache. The second file `Single_Core.py` contains the CPU specifications as well as how the L1 caches are connected to the CPU, connected to other caches, and what applications are being simulated. Those two files are as follows:

**caches.py**

This file contains the configuration of the L1 and L2 caches used by the architecture. Both of the caches specifications are written manually and are imported to `Single_Core.py` file to connect the caches to the CPU. Listing 1 shows the caches' specifications. Both of the caches, `L1Cache` and `L2Cache`, extend the BaseCache object [56]. The `L1Cache` has an associativity of 2, and no specified size as that is specified when creating the `L1ICache` as an instruction cache and `L1DCache` as a data cache. `L2Cache` has an associativity of 4 and a fixed size of 64kB.

**Single_Core.py**

This file contains the specifications of the CPU, buses, how the different cache levels are connected to the CPU and the buses, and the applications being executed by the simulator. Listing 2 shows the `Single_Core.py` specification of the architecture.

For the CPU, `TimingSimpleCPU()` CPU model was used. This model was used because it executes a single instruction in one clock cycle [57] one cycle is one nanosecond. The CPU clock was defined using `system.clk_domain.clock` parameter at `1GHz`. The `system.mem_mode` parameter is set to `'timing'` which will allow to accurate time measurement of the cache system during the simulation. This cache `timing` mode will stop the CPU from requesting any new instructions for execution until it receives a response for the latest miss occurring in the cache system. The CPU will be idle until the cache system has mitigated the miss. This feature will be very important for analysing the outputs of the debugging flags described in 4.3.2 .

The `from caches import *` is used to import the cache configurations. Two `L1ICache` caches are named `system.cpu.icache` and `system.cpu.dcache`. Those two caches connected to the CPU: one as an instruction cache using the `system.cpu.icache.connectCPU()` function and one as a data cache using the `system.cpu.dcache.connectCPU()` function. One `L2Cache` named `system.l2cache` was created as the L2 cache.

Two buses were created, one to connect the L1 caches to the L2 shared caches using the `L2XBar()` function named `system.l2bus`.

This bus is inside the CPU and is used to connect the L1 caches to the L2 cache using the `system.cpu.icache.connectBus()` and `system.cpu.dcache.connectBus()` functions to connect the I-cache and D-cache to the L2 cache respectively. The second bus is created to be a system wide bus using the `SystemXBar()` function. This is to enable the connecting of the L2 cache to the main memory outside the CPU. the The L2 cache is connected to the `system.membus` using the `system.l2cache.connectMemSideBus()` function. In order for the memory to be created, a memory controller was added using the standard gem5 implementation of a memory controller using the `system.mem_ctrl` parameter. The Main Memory was created using the standard gem5 implementation using the `DDR3_1600_8x8()` function. Figure 10 shows the illustration of the architecture configured in the `Single_Core.py` file.

A process is then created using the `Proccess()` function. When the process is created, the `proccess.cmd` parameter defines which application is to be simulated with what input parameters. The process is then assigned to the CPU using the `system.cpu.workload` parameter. The function `m5.instantiate()` signals the start of the simulation.

Listing 1. caches.py.

```
#caches.py
from m5.objects import *
class L1Cache(Cache):
assoc = 2
tag_latency = 2
data_latency = 2
response_latency = 2
...
class L2Cache(Cache):
size = '64kB'
assoc = 4
tag_latency = 20
data_latency = 20
response_latency = 20
...
class L1ICache(L1Cache):
```

```
size = '4kB'
class L1DCache(L1Cache):
size = '4kB'
...
```

Listing 2. Single_Core.py.

```
#Single_Core.py
import m5
from caches import *
...
system.clk_domain.clock = '1GHz'
...
system.mem_mode = 'timing'
system.cpu = TimingSimpleCPU()
...
system.cpu.icache = L1ICache(options)
system.cpu.dcache = L1DCache(options)
system.cpu.icache.connectCPU(system.cpu)
system.cpu.dcache.connectCPU(system.cpu)
...
system.l2cache = L2Cache(options)
system.l2bus = L2XBar()
system.cpu.icache.connectBus(system.l2bus)
system.cpu.dcache.connectBus(system.l2bus)
system.l2cache.connectCPUSideBus(system.l2bus)
system.membus = SystemXBar()
system.l2cache.connectMemSideBus(system.membus)
...
process = Process()
process.cmd = ['application','parameter1','parameter2', ...]
system.cpu.workload = process
...
m5.instantiate()
```

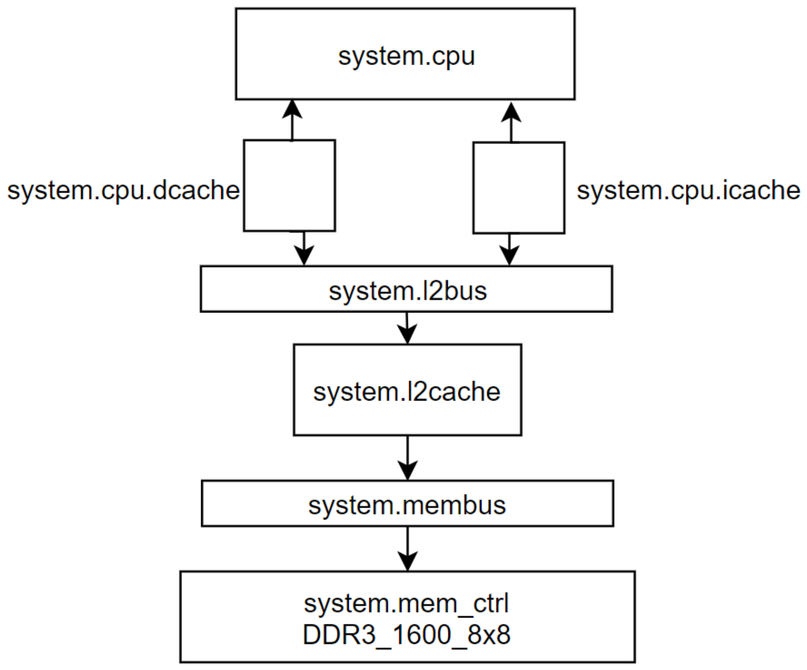


Figure 10. Illustration of the architecture design implemented in Single_Core.py configuration file.

### 4.3.2 Debugging Flags

In gem5, debugging flags are used to extract information about the simulation during simulation run time. These trace flags record the behavior of the different parts of the simulator. For the purposes of this thesis, additional debugging flags were created to record the behavior of the L1 I-cache. These two debugging flags were named `CacheTr` and `ExecTr`.

The first flag is `CacheTr`. `CacheTr` is a simple flag that prints out an identifier at the time when a miss occurs to the trace file. This is used to distinctly identify when a miss happens. The `CacheTr` is added to the `bace.cc` file of the gem5 source code. `bace.cc` is the file where the base behavior of the cache is defined in the simulator. Listing 3 shows where the `CacheTr` is called during simulation. In listing 3, `handleTimingReqMiss` function handles any misses that occur in the cache. When a miss occurs, this function defines the behavior of the cache by requesting the data/instruction from the higher levels of the caches in the memory system. When the CPU requests the next instruction, if the instruction is not in the cache, the cache will issue a Timing Request using the `handleTimingReqMiss` function. Here, adding the `CacheTr` flag to the `handleTimingReqMiss` will result in distinctly identifying when a miss occurs and adding the "THIS IS A MISS" identifier to the debugging file. This identifier includes the name of the cache where the miss occurred, as shown in listing 4, which will come handy in extracting the instructions causing the misses in the I-cache. This flag will be triggered for every cache in the cache system.

Listing 3. `CacheTr` flag Implementation.

```
BaseCache::handleTimingReqMiss(...)
{
...
missAt = (long)curTick(); //Time of miss
DPRINTF(CacheTr, "THIS IS A MISS : Miss request sent at %ld
   \n", missAt); //flag the miss into the trace file
...
```

The second flag is `ExecTr`. `ExecTr` is a more complex flag made from a combination of gem5 flags called `CompoundFlag`. The `CompoungFlag` is very useful when multiple pieces of information from different parts of the simulation are traced at the same time. These `CompoungFlags` are execution flags that do not require additional code being added to the source code of the CPU model – it is only included in the build scripts. The `ExecTr` is defined in the build script of gem5's BaseCPU model – the base model for the Timing CPU model used in the simulated architecture. The `ExecTr` flag will print the time of when an instruction is received by the CPU, the CPU ID of the CPU that received the instruction, and what instruction is being received. These two flags are effective because whoever the `CacheTr` flag is triggered, this means that the CPU is idle and the next time `ExecTr` is triggered is when the CPU receives the instruction that caused the `CacheTr` to be triggered in the first place. This will help to extract and organize the information in the debugging file after each simulation. A sample of the output of `CacheTr` and `ExecTr` flag is shown in listing 4.

Listing 4. `CacheTr` and `ExecTr` sample traces.

```
... //CacheTr sample trace
1812000: system.cpu.icache: THIS IS A MISS : Miss request sent
   at 1812000
... // ExecTr sample trace
1913000: system.cpu 0xb7fdb950 : mov ecx, 0x6ffffeff
...
```

### 4.3.3 Simulation Commands

The gem5 simulator is ran through the command line interface of a Linux machine. To run the simulation, the following command was issued:
`build/X86/gem5.opt --debug-flags=CacheTr,ExecTr --debug-file=tr/Trace_IcacheSize_DcacheSize_L2cacheSize.txt configs/Single_Core.py --l1i_size IcacheSize --l1d_size DcacheSize --l2_size L2cacheSize` This command executes as follows:

- Launching the simulator
  - The `build/X86/gem5.opt` command
    This part starts the gem5 simulator. The `gem5.opt` is the executable that initializes the simulator.
  - The `--debug-flags` and `--debug-file` options
    These are the simulator options. `--debug-flags` activates the debugging flags for the trace file during the execution of the simulator. The debugging flags set by this command are the flags previously discussed – `CacheTr` and `ExecTr`. the `--debug-file` option specifies the file where the traces from the simulation as recorded. In this case, the traces are saved as a text file that is named according to the sizes of the I-cache, D-cache, and L2 cache in a specified folder `tr`.
- Simulation configuration
  - The `configs/Single_Core.py` parameter
    This is the system configuration file. It specifies the system architecture, cache design, and the application being simulated.
  - The `--l1i_size`, `--l1d_size`, and `--l2_size` options
    These are the system configuration options. They specify the sizes of the I-cache, D-cache, and L2 cache respectively.

## 4.4 NVSim Configuration

The configuration of NVSim requires two types of files. The first type are the `.cfg` files where the specifications of the Non-Volatile device are being described. In the `.cfg` files, the capacity, type, and NVM cell model of the device are set. The second type is the

`.cell` file where the specification of a single Non-Volatile cell is modeled. It describes the design specifications of the NVM cell. For the purposes of this thesis, the generic NVM prototypes provided by NVSim are used for both the `.cfg` and `.cell` files. The only modification done on these files is changing the device capacity in the `.cfg` file to acquire NVM performance and energy estimation of the different NVM technologies' devices at different sizes.

## 4.5 Framework Implementation

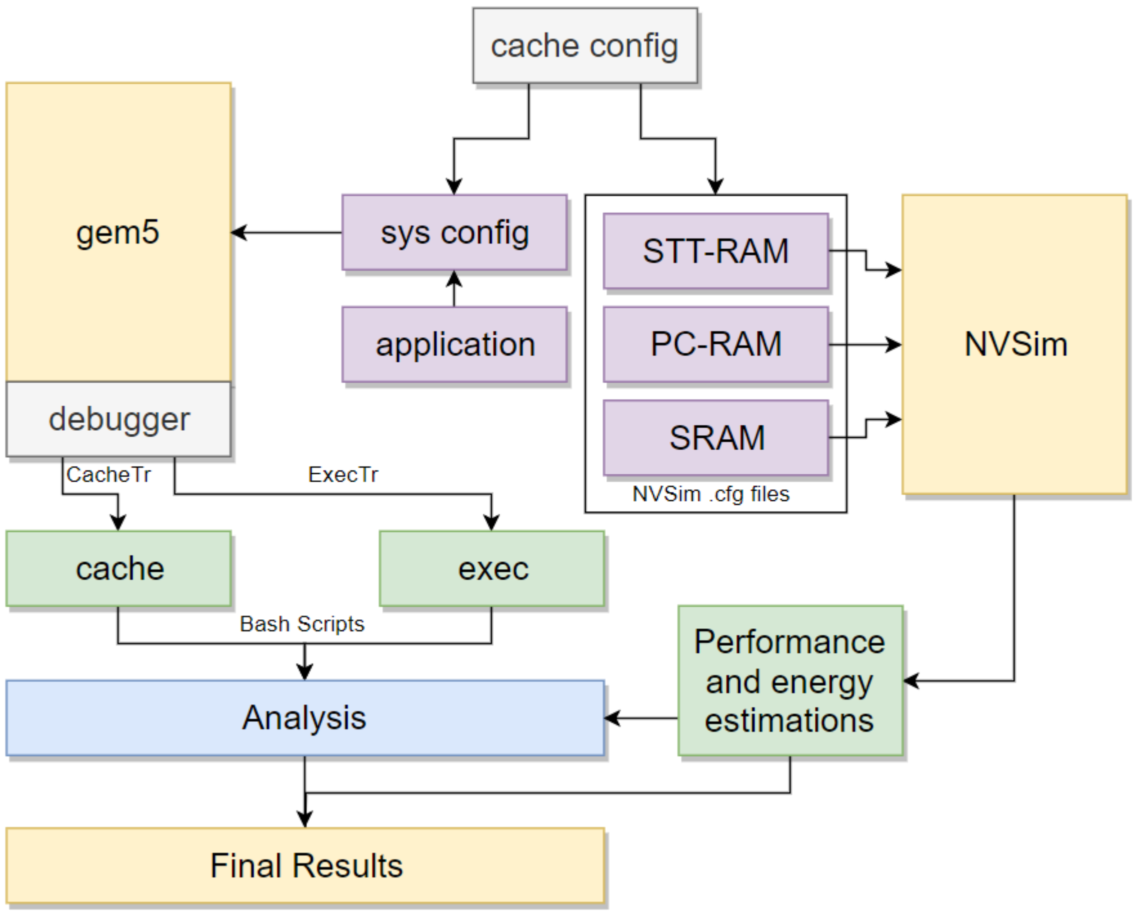


Figure 11. Illustration of the framework adopted in this thesis to obtain results.

In this thesis, the framework described in section 1.2 is implemented to generate the final results. Figure 11 illustrates the framework implementation flow used to obtain the final results.

The first stage is to determine the cache configuration. After trying multiple configurations, the cache sizes were set at 4kB for the L1 caches and 64kB for the L2cache. The gem5 configuration files were finalized as described in 4.3.1 with the architecture illustrated in Figure 10. NVSim `.cfg` files are also finalized according to the cache configuration. Applications were set to be simulated using the system configurations as well. The debugging flags described in 4.3.2 were also added to the simulator. However, the debugging file – referred to as the trace file – where the debugging flags write the trace information is huge considering that there is information written into it about every single access to every cache in the system as well as information about every instruction being executed by the CPU.

### 4.5.1 Bash Scripts Trace Parser

In order to organize the information in the trace files and count how many misses each instruction have caused, a bash script was written to execute at the end of each simulation. The bash script, named `trParser` or trace parser, will read the trace file line by line and checks for the "THIS IS A MISS" identifier written to the trace file by the `CacheTr` flag. However, since the `CacheTr` is triggered for every cache, the `trParser` will also identify which instruction causing the miss by reading the name of the cache where the miss occurred as well. Due to the cache `timing` mode feature set in the system configuration file, the CPU is idle until any cache miss is resolved including the instruction misses. Whenever the `CacheTr` flag is triggered, the next `ExecTr` trigger will contain the original instruction causing the miss triggering the `CacheTr` flag. `CacheTr` will print out a "THIS IS A MISS" identifier for the I-cache and the `ExecTr` will print the instruction causing the miss when the miss is resolved by the caches and the instruction is received by the CPU. This will result in printing the instruction right after printing the "THIS IS A MISS" identifier it triggered. Taking advantage of this, the trace parser can identify which instruction caused a miss during the simulation. These instructions are written into a temporary file named `ALLMISSESINST`. Figure 12 show the flow in which the `CacheTr` and `ExecTr` flags are triggered. The `ALLMISSESINST` file will contain all the instructions executed during the simulation and the "THIS IS A MISS" identifier one line before the instructions that caused I-cache misses.

The next step of the `trParser` is to use the `ALLMISSESINST` file to count the number of misses each instruction causes. To do so, the `trParser` reads the `ALLMISSESINST` file line by line to extract the instructions causing misses. For every miss occurrence,

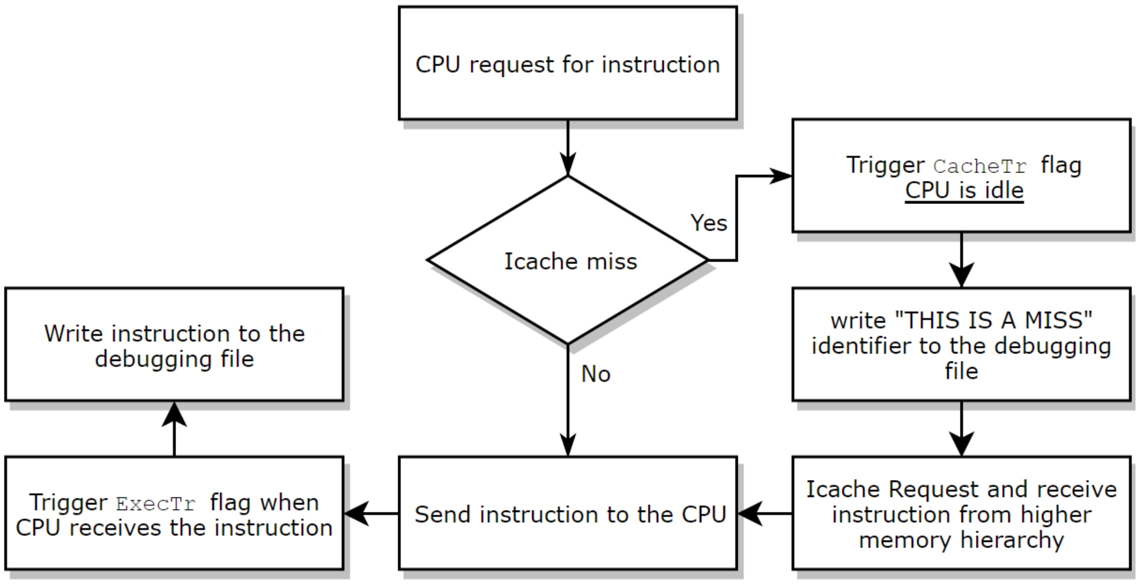

CPU request for instruction
Icache miss
Yes
No
Trigger CacheTr flag
CPU is idle
write "THIS IS A MISS" identifier to the debugging file
Icache Request and receive instruction from higher memory hierarchy
Send instruction to the CPU
Trigger ExecTr flag when CPU receives the instruction
Write instruction to the debugging file

Figure 12. The flow in which the `CacheTr` and `ExecTr` flags are triggered..

the `trParser` writes the instruction to `LISTOFINST` file. When this is done, the `trParser` counts the occurrences of each instruction found in the `LISTOFINST` file and sorts them from the highest number of occurrences to the lowest. At the end, `trParser` will generate a filed named `SORTEDINST` that contains all the instructions causing misses during the simulation the number of misses each instruction had caused.

### 4.5.2 NVSim Output Parser

For every time NVSim is used, an output file is generated with the estimations pertaining to the `.cfg` file passed to the simulator as input. A bash script, `NVSimParser` was written to run the NVSim simulator on multiple `.cfg` files in order. Each of the output files generated for each of the `.cfg` files is parsed by `NVSimParser` line by line. The `NVSimParser` will extract information regarding latency, dynamic energy, and leakage power of each of the `.cfg` files and write them into a file named `NVSimFINAL`. In the end, the `NVSimFINAL` file will have the latency, dynamic energy, and leakage power estimations for MRAM, PCRAM, and SRAM. These numbers are later used to evaluate the implementation of NV-SP.

## 4.6 Analysis

In order to evaluate the impact of the NV-SP and the I-cache before and after the incorporation of NV-SP into the memory hierarchy, an analysis flow was developed to calculate the effects of the NV-SP on the I-cache performance and energy. This flow uses the preliminary results from gem5 and NVSim along with the results from the bash scripts.

### Performance and Energy Evaluation before NV-SP

First, the latency of the I-cache is determined using the configuration of the caches.py in listing 1. During a miss, the I-cache issues two responses and one tag request: one tag request (Read request) from the CPU for an instruction, when the instruction is not in the cache, a response is issued asking for the instruction from the L2 cache. The L2 cache then responds to the I-cache followed by a response from the I-cache to The CPU. The latency of an I-cache miss is:

$$I_cL = L2R_l + (L1R_l * 3) \tag{1}$$

Where $I_cL$ is I-cache latency, $L1T_l$ is L1 cache tag_latency– parameter for cache Read latency, $L2R_l$ is L2 cache response_latency, and $L1R_l$ is L1 cache is response_latency– parameter for responding to a request. Using equation 1, the latency of the cache is 26 ns, or 26 cycles.

Second, using the statistics from the gem5 debugger, the total number of cycles, the total number of misses, and the total number of accesses to the I-cache. These statistics are used to obtain more information.

Since the total number of I-cache misses is known and from equation 1 the I-cache latency is obtained, to find the total number of cycles spent on all the misses:

$$TI_mC = I_cL * N_m \tag{2}$$

Where $TI_mC$ is the total instruction miss cycles, $I_cL$ is I-cache latency, and $N_m$ is the total number of I-cache misses. Using equation 2, the total latency caused by all the misses can be calculated.

Using the information obtained on the energy estimation for SRAM from NVSim, the total access energy can be determined. All the accesses to the I-cache are Read Accesses from the CPU– As shown by the statistics generated by the debugger. However, in case of a miss, when the I-cache receives information from the L2 cache, I-cache write energy also must be taken into account. Equation 3 shows the calculation for the total I-cache access energy:

$$TI_{AccessEnergy} = (SRAM_{ReadEnergy} * I_cA) + (SRAM_{WriteEnergy} * N_m) \tag{3}$$

Where $TI_{AccessEnergy}$ is the total I-cache access energy, $SRAM_{ReadEnergy}$ is NVSim Read energy estimation for SRAM, $I_cA$ is the total Read accesses to the I-cache, $SRAM_{WriteEnergy}$ is NVSim Write energy estimation for SRAM, and $N_m$ is the total number of I-cache misses.

The last equation is for determining the total power leakage of the I-cache. NVSim preliminary results also show the power leakage of the SRAM device per second. Since the total number of cycles is known from the gem5 debugger statistic, the total time can be determined. Since each cycle is one nanosecond in simulation time, the total time of simulation can be calculated. Equation 4 shows the calculation of the total power leakage for the I-cache:

$$TI_{PowerLeakage} = SRAM_{PowerLeackage} * Time_{execution} \tag{4}$$

Where $TI_{PowerLeakage}$ total power leakage for the I-cache, $SRAM_{PowerLeackage}$ is NVSim Read power leakage per second estimation for SRAM, and $Time_{execution}$ is the total execution time in seconds.

Using Equations 1, 2, 3, and 4, both the performance and energy evaluation of the I-cache can be done. By obtaining this information, comparing the performance and energy of the I-cache before the incorporation of the NV-SP can be achieved.

**Performance and Energy Evaluation after NV-SP**

In order to evaluate the impact of the NV-SP on the performance of the I-cache, the number of misses caused by each instruction must be added to the calculation. Since the latency of the Read and Write operations differ depending on the technology used in the NV-SP, and the NV-SP under evaluation for different sizes, new latency, and energy-related equations must be used.

First, different sizes of the NV-SP can store different numbers of instructions. In this thesis, three sizes are used. Since the architecture is 32-bit x86, the number of instructions that can be stored in the NV-SP are 250, 500, and 1000 instructions for sizes 1 kB, 2kB, and 4kB respectively. In accordance, the number of instructions causing the highest number of misses are moved to the NV-SP to fill. Meaning that in case the NV-SP is 1kB, for example, the 250 instructions causing the highest number of misses, found in the `SORTEDINST` file, are moved to the NV-SP. This can greatly reduce the number of misses happening in the I-cache as well as leave more space for instructions causing fewer misses. This reduction of misses can greatly affect the latency of the total number of misses in the cache.

Equation 5 shows the calculation of the total instruction miss cycles of I-cache after incorporating the NV-SP:

$$SP\_TI_mC = I_cL * (N_m - NVSP_m) \tag{5}$$

Where $SP\_TI_mC$ is the total instruction miss cycles of I-cache after incorporating the NV-SP, $I_cL$ is I-cache latency, $N_m$ is the total number of I-cache misses, and $NVSP_m$ is the total number of misses caused by the instructions stored in the NV-SP.

Similarly, the access energy to the I-cache will also have an effect on the performance after incorporating the NV-SP, Equation 6 shows the calculation of the access energy of the I-cache after the incorporation of the NV-SP:

$$SP\_TI_{AccessEnergy} = (SRAM_{ReadEnergy} * I_cH) + (SRAM_{WriteEnergy} * (N_m - NVSP_m)) \tag{6}$$

Where $SP\_TI_{AccessEnergy}$ is the total access energy of I-cache after incorporating the NV-SP,$SRAM_{ReadEnergy}$ is NVSim Read energy estimation for SRAM, $I_cH$ is I-cache number of hits, $SRAM_{WriteEnergy}$ is NVSim Write energy estimation for SRAM, $N_m$ is the total number of I-cache misses, and $NVSP_m$ is the total number of misses caused by the instructions stored in the NV-SP.

The next step is to calculate the total number of cycles when the NV-SP was used. This is equal to the total number of Read accesses to the NV-SP as well as the number of instruction stored in the NV-SP since that is the number of Write operations to the

NV-SP. The number of Read accesses to the NV-SP is the total number of misses caused by the instructions with the highest number of misses, since the NV-SP is only accessed to retrieve these instructions. The number of Write accesses is the same as the number of instructions stored in the NV-SP since they are only written once and never evicted.

From NVSim, the access latency and energy of the NVM technologies is obtained. For all the NVM technology, equation 7 describes the calculation of the total access latency cycles of the NV-SP. Equation 8 describes the calculation of the total access energy of the NV-SP for all the NVM technologies:

$$NVM_{AccessLatency} = NVM_{ReadLatency} * NVSP_m + NVM_{WriteLatency} * NVSP_i \tag{7}$$

$$NVM_{AccessEnergy} = NVM_{ReadEnergy} * NVSP_m + NVM_{WriteEnergy} * NVSP_i \tag{8}$$

Where $NVM_{AccessLatency}$ is the total NVM access latency, $NVM_{AccessEnergy}$ is the total NVM access energy, $NVM_{ReadLatency}$ is NVSim Read latency estimation for the NVM technology, $NVM_{ReadEnergy}$ is NVSim Read energy estimation for the NVM technology, $NVM_{WriteLatency}$ is NVSim Write latency estimation for the NVM technology, $NVM_{WriteEnergy}$ is NVSim Write energy estimation for the NVM technology, $NVSP_m$ is the total number of misses caused by the instructions stored in the NV-SP, and $NVSP_i$ is the total number of instructions stored in the NV-SP.

To calculate the total power leakage of the NVM, a similar equation to equation 4 is used. From NVSim, the power leakage per second of the NVM technologies is known. However, the power leakage of the NV-SP is only calculated for the total number of cycles it was accessed. Equation 9 shows the calculation of the power leakage of the NVM technologies:

$$NVM_{PowerLeakage} = NVM_{PowerLeackage} * Time_{NVMaccess} \tag{9}$$

Where $NVM_{PowerLeakage}$ is the total NVM power leakage, $NVM_{PowerLeackage}$ is NVSim power leakage estimation for the NVM technology, and $Time_{NVMaccess}$ is the total access time to the NV-SP.

### Results' Normalization

Since there are many different applications used to evaluate the performance and energy of the different NVM technologies, there must be a scale where the results can be comparable. Therefore, in order to increase simplicity for the many results obtained from the previous equations, all results are normalized to 1. The normalization base is the original SRAM performance before incorporating the NV-SP. Applications are ran through simulations to obtain preliminary results for each application. Next, equations 1, 2, 3, 4, 5, 6, 7, 8, and 9 are applied to these results for each application. In order to normalize the results, the I-cache initial results are equal to the execution time. Equation 10 shows the normalization calculation for the NVM performance:

$$NVM_{performance} = \frac{SP\_TI_mC + NVM_{AccessLatency}}{Time_{execution}} \tag{10}$$

Where $NVM_{performance}$ is the total NVM performance measurement in relation to the original simulation time, $SP\_TI_mC + NVM_{AccessLatency}$ will be equal to the new time of execution, and $Time_{execution}$ is the total execution time in seconds. By dividing the new execution time after incorporating the NV-SP by the original execution time of each application, a scale is developed to evaluate the performance of the NV-SP for all the applications using all the different NVM technologies at different sizes. A similar process is done to obtain the Measurements for the NVM energy evaluation. Since the $TI_{AccessEnergy}$ is equivalent to the total energy of the I-cache before incorporating the NV-SP, it is used as the base for the normalization oin the case of evaluating NVM energy. Equation 11 shows the normalization of the NVM energy:

$$NVM_{Energy} = \frac{SP\_TI_{AccessEnergy} + NVM_{AccessEnergy}}{TI_{AccessEnergy}} \tag{11}$$

# 5 Computation in Memory (CiM) with NVM

For more than 150 years, it was believed that resistors, capacitors, and the inductors are the only three basic passive circuit elements. In 1971, Chua [7] set forth the logical and scientific basis for the existence of a fourth basic circuit element, along with the resistor, capacitor, and the inductor. Chua theorized that such circuit element is a two-terminal circuit element with no internal power supply – the **Memristor**. It's functional relationship, memristance, is between the magnetic flux ($\Phi$) and charge ($q$). It was 37 years later that researchers at the Hewlett Packard Labs introduced the first working prototype of a memristor in 2008 [8].

Memristors, along with their non-linear resistance, can "remember" [32] the recent resistance of the device when the voltage is turned off. The characteristics of memristance and the ability to store information give memristors a great advantage for nanoscale electronics. It has been shown that memristors can be used for logical operations [17] [16] as well. Non-volatile Memory, in the form of memristors, can offer a possible path towards efficient and configurable memory arrays that can be programmed to complete complex logical operations. In other words, memristor-based NVM devices have the potential to complete tasks of Computation in Memory (CiM).

It has been demonstrated that logical operations such as *AND* and *OR* can be achieved with as little as two memristors [16] as shown in figure 13. An *IMPLY* gate can also be achieved using two memristors as demonstrated in [18].

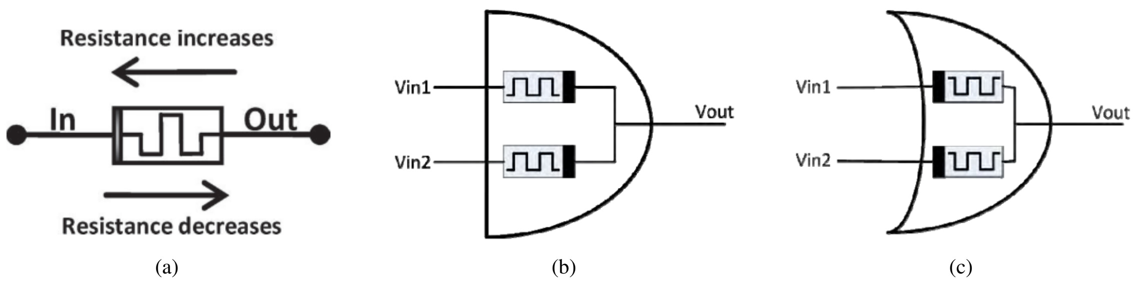
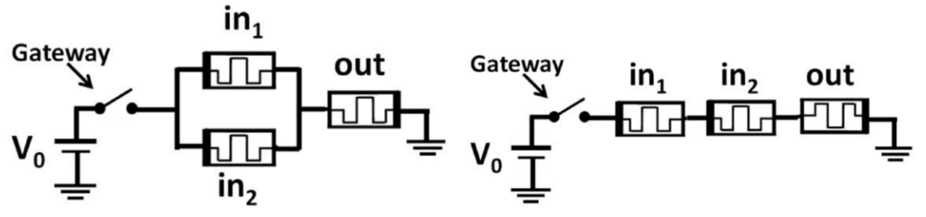


Figure 13. Demonstration of a: (a) memristor [17], and memristor-based (b) *AND* and (c) *OR* logic gates [16].

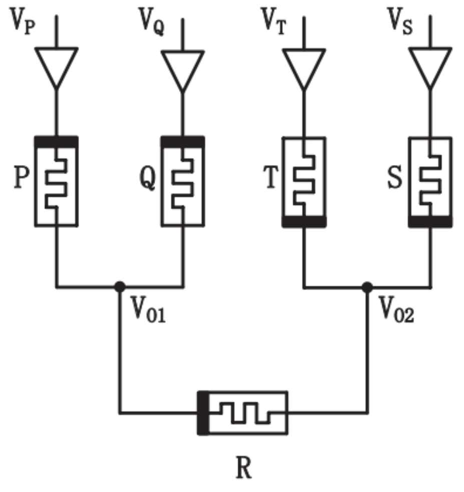
While *AND* and *OR* operations are simple, other logical operations such as *NOR* and *NAND* can take three memristors [17] shown in figure 14.



(a) (b)

Figure 14. Demonstration of memristor-based (a) *NOR* and (b) *NAND* logic gates [17].

In [15], an XOR gate was created using a memristor-based *AND* gate and a memristor-based *OR* gate along with a fifth memristor to store the output of the XOR gate. While traditionally an XOR gate is created using a *NAND*, an *AND*, and an *OR* gates, memristor-based logic offered the same functionality of an *XOR* gate using only two logical gates.



(a)

| $V_P\,(V_T)$ | $V_Q\,(V_S)$ | $V_{01}$ | $V_{02}$ | $M_R$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 |

(b)

Figure 15. Demonstration of (a) memristor-based *XOR* and (b) truth table of memristor-based *XOR* [15].

Since memristor-based logical operations are possible, several applications for memristor-based logic were proposed. Most notably, [19] and [15] implemented fully functional memristor-based Full Adders, [20] created and evaluated a memristor-based crossbar to implement *dot* product for Vector-Matrix Multiplication used in efficient neural networks, while [21] demonstrated that such crossbars can also be used for precise signal and image processing.

## 5.1 NVM Security

Not only that NVMs can offer analog logic operations, but they can also be used to increase security by decreasing the level of data management within the system. By doing so, NVM-based CiM can offer secure implementations of common mathematical transformation algorithms used in cryptography.

### 5.1.1 AES S-Box

As discussed in section 4.2.1, different cryptographic algorithms are used to evaluate the performance, access energy, and power leakage of the proposed NV-SP. However, in this section, the focus will be drawn towards the Advanced Encryption Standard that is based on the implementation of the Rijndael S-box. The Rijndael S-box [33] is a lookup table implementation created by Joan Daemen and Vincent Rijmen as part of the National Institute for Standards and Technology's (NIST) contest for the new Advanced Encryption Standard (AES), which Rijndael had eventually won in 2001 [34].

The goal of the design of Rijndael S-box is to create a transformation algorithm that is highly resistant to linear and differential cryptanalysis. This is achieved by minimizing the correlations between the input bits and output bits of the linear transformations [33]. The Rijndael S-box maps an input of 8-bits, $i$, to 8-bit output, $s$. The following steps describe the transformation algorithm used to map the input to the output:

- Both of the input and output are polynomial interpretations over Rijndael's finite field, also known as Galois field $GF(2^8)$.
- Map input into it's multiplicative inverse, $b$, using the Nyberg transformation [35]:

$$GF(2^8) = \frac{GF(2)[x]}{x^8 + x^4 + x^3 + x + 1} \tag{12}$$

- Transform the multiplicative inverse, $b$, using the affine transformation shown in equation 13. This equation is an expression of the summation of multiple rotations of multiplicative inverse as a vector. It is important to point out that the addition in this equation is an $XOR$ operation:

$$s = b \oplus (b \lll 1) \oplus (b \lll 2) \oplus (b \lll 3) \oplus (b \lll 4) \oplus 63_{16} \tag{13}$$

$$S\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \end{bmatrix} = M\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} b\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} \oplus c\begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \tag{14}$$

Equation 14 describes the affine transformation used in for Rijndael S-Box, expressed in equation 13, as a vector-matrix multiplication.

### 5.1.2 NVM-based AES S-Box

In this section, memristor-based logical operations are used to implement the affine transformation described in equation 14. Since the addition used in the equation is an XOR operation, crossbars described and implemented in [20] and [21] can not be used since those crossbars are *dot* product multiplications where the addition is an *OR* operation. For this implementation, the $XOR$ gate described in [15] is used.

Similar to the implementation of [15], the implementation of the $XOR$ gate included two SPICE memristor models: a standard memristor model described in [36] and a threshold memristor described in [37]. Since the vector matrix multiplication ($M.b$) has an $XOR$ operation as addition, the resulting vector $V$ can be produced using only 5 $XOR$ operations. The reasoning behind this is that every row in matrix $M$ has three 0 elements in sequence. Knowing that

$$s_i = \bigoplus_{i=0}^{7} \bigoplus_{j=0}^{7} (M_{ij}.bi) \oplus c_i \tag{15}$$

and assuming for every multiplication $M_{ij}.bi$ for $s_i$ that all the *dot* product where the $M_{ij}$ element is equal to **"1"** are $XOR$ed and the result is a boolean value $x_i$. When the *dot* product where the $M_{ij}$ element is equal to **"0"**, the result *dot* product will always equal to **"0"**. The result of the vector-matrix multiplication is a vector $V$ :

$$V_i = x_i \oplus 0 \oplus 0 \oplus 0 \tag{16}$$

where $x_i$ can have two potential values as follows:

$$V_i(x_i) = \begin{cases} 1, & 1 \oplus 0 \oplus 0 \oplus 0 = 1 \\ 0, & 0 \oplus 0 \oplus 0 \oplus 0 = 0 \end{cases} \tag{17}$$

After simplification of $M_{ij}.bi$, equation 17 shows that the final result of the multiplication will always equal to the value of $x_i$. This means that the three 0 elements in every matrix row have no effect on the $XOR$ summation of the *dot* product of the vector-matrix multiplication. Furthermore, since the By reducing the needed $XOR$ operations, a simple, yet effective, implementation can be designed. Resulting vector $V$ shown in equation 18:

$$V = \begin{bmatrix} b_0 \oplus b_4 \oplus b_5 \oplus b_6 \oplus b_7 \\ b_0 \oplus b_1 \oplus b_5 \oplus b_6 \oplus b_7 \\ b_0 \oplus b_1 \oplus b_2 \oplus b_6 \oplus b_7 \\ b_0 \oplus b_1 \oplus b_2 \oplus b_3 \oplus b_7 \\ b_0 \oplus b_1 \oplus b_2 \oplus b_3 \oplus b_4 \\ b_1 \oplus b_2 \oplus b_3 \oplus b_4 \oplus b_5 \\ b_2 \oplus b_3 \oplus b_4 \oplus b_5 \oplus b_6 \\ b_3 \oplus b_4 \oplus b_5 \oplus b_6 \oplus b_7 \end{bmatrix} \tag{18}$$

### 5.1.3 Memristor-based S-Box implementation

The memristor-based S-box implementation can take advantage of this by reducing the number of $XOR$ gates needed to implement the multiplication array. A five-input array is used to $XOR$ the elements in each row $V_i$. Each array consists of 4 $XOR$ gates as shown in figure 16 (a):

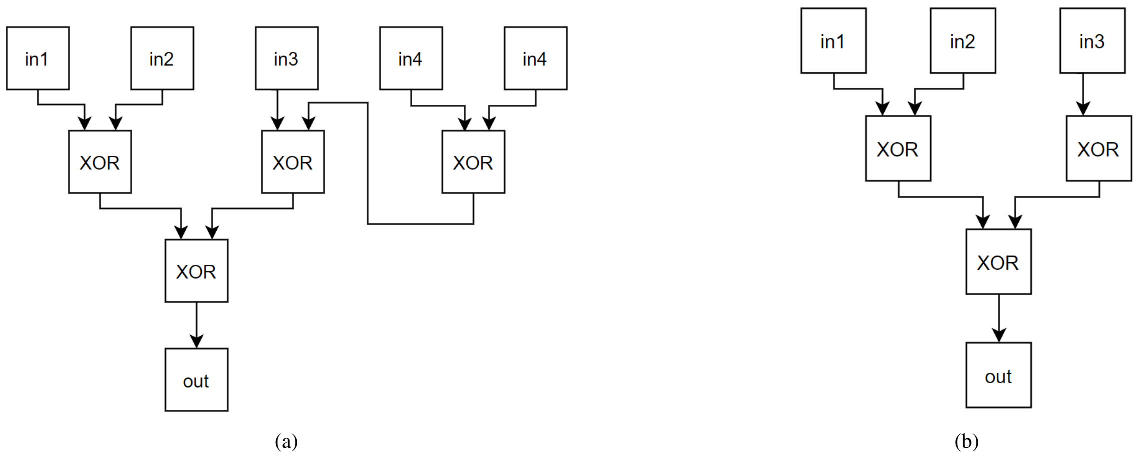


Figure 16. Demonstration of the memristor-based $XOR$ array used to implement (a) the S-box affine transformation and (b) the inverse S-box affine transformation.

This means that for every row $i$ of the matrix, five $XOR$ gates can be used to produce the $s_i$ where five $XOR$ gates are placed in an array to $XOR$ the $M_{ij}$ value where it is equal to **"1"** to produce the $V_i$ value. The fifth $XOR$ gate is placed to $XOR$ the $V_i$ and the $c_i$ to produce the final $s_i$ value.

To test this, let $i$ be an 8-bit input. $i = 10101010_2$ or $AA_{16}$, the leftmost bit being the least significant. To find the Sbox equivalent of $i$, we calculate the multiplicative inverse of $i$, which, in this case, is $00010010_2$ or $12_{16}$. By applying the $V$ on $12_{16}$:

$$V = \begin{bmatrix} b_0 \oplus b_4 \oplus b_5 \oplus b_6 \oplus b_7 \\ b_0 \oplus b_1 \oplus b_5 \oplus b_6 \oplus b_7 \\ b_0 \oplus b_1 \oplus b_2 \oplus b_6 \oplus b_7 \\ b_0 \oplus b_1 \oplus b_2 \oplus b_3 \oplus b_7 \\ b_0 \oplus b_1 \oplus b_2 \oplus b_3 \oplus b_4 \\ b_1 \oplus b_2 \oplus b_3 \oplus b_4 \oplus b_5 \\ b_2 \oplus b_3 \oplus b_4 \oplus b_5 \oplus b_6 \\ b_3 \oplus b_4 \oplus b_5 \oplus b_6 \oplus b_7 \end{bmatrix} = \begin{bmatrix} 0 \oplus 1 \oplus 0 \oplus 0 \oplus 0 \\ 0 \oplus 1 \oplus 0 \oplus 0 \oplus 0 \\ 0 \oplus 1 \oplus 0 \oplus 0 \oplus 0 \\ 0 \oplus 1 \oplus 0 \oplus 0 \oplus 0 \\ 0 \oplus 1 \oplus 0 \oplus 0 \oplus 1 \\ 1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 \\ 0 \oplus 0 \oplus 1 \oplus 0 \oplus 0 \\ 0 \oplus 1 \oplus 0 \oplus 0 \oplus 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \oplus c \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = S \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \qquad (19)$$

The resulting vector $S$, is an 8-bit output as $10101100_2$ or $AC_{16}$, which is the Sbox equivalent of input $i$.

The decryption process can also be implemented by applying the same method to the inverse S-box equation 20, and expressed as a vector-matrix multiplication in 21:

$$b = (s \lll 1) \oplus (s \lll 3) \oplus (s \lll 6) \oplus 5_{16} \tag{20}$$

$$b \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} = Inv\_M \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} S \begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \end{bmatrix} \oplus Inv\_c \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{21}$$

By applying the same method used to generate the $V$ vector from the $M_{ij}.bi$ vector-matrix multiplication of the affine transformation of the S-Box, the inverse vector $Inv\_V$ is also generated from the $Inv\_M_{ij}.Si$ vector-matrix multiplication as follows:

$$Inv\_V = \begin{bmatrix} b_2 \oplus b_5 \oplus b_7 \\ b_0 \oplus b_3 \oplus b_6 \\ b_1 \oplus b_4 \oplus b_7 \\ b_0 \oplus b_2 \oplus b_5 \\ b_1 \oplus b_3 \oplus b_6 \\ b_2 \oplus b_4 \oplus b_7 \\ b_3 \oplus b_5 \oplus b_0 \\ b_1 \oplus b_4 \oplus b_6 \end{bmatrix} \tag{22}$$

The implementation of the array to generate the inverse vector $Inv\_V$ consists of 3 $XOR$ gates as shown in figure 16 (b).

By using the implementations above, memristor-based logic can be used to implement the AES S-box affine transformations, both forward and inverse, shown in figures 14 and 21. The data can be encrypted and decrypted in memory at very little cost of energy and time. By doing so, memristor-based logic is demonstrated to offer in-memory computing capabilities with very little data management, allowing higher accuracy and fault management. Such implementation offers high security as data is directly stored as encryption with no copy of the actual data existing anywhere else. The data can be decrypted in memory as it is being retrieved from memory. The arrays can be implemented as stand-alone components, but it is possible to program a collection of memristors (or NVM cells) in an NVM memory device to complete these $XOR$ operations while also using the same memristors for storage.

### 5.1.4 AES Sbox implementation comparison

In this section, the number of elements needed for the different implementations of the AES S-Box are presented. Three implementations are used in the comparison when using NVM technology to store/generate the S-Box and inverse S-Box:

- AES S-Box as a look up table. This implementation is the one used in the AES application used in the simulation. The S-Box and the inverse S-Box are stored as a Lookup Tables stored in memory. The S-Box S-Box and the inverse S-Box are accessed by the algorithm during execution to find the S-Box and inverse S-Box equivalents.
- AES S-Box as code. For the purposes of this thesis, a C Language implementation of the S-Box was developed. In this implementation, the S-box is not stored as a Lookup Table, but instead it is embedded in code. The S-Box and inverse S-Box equivalents are generated during execution. Listing 5 presents the C Language code implementations for generating the S-Box and the inverse S-Box. This code was tested successfully with a 32-bit instruction.
- As shown in section 5.1.3, implementation of the S-Box and inverse S-Box using NVM-based CiM is achieved. Such implementation is highly secure and it is possible at a very low cost of energy and time.

Listing 5. S-Box C language Implementation.

```
makesbox(in) {
x = multiplicative_inverse(in)
x = x ^ ((x << 1) | (x >> 7)) ^ ((x << 2) | (x >> 6)) ^ ((x <<
   3) | (x >> 5)) ^ ((x << 4) | (x >> 4)) ^ 0x63;
return x;
}
invsbox(s) {
x = ((s << 1) | (s >> 7)) ^ ((s << 3) | (s >> 5)) ^ ((s << 6) |
   (s >> 2)) ^ 0x05;
x = gmul_inverse(x);
return x;
}
```

Each of those implementations offer advantages but also come with a cost. Lookup Tables increase the performance of the AES algorithm, but it is insecure as it is vulnerable to different types of attacks. Implementing the S-Box as C code reduces the performance of the algorithm since each time S-Box equivalents are needed the code will generate them, but it is more secure than using a Lookup Table. The NVM-based S-Box can be complicated to implement and requires programming the NVM cells to complete the logical operations.

Table 3 presents the total storage and number of NVM elements each implementations require. The Lookup Tables implementations will require storing the S-Box in memory. Each of the S-Boxes' lookup table is made of 16 rows and 16 columns, which means that there are 256 elements in each table. Since the S-Box takes an input of 8 bits and gives output of 8 bits, each element in the lookup table will require one Byte of storage. This means that each lookup table requires 256 Bytes making a total of 512 Bytes of storage for both the tables. Sine each element can store one bit, a total of 4096 elements needed to store the lookup tables in memory.

The C code implementation, assuming the the code is stored in memory and never evicted, is accessed whenever the algorithm needs to generate either the S-Box or the inverse S-Box equivalents. Non-optimized 32-bit assembly code was generated for both of the functions used in the C Language code. The assembly code showed that 50 instructions are required to complete the function `makesbox()` and 39 instructions to complete the function `invsbox()`. Each of those instructions is 32 bits, equalling to 4 Bytes per instruction. The `makesbox()` will require 1600 elements while `invsbox()` will require 1248 elements, totalling to 2848 elements to store the code.

As for the NVM-implementation of the S-Box, the AES equivalent is generated for a 32-bit data for the purpose of consistency with the other two implementations. Section 5.1.3 established that for an 8 bit input only 5 bits and 3 bits of that input are required to be $XOR$ed to generate the S-Box and inverse S-box equivalents respectively. This means, for a 32-bit input, 4 arrays are needed. For the S-Box equivalent, each array has 4 $XOR$ gates, each gate required 5 memristors, totalling to 80 memristors for 32 bit input to generate the S-Box equivalent. As for the inverse S-Box equivalent, each array has 3 $XOR$ gates, each gate required 5 memristors, totalling to 60 memristors for 32 bit input to generate the inverse S-Box equivalent. This totals to 140 memristors, or NVM elements needed to generate the S-Box equevelant using NMV-based CiM.

Table 3 shows that implementing the S-Box using NVM-based CiM will greatly

Table 3. The total storage and number of elements needed for each of the AES S-Box implementations.

| S-Box Implementation | Storage needed | Number of storage elements needed |
|---|---|---|
| Lookup Tables | 256 Bytes for each of the S-Box and inverse S-Box | 4096 elements |
| C code | 50 32-bit instructions for S-Box<br>39 32-bit instructions for inverse S-Box | 2848 elements |
| NVM-Based | 80 elements for S-Box<br>60 elements for inverse S-Box | 140 NVM elements |

reduce the number of elements needed to complete the encryption and decryption of the data. The NVM-based implementation of the S-Box and inverse S-Box shows that using CiM can bring huge advantages such as greatly reduce the energy cost and number of storage elements needed to process and store data. By doing complex logical operations in memory, data that can be generated when needed instead of being stored in memory. This will further improve performance by reducing time and increase the energy efficiency of the system.

# 6 Experimental results

In this chapter, results from the experiment are shown and discussed. In section 1, preliminary results from gem5 and NVSim are shown. Results from gem5 are shown and compared for each of the workloads while results from NVSim estimations of performance and energy for each of the technologies.

## 6.1 Preliminary Results

Preliminary results from gem5 and NVSim are crucial for this thesis. These results will be used to draw conclusions on the performance of the I-cache before and after the implementation of the NV-SP. These results are also utilized to formulate the final results.

### 6.1.1 gem5 Simulation Results

For the gem5 simulator, the preliminary results are obtained from the statistical output of simulator. The simulator generates a statistics file after each simulation. This file contains information on the total number of accesses to all the caches, the number of instructions simulated, and information regarding the timing of the simulation and overall statistics. For the purposes of this thesis, information regarding the number of accesses, number of hits, and number of misses regarding the I-cache are extracted from the statistics file. The duration of the simulation (simulator time, not real time) are also used to compare the execution time before and after the implementation of NV-SP.

Table 4 shows the number of accesses, hits, misses, and miss rate of the I-cache for each of the Cryptographic Algorithms assigned to the architecture during simulation.

Table 4. I-cache simulation results for the Cryptographic Algorithms.

| | AES | DES | ARCFOUR | Twofish | Blowfish | ROT13 |
|---|---|---|---|---|---|---|
| Accesses | 1134165 | 1218380 | 237783 | 419250 | 1719628 | 193483 |
| Hits | 1115893 | 1215756 | 236108 | 413552 | 1717670 | 191804 |
| Misses | 18258 | 2764 | 1852 | 5594 | 1953 | 1676 |
| Miss Rate | 1.60% | 0.22% | 0.77% | 1.33% | 0.11% | 0.86% |

From table 4, the number of misses compared to the number of the total number of accesses to the I-cache are fairly low, this is largely due to the small size of the workloads assigned to the architecture. While the number of misses is low, the miss rates of the workloads are fairly high. This is due to the small size of the I-cache. A number of compulsory misses occur while the I-cache is being populated with the first instructions added to the I-cache, a low number of conflict and capacity misses actually occur. Furthermore, AES and Twofish workloads show a much larger gap of miss rate compared to the other workloads, which can be attributed to the fact that both these algorithms are using an Substitution Box (Sbox) as part of the encryption and decryption process. Access to the Sbox repeatedly at different stages of the algorithm causes instructions that access the Sbox to be evicted from the I-cache, thus causing both conflict and capacity misses to occur.

Table 5 shows the number of accesses, hits, misses, and miss rate of the I-cache for each of the Non-Cryptographic Algorithms assigned to the architecture during simulation.

Table 5. I-cache simulation results for the Non-Cryptographic Algorithms.

| | Djpeg | Cjpeg | bzip2-d | bzip2-c | specrand |
|---|---|---|---|---|---|
| Accesses | 419250 | 1719628 | 97790934 | 97790934 | 44498463 |
| Hits | 413552 | 1717670 | 97533505 | 97153453 | 42999835 |
| Misses | 6460 | 6934 | 257412 | 637481 | 1489776 |
| Miss Rate | 1.54% | 0.40% | 0.26% | 0.65% | 3.34% |

Table 5 shows that the Non-Cryptographic Algorithms have shown a similar pattern to the Cryptographic Algorithms with low misses compared to the total number of accesses to the I-cache. Bzip2 compression and decompression had the same number of accesses to the I-cache, but bzip2 compression had more than double the misses caused by bzip2 decompression. However, this difference in I-cache misses for both the bzip2 modes does not exclude them from the aforementioned pattern. Nonetheless, specrand is an exception to the pattern. The miss rate of specrand is large for low miss rates expected from an I-cache. Although specrand, had less than half of the accesses to the I-cache compared to bzip2, specrand caused a little more than double the misses caused by bzip2.

### 6.1.2 NVSim Estimation Results

For the NVSim, the preliminary results are obtained from the direct output of the modeling program. The output of the modeling program contains estimations regarding the area, performance, access energy, and power leakage of the memory cell defined in the configuration file passed as input. Estimations for access latency in Table 6 and access energy and power leakage per second in Table 7 were obtained for 1kB, 2kB, and 4kB PCRAM and MRAM to be used in measuring the performance and energy of the NV-SP. However, since SRAM is only used to estimate the performance and energy of the I-cache before and after incorporating the NV-SP into the system, estimations for 4kB were only obtained.

#### NVSim Access Latency Results

Table 6. NVSim Read and Write access latency estimations for SRAM, PCRAM, and MRAM.

| | **1 kB** | | **2 kB** | | **4 kB** | |
|---|---|---|---|---|---|---|
| | Read | Write | Read | Write | Read | Write |
| SRAM | 0.141 ns | 0.112 ns | 0.159 ns | 0.149 ns | 0.324 ns | 0.227 ns |
| PCRAM | 0.153 ns | 150.091 ns | 0.159 ns | 150.096 ns | 0.267 ns | 150.122 ns |
| MRAM | 1.582 ns | 10.124 ns | 1.584 ns | 10.125 ns | 1.766 ns | 10.206 ns |

Table 6 shows the NVSim latency estimations for each of the memory technologies used in this thesis. PCRAM clearly has the advantage of Read latency over MRAM. Since PCRAM Read operation, explained in section 3.1.2, is similar to elecrical discharge, the information stored in PCRAM takes little time to be released from the memory cell using a low voltage pulse. Since information is stored in the MRAM cell as the MTJ's resistance, an electrical circuit is connected to the bottom electrode of the MTJ to interpret the stored information. This increases the read latency of the MRAM memory. PCRAM also have an advantage over SRAM at 4kB, but this advantage is not noteworthy.

Nonetheless, PCRAM is overwhelmingly outperformed when it comes to Write latency. MRAM requires less time to write into the MRAM memory than PCRAM due to the nature of the Write operations for each of the technologies. MRAM takes a fairly long time to Write into memory compared to SRAM due to the rotation time of the MTJ Free Layer, PCRAM requires a longer time to $SET$ the PCRAM cell. This longer charge time is attributed to the long waiting time needed for the long *SET* pulses that cause the phase-changing material to heat into crystallization temperature ($T_{cryst}$).

## NVSim Access Energy and Power Leakage Results

Table 7. NVSim Read and Write access energy estimations for SRAM, PCRAM, and MRAM.

| Size | Estimation | SRAM | PCRAM | MRAM |
|---|---|---|---|---|
| **1 kB** | Read Energy | – | 0.001 nJ | 0.008 nJ |
| | Write Energy | – | 3.241 nJ | 0.032 nJ |
| | Power Leakage/s | – | 8.303 mW | 0.762 mW |
| **2 kB** | Read Energy | – | 0.001 nJ | 0.008 nJ |
| | Write Energy | – | 3.241 nJ | 0.032 nJ |
| | Power Leakage/s | – | 16.56 mW | 2.892 mW |
| **4 kB** | Read Energy | 0.004 nJ | 0.004 nJ | 0.027 nJ |
| | Write Energy | 0.003 nJ | 10.533 nJ | 0.106 nJ |
| | Power Leakage/s | 20.189 mW | 31.771 mW | 5.377 mW |

Table 7 shows the NVSim energy estimations for each of the memory technologies used in this thesis. In correspondence to the Read latency in Table 6, PCRAM requires much less Read energy since the read energy required is very low compared to the Read energy required for the electrical circuit connected to the bottom electrode of the MTJ used in MRAM. However, the same long pulses needed for PCRAM in the *SET* operation are also attributed to higher access energy needed for PCRAM to perform a Write operation. PCRAM features a very low Read access Energy, which is overshadowed by the very high Write Access energy. In comparison, MRAM features slightly higher Read access energy compared to PCRAM and SRAM and low Write access energy compared to PCRAM. In fact, MRAM maintains lower Write energy that is 100 times lower of PCRAM write energy at all sizes. In the energy estimations, MRAM clearly more advantageous over PCRAM. Although PCRAM has a slightly lower Read access energy, this difference is overshadowed by the Write access energy advantage that MRAM holds over PCRAM. MRAM's low Read and Write access energies are somewhat high compared to SRAM

Read and Write access energy. However, as demonstrated later, MRAM's disadvantage of higher access energy compared to SRAM's is overshadowed by the much higher power leakage that SRAM has.

As for power leakage, MRAM clearly is more advantageous as it maintains a very low power leakage at all sizes compared to PCRAM. While a significant increase of power leakage in PCRAM is very apparent as size increases, this can be attributed to the long duration and high voltages of the SET and RESET pulses needed for the PCRAM to reach the $T_{cryst}$ and $T_{melt}$ temperatures. However, the factor that sets the power leakages of PCRAM and MRAM apart from the power leakage of SRAM, is that SRAM needs to be constantly operating or else the data stored in SRAM will be lost. This means that the duration of time in which power leakage happens in the SRAM is the total execution time. The non-volatile nature of the PCRAM and MRAM mitigates this issue. MRAM and PCRAM do not need to constantly operate, they only need to operate when they are being accessed to write or read data. The high power leakage of PCRAM maybe higher than that of SRAM, but PCRAM is "on" far less time than SRAM, making the power leakage of PCRAM less of an issue.

## 6.2 NV-SP Evaluation

This section focuses on analyzing the final results to evaluate the performance, energy access, and power leakage of the NV-SP while being implemented using MRAM and PCRAM.

### 6.2.1 NV-SP Performance evaluation

#### MRAM Performance

Figure 17 shows the execution time of all the applications when using MRAM technology in implementing the NV-SP. It shows clearly that the MRAM NV-SP helped improve the execution time up to 22.35% and 16.49% for specrand and AES applications respectively. The worst case was for the Blowfish cipher with only 1% improvement. Nonetheless, after introducing the MRAM NV-SP to the system, all the applications showed performance improvement for all the MRAM NV-SP sizes.

Figure 17, along with tables 4 and 5, draw a correlation between the improved time of ex-

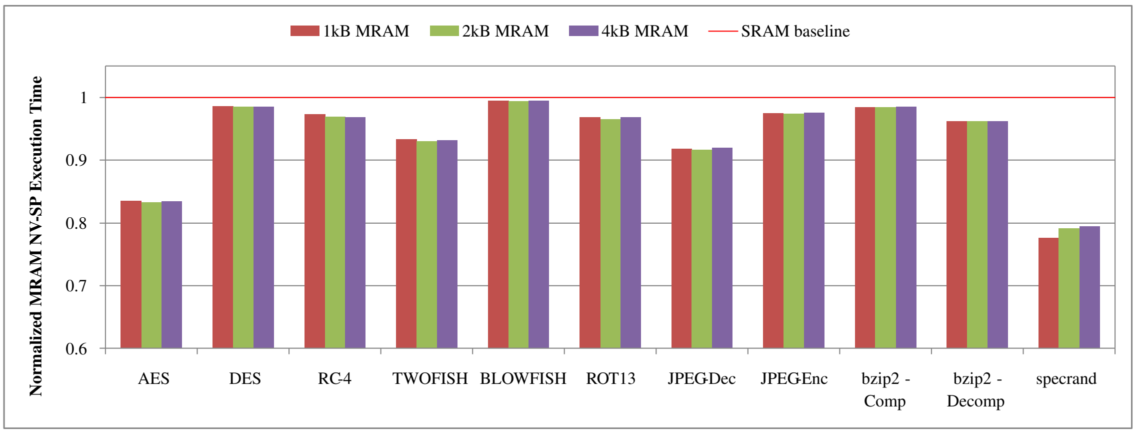


Figure 17. Performance evaluation when incorporating the MRAM NV-SP.

ecution and the miss rate of the application before incorporating the NV-SP. Applications with higher execution time improvement experienced higher miss rate before MRAM NV-SP was introduced to the system. specrand and AES had the highest miss rates with 3.34% and 1.60% respectively, while blowfish only had a miss rate of 0.11% making it the lowest of all the applications. Higher miss rate meant that there is more execution time spent on miss latency and miss mitigation. When moving instructions with the highest misses to the MRAM NV-SP, miss latency becomes less of an issue as shown by the improvement in performance for all the applications. Furthermore, this is also evident in the performance improvement of both of the bzip2 compression and decompression modes. While both of the bzip2 modes had the same total number of accesses, bzip2 compression experienced a higher miss rate of 0.65% compared to decompression mode with a miss rate of 0.26%. This difference of miss rate caused bzip2 compression to have a higher performance improvement of 3.85% compared to bzip2 decompression's 1.55% performance improvement.

However, as table 6 shows, MRAM has the slowest Read accesses compared to SRAM and PCRAM. It's Write latency is relatively high as well. A higher number of misses means a higher access latency for MRAM. However, when the instructions are moved to the NV-SP, a lower number of I-cache misses are occurring. This means that accesses that were causing I-cache misses before the incorporation of MRAM NV-SP has become regular Read accesses after incorporating to the NV-SP, thus causing lower latency leading to improved performance. Although MRAM has a relatively high Write latency compared to SRAM, MRAM still shows a performance improvement since it has low Write operations with a Write latency of 10.124*ns* which is far lower than the

I-cache miss latency of 26*ns*.

Figure 17 shows that the size of the MRAM NV-SP is irrelevant to the performance improvement of the system. This is largely due to the convenient Write latency of MRAM. While MRAM Read latency is high compared to SRAM and PCRAM, it is still low enough to have little to no impact on the overall performance of the system. Furthermore, since Write operations are equal to the number of instructions stored in the NV-SP, the total Write access latency of the MRAM, similar to MRAM total Read latency, is still too low to have a major impact on the overall performance. By adding the instructions with the highest number of misses in the NV-SP all the miss latency caused by these instructions is mitigated.

**PCRAM Performance**

Figure 18 shows the execution time of all the applications when using PCRAM technology in the NV-SP implementation. Unlike figure 17, figure 18 suggests a correlation between the performance improvement/degradation and the size of the NV-SP as well as the correlation between performance degradation and the low number of misses and low miss rate. Similar to MRAM NV-SP, applications with high miss rate generally show higher performance improvement after incorporating the PCRAM NV-SP. Unlike with MRAM NV-SP, however, some applications with lower miss rate perform worse after incorporating the PCRAM NV-SP at 2 kB and 4 kB.

Applications with high miss rate such as specrand show performance improvement of 23% for all PCRAM NV-SP sizes. Similarly, AES had a performance improvement of 13% for 4kB PCRAM NV-SP and 15.9% and 17.12% for 1 kB and 2 kB respectively, showing that the size of the PCRAM NV-SP has an effect on the performance as well. Furthermore, applications with lower number of misses such as ARCFOUR experienced performance degradation of 0.38%, 3.44%, and 10.19% with 1 kB, 2 kB and 4 kB respectively. Similarly ROT13 showed performance degradation of 0.42%, 3.95%, and 12.18% with 1 kB, 2 kB and 4 kB respectively.

Non-Cryptographic applications showed performance improvement with none experiencing any performance degradation. This is largely due to the fact that all these applications have a high number of misses. While JPEG Decoder showed higher percentages of performance improvement with lower PCRAM NV-SP sizes, the rest of the Non-Cryptographic

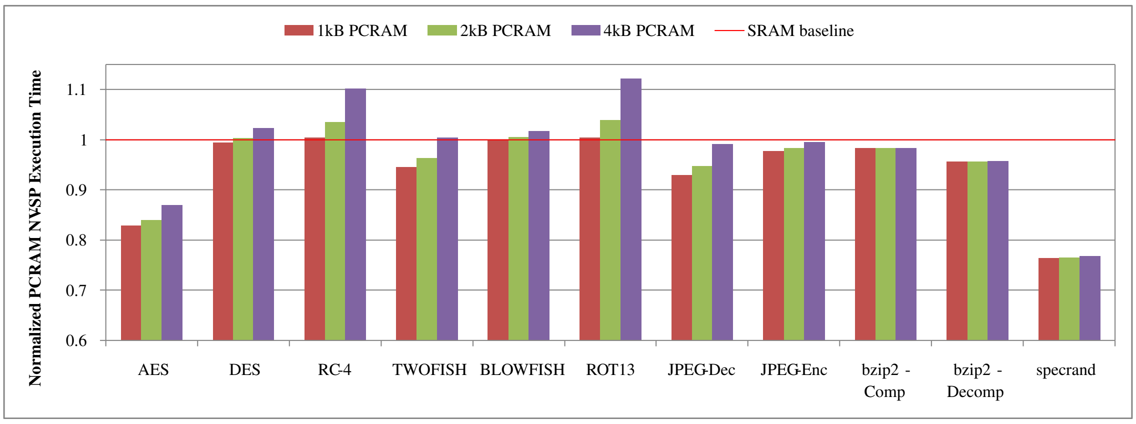


Figure 18. Performance evaluation when incorporating the PCRAM NV-SP.

applications had similar performance improvement for all the NV-SP sizes.

Although PCRAM has a very high Write latency, the majority of accesses to the PCRAM NV-SP are Read accesses which mitigates the high Write latency. Applications with a higher number of misses show performance improvement because of PCRAM's low Read latency, which, in turn, has more impact on the performance than the latency caused by the Write operations. In PCRAM, a higher number of misses leads to a low access latency. When the number of misses is low compared to the overall number of accesses – or has a low miss rate – the impact of the Write access latency increases leading to performance degradation. This is observable in the performance of ARCFOUR and ROT13.

While DES has a low miss rate, it achieved performance improvement of 0.55% at 1 kB. This improvement is insignificant compared to other applications with high miss rates and a high number of misses, but it highlights the correlation of performance improvement/degradation with the PCRAM NV-SP size. While DES had performance improvement at 1 kB, it showed an increasing performance degradation of 0.34% and 2.27% at 2 kB and 4 kB respectively. This trend of performance degradation increase can be observed for Towfish and Blowfish as well. These applications showed performance improvement at 1 kB, then experienced performance degradation at larger sizes. A similar trend of decreasing performance gains as the size of the PCRAM NV-SP increases can also be seen for AES, JPEG decode, and JPEG encode. While these applications maintained performance improvement at all sizes, the gains are clearly higher for 1 kB compared to 4 kB when using PCRAM NV-SP. The reasoning behind this behavior is the high latency of Write operations. If the size of the NV-SP increases, more instructions are written to the PCRAM NV-SP, thus increasing the impact high Write latency has on the overall latency of the PCRAM NV-SP.

### 6.2.2 NV-SP Access Energy evaluation

#### MRAM Access Energy

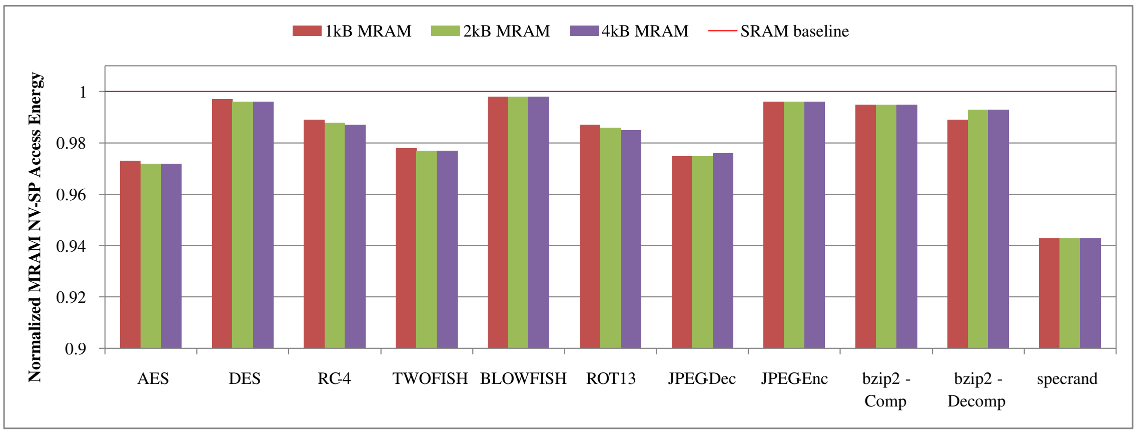


Figure 19. Access Energy evaluation when incorporating the MRAM NV-SP.

Figure 19 shows the total access energy of all the applications when using MRAM technology in the NV-SP implementation. While Figure 19 shows that using the MRAM NV-SP implementation did not improve the total access energy, a slight improvement can be noted for specrand and AES. This shows a correlation between the high miss rates of specrand and AES and slightly lower access energy. In this case, MRAM NV-SP has a much lower number of accesses compared to I-cache. Since 4 accesses per instruction, as shown in equation 1, are needed for every miss, only 2 Read accesses needed for the same instruction when it is stored in the MRAM NV-SP, meaning that an instruction in the NV-SP should require as half as accesses energy as a miss in the I-cache. Nonetheless, this improvement on the number of accesses is overshadowed by the relatively high access energy of the MRAM compared to SRAM as shown in table 7. Furthermore, the access energy of the MRAM NV-SP has very little effect on the total access energy since the majority of accesses are still happening in the I-cache. This is why the total access energy did not change dramatically.

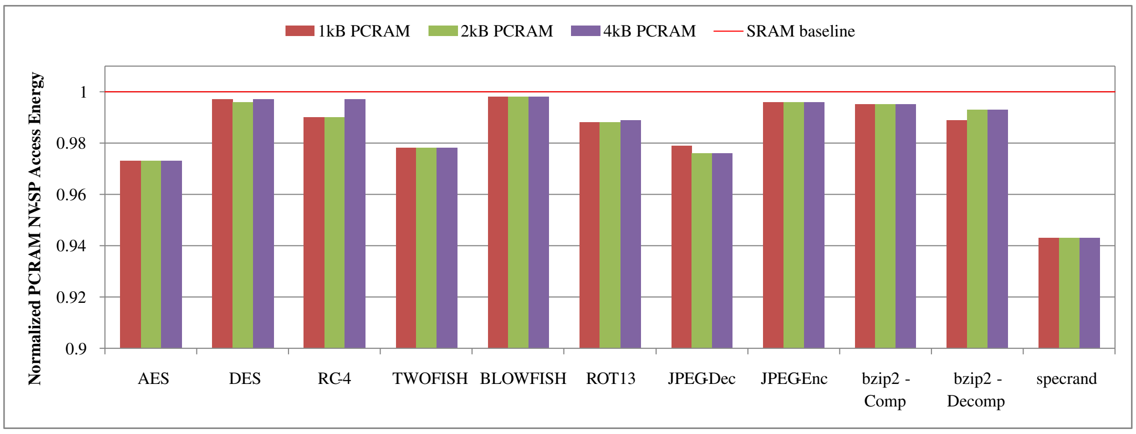


Figure 20. Access Energy evaluation when incorporating the PCRAM NV-SP.

### PCRAM Access Energy

Figure 20 shows the total access energy of all the applications when using PCRAM technology in the NV-SP implementation. Although PCRAM has a very low Read access energy similar to SRAM as shown in table 7, it has not improved the total access energy after incorporating the PCRAM NV-SP into the system. This can be attributed to the overshadowing of the very high Write access energy of PCRAM. A slight decrease in total access energy can be observed from both AES and specrand since they require much fewer accesses for the same instructions stored in the PCRAM NV-SP compared to when those instructions are stored in the I-cache. This is largely due to the much higher number of accesses into the I-cache. Figure 20 shows that the total access energy did not improve very much when using PCRAM NV-SP.

## 6.2.3 NV-SP Power Leakage evaluation

### MRAM Power Leakage

Introducing the MRAM NV-SP to the system has reduced the power leakage measurements for all the applications. As shown in table 7, the leakage power of MRAM is much lower than PCRAM and SRAM power leakage. Figure 21 shows the combined power leakage measurements for I-cache and MRAM NV-SP. It shows that incorporating the MRAM NV-SP has reduced the power leakage of the I-cache for all applications.

Since MRAM has low power leakage and because the MRAM is only operating when it is accessed, the total power leakage of the MRAM is very small. The effect the MRAM

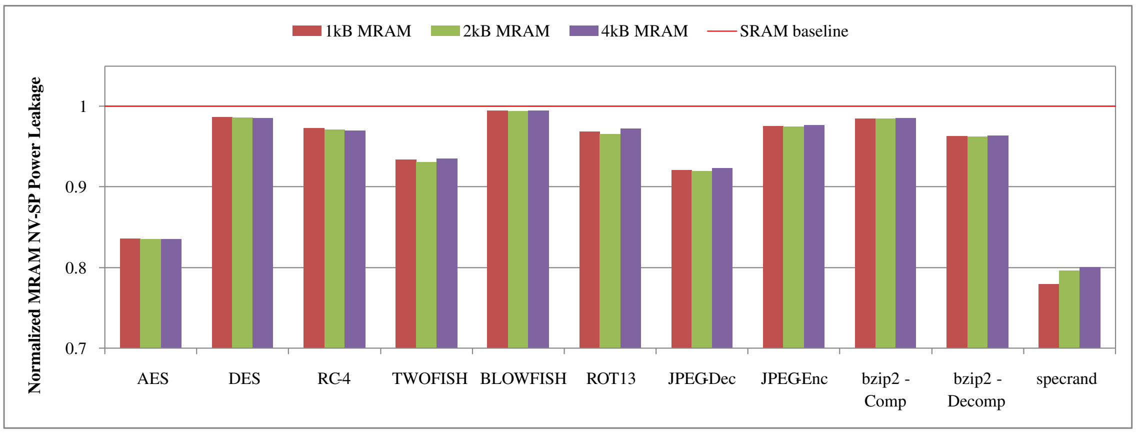


Figure 21. The combined power leakage of I-cache and MRAM NV-SP.

power leakage on the total combined power leakage of the NV-SP and I-cache is also very small. Nonetheless, the majority of power leakage reduction is happening in the I-cache. Since the I-cache is constantly operating, any reduction in execution time – or performance improvement – reduced the time in which the I-cache is operating. This means lower power leakage from the I-cache. Since MRAM has improved the performance of all the applications, as shown in figure 17, the time in which the I-cache is operating is reduced, thus decreasing its power leakage for all the applications.

## PCRAM Power Leakage

Introducing the PCRAM NV-SP to the system has reduced the power leakage measurements for some applications while greatly increased it for other applications. As shown in table 7, the leakage power of PCRAM is much higher than MRAM and is high compared to SRAM. Figure 22 shows the combined power leakage measurements for I-cache and PCRAM NV-SP. It shows that incorporating the PCRAM NV-SP has reduced power leakage for applications with high miss rate such as AES and specrand, while drastically increasing it for applications with miss rate such as RC-4 and ROT13.

This draws a relationship between the power leakage of the I-cache and performance improvement when introducing the PCRAM NV-SP. This relationship is seen clearly for all the Cryptographic applications. AES has maintained a performance improvement for all sizes of the PCRAM NV-SP, which caused power leakage reduction for all sizes. Furthermore, Twofish had performance improvement at sizes 1 kB and 2 kB causing the power leakage at those sized to be reduced. However, Twofish experienced performance degradation at 4 kB, which in turns increased the power leakage at that size.

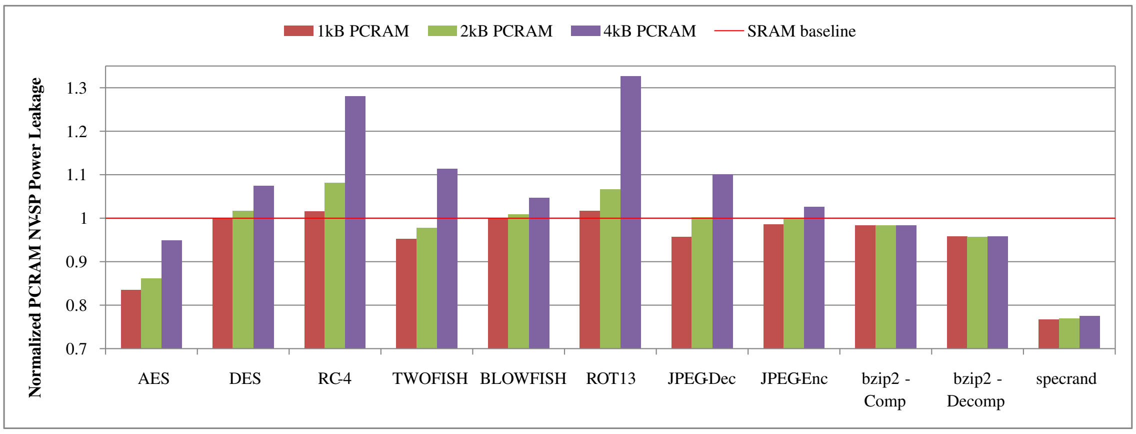


Figure 22. The combined power leakage of I-cache and PCRAM NV-SP.

Although PCRAM NV-SP improved performance for JPEG-Decode and JPEG-Encode, both of the JPEG mode had a clear power leakage increase when using the PCRAM NV-SP. The reason for this is the high power leakage of the PCRAM at larger sizes. In table 7, the power leakage of PCRAM surpasses that of SRAM. Even though PCRAM is only operated when it is accessed, it still causes a significant increase in the power leakage for the JPEG modes.

In the case of JPEG-Decode, it experiences a relatively high miss rate. When instructions were stored in the NV-SP, accesses to it increased resulting in more "on" time for the NV-SP. When using a PCRAM NV-SP, those high number of accesses cause higher power leakage. In fact, in the case of JPEG-Decode, the PCRAM NV-SP contributed 10% of the total combined power leakage of the I-cache and NV-SP. This shows that a high number of accesses with low performance improvement when using the NV-SP can greatly increase the power leakage of the NV-SP. The same reasoning can be attributed to the power leakage increase for JPEG-Decode. However, in the of JPEG-Decode, the major factor was the low performance improvement of 0.44 with 4 kB combined with the high power leakage of the PCRAM at that size is what caused the increase of the power leakage to be above the SRAM baseline.

# 7 Summary

The work of this thesis focused on evaluating the performance, energy, and power leakage of a Non-Volatile Scratch Pad that is incorporated into the low-level memory hierarchy. More specifically, the NV-SP is used to store instructions that cause the highest number of misses to mitigate the miss latency of the L1 I-cache causes by those misses. Such implementation gives insight into the possible advantages that emerging NVM technologies can present. A framework was developed and implemented using different tools and technologies to evaluate the impact of incorporating the NV-SP into the memory hierarchy. Applications based on cryptographic algorithms and other non-cryptographic algorithms.

In this thesis, it is shown that using NVM technologies such as PCRAM and MRAM in the form of an NV-SP can greatly impact the performance of the I-cache. For instance, using MRAM NV-SP reduced the latency of caused by the I-cache and improved performance up to 22.35% and 16.49% for the applications specrand and AES respectively. In fact, MRAM has improved the performance of the system for all the applications used in the experiment. Although the improvement in performance for those applications was at different degrees, it was established that due to MRAM's low read latency and the high miss latency of the I-cache. This points out that the penalty of using the relatively slower MRAM, compared to SRAM, is far lower than the penalty of miss latency in the I-cache. This performance gain was observed for all sized of the NV-SP.

In comparison, PCRAM was shown to degrade performance for applications with low miss rate, especially at larger NV-SP sizes. Unlike MRAM, PCRAM features very high write latency. This means that the penalty of writing instructions to larger sizes of NV-SP is greater than the penalty of miss latency in the I-cache. Applications such as ROT13 and ARCFOUR experienced up to 12.18% and 10.19% performance degradation respectively. Furthermore, some applications with higher miss rates experience performance improvement at smaller sizes while experiencing performance degradation at larger sizes. Nonetheless, applications with high miss latency showed performance improvement relative to their miss rates – higher application miss rates for I-cache meant better performance when using the NV-SP. Applications such as AES and specrand, where both feature the highest miss rates among all the applications, had experienced up to 23% and 17.12% performance improvement when using PCRAM NV-SP.

This thesis also showed that using MRAM NV-SP or PCRAM NV-SP helped reduce the total access energy of the I-cache, however, this reduction of access energy is not very high. Since the NV-SP stores instructions and never evicts them, the majority of accesses to the NV-SP are Read accesses. The number of Read accesses to the NV-SP is the same as the total number of misses caused by the instructions stored in the NV-SP. Since, generally, the number of misses caused by the instructions stored in the NV-SP is much lower than the total number of accesses to the I-cache, even after the incorporation of the NV-SP, the total access energy of the I-cache did not improve very much. In other words, the I-cache still counts for most of the total access energy after the incorporation of the NV-SP, since the NV-SP access energy has a smaller number of accesses compared to that of the I-cache. This is true for both MRAM and PCRAM.

Since NV-SP is non-volatile, this allows it to retain information while turned off, thus it is only on when accessed. Because of this non-volatile nature, the power leakage of the NV-SP is very little effect on the total power leakage of the I-cache. But, since the I-cache is on for the whole duration of the execution, any improvement in performance means an improvement on the I-cache power leakage. Furthermore, this thesis showed that MRAN NV-SP has reduced the total power leakage of the I-cache. This is largely due to the fact that MRAM NV-SP improved the performance of the I-cache for all the applications. This is not true for all the applications when using the PCRAM NV-SP. For some applications such as ROT13 and ARCFOUR, the PCRAM NV-SP degraded performance, which in turn caused the I-cache to be on for longer time resulting in a much higher power leakage compared to before incorporating the PCRAM NV-SP. In fact, applications such as JPEG-Dec and JPEG-Enc, although saw improvement in performance when using PCRAM NV-SP, it still showed higher power leakage due to the high power leakage of PCRAM at larger sizes.

Furthermore, this thesis also presented an NVM-based implementation of the AES S-Box as a demonstration of the potential application of NVM technologies in the field of Computation in Memory. By using memristor-based *AND* and *OR* logic gates, an $XOR$ gate can be created to compute and store AES S-box equivalents of data in the NVM devices. Such AES S-box implementations can be used to encrypt and decrypt information while it is in the cache with very little data management and low energy cost.

## References

[1] E. Pop, "Energy dissipation and transport in nanoscale devices", *Nano Research*, vol. 3, no. 3, pp. 147–169, 2010, ISSN: 19980124. DOI: `10.1007/s12274-010-1019-z`. arXiv: `1003.4058`.

[2] A. Ceyhan, S. Panth, Moongon Jung, A. Naeemi, and Sung Kyu Lim, "Evaluating chip-level impact of cu/low-k performance degradation on circuit performance at future technology nodes", *IEEE Transactions on Electron Devices*, vol. 62, no. 3, pp. 940–946, 2015, ISSN: 0018-9383. DOI: `10.1109/ted.2015.2394407`.

[3] T. Endoh, H. Koike, S. Ikeda, T. Hanyu, and H. Ohno, "An Overview of Non-volatile Emerging Memories-Spintronics for Working Memories", *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 6, no. 2, pp. 109–119, 2016, ISSN: 21563357. DOI: `10.1109/JETCAS.2016.2547704`.

[4] E. Karl, Y. Wang, Y. G. Ng, Z. Guo, F. Hamzaoglu, U. Bhattacharya, K. Zhang, K. Mistry, and M. Bohr, "A 4.6GHz 162Mb SRAM design in 22nm tri-gate CMOS technology with integrated active VMIN-enhancing assist circuitry", *Digest of Technical Papers - IEEE International Solid-State Circuits Conference*, vol. 55, pp. 230–231, 2012, ISSN: 01936530. DOI: `10.1109/ISSCC.2012.6176988`.

[5] N. Srinivasan, Shalakha D., Sivaranjani D., B. B. T. Sundari, S. Sri Lakshmi G., and N. S. Prakash, "Power Reduction by Clock Gating Technique", *Procedia Technology*, vol. 21, pp. 631–635, 2015, ISSN: 22120173. DOI: `10.1016/j.protcy.2015.10.075`. [Online]. Available: `http://dx.doi.org/10.1016/j.protcy.2015.10.075`.

[6] N. P. Jouppi and S. J. E. Wilton, "Tradeoffs in two-level on-chip caching", *SIGARCH Comput. Archit. News*, vol. 22, no. 2, pp. 34–45, Apr. 1994, ISSN: 0163-5964. DOI: `10.1145/192007.192015`. [Online]. Available: `http://doi.acm.org/10.1145/192007.192015`.

[7] L. Chua, "Memristor - The Missing Circuit Element", *IEEE Transactions on Circuit Theory*, vol. 18, no. 5, pp. 507–519, 1971.

[8] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found", *Nature*, vol. 453, no. 7191, pp. 80–83, 2008, ISSN: 00280836. DOI: `10.1038/nature06932`.

[9] J.-G. ( Zhu and C. Park, "Magnetic Tunnel Junctions", vol. 9, no. 11, pp. 36–45, 2006.

[10] A. V. Khvalkovskiy, D. Apalkov, S. Watts, R. Chepulskii, R. S. Beach, A. Ong, X. Tang, A. Driskill-Smith, W. H. Butler, P. B. Visscher, D. Lottis, E. Chen, V. Nikitin, and M. Krounbi, "Erratum: Basic principles of STT-MRAM cell operation in memory arrays (Journal of Physics D: Applied Physics (2013) 46 (074001))", *Journal of Physics D: Applied Physics*, vol. 46, no. 13, 2013, ISSN: 00223727. DOI: `10.1088/0022-3727/46/13/139601`.

[11] S. Senni, A. Gamatie, G. Sassatelli, R. M. Brum, B. Mussard, and L. Torres, "Potential Applications Based on NVM Emerging Technologies", *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1012–1017, 2015. DOI: `10.7873/date.2015.1120`.

[12] S. Senni, T. Delobelle, O. Coi, P. Y. Peneau, L. Torres, A. Gamatie, P. Benoit, and G. Sassatelli, "Embedded systems to high performance computing using STT-MRAM", *Proceedings of the 2017 Design, Automation and Test in Europe, DATE 2017*, pp. 536–541, 2017. DOI: `10.23919/DATE.2017.7927046`.

[13] R. Dave, K. Smith, M. DeHerrera, J. Slaughter, B. Engel, J. Sun, J. Janesky, B. Butcher, G. Grynkewich, J. Akerman, S. Pietambaram, M. Durlam, S. Tehrani, and N. Rizzo, "A 4-Mb toggle MRAM based on a novel bit and switching method", *IEEE Transactions on Magnetics*, vol. 41, no. 1, pp. 132–136, 2005, ISSN: 0018-9464. DOI: `10.1109/tmag.2004.840847`.

[14] S. Bandiera and B. Dieny, "Thermally assisted MRAM", *Handbook of Spintronics*, pp. 1065–1100, 2015. DOI: `10.1007/978-94-007-6892-5_40`.

[15] X. Wang, H. Deng, W. Feng, Y. Yang, and K. Chen, "Memristor-based XOR gate for full adder", *Chinese Control Conference, CCC*, vol. 2016-Augus, no. 1, pp. 5847–5851, 2016, ISSN: 21612927. DOI: `10.1109/ChiCC.2016.7554272`.

[16] A. Kolodny, S. Kvatinsky, U. C. Weiser, E. G. Friedman, G. Satat, and N. Wald, "MRL- Memristor Ratioed Logic", *2012 13th International Workshop on Cellular Nanoscale Networks and their Applications*, pp. 1–6, 2012. DOI: `10.1109/cnna.2012.6331426`.

[17] S. Kvatinsky, S. Member, D. Belousov, S. Liman, G. Satat, S. Member, N. Wald, E. G. Friedman, A. Kolodny, S. Member, and U. C. Weiser, "MAGIC — Memristor-Aided Logic", *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, no. 11, pp. 895–899, 2014. DOI: `10.1109/TCSII.2014.2357292`.

[18] S. Kvatinsky, S. Member, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, S. Member, and U. C. Weiser, "Memristor-Based Material Implication ( IMPLY ) Logic : Design Principles and Methodologies", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 10, pp. 2054–2066, 2014. DOI: `10.1109/TVLSI.2013.2282132`.

[19] L. Guckert, S. Member, E. E. Swartzlander, and L. Fellow, "MAD Gates — Memristor Logic Design Using Driver Circuitry", *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 64, no. 2, pp. 171–175, 2017. DOI: `10.1109/TCSII.2016.2551554`.

[20] M. Hu, C. E. Graves, C. Li, Y. Li, N. Ge, E. Montgomery, N. Davila, H. Jiang, R. S. Williams, J. J. Yang, Q. Xia, and J. P. Strachan, "Memristor-Based Analog Computation and Neural Network Classification with a Dot Product Engine", vol. 1705914, pp. 1–10, 2018. DOI: `10.1002/adma.201705914`.

[21] C. Li, Y. Li, H. Jiang, W. Song, P. Lin, Z. Wang, J. J. Yang, Q. Xia, M. Hu, E. Montgomery, J. Zhang, N. Dávila, C. E. Graves, Z. Li, J. P. Strachan, and R. S. Williams, "Large Memristor Crossbars for Analog Computing", pp. 2–5, 2018. DOI: `10.1109/ISCAS.2018.8351877`.

[22] H. Chiueh, J. Draper, and J. Choma, "A dynamic thermal management circuit for system-on-chip designs", *Analog Integrated Circuits and Signal Processing*, vol. 36, no. 1-2, pp. 175–181, 2003, ISSN: 09251030. DOI: `10.1023/A:1024430504653`.

[23] D. Etiemble, "45-year CPU evolution: one law and two equations", no. 1, pp. 1–6, 2018. arXiv: `1803.00254`. [Online]. Available: `http://arxiv.org/abs/1803.00254`.

[24] X. Dong, C. Xu, N. Jouppi, and Y. Xie, "NVSim: A circuit-level performance, energy, and area model for emerging non-volatile memory", *Emerging Memory Technologies: Design, Architecture, and Applications*, vol. 9781441995, no. 7, pp. 15–50, 2014, ISSN: 0278-0070. DOI: `10.1007/978-1-4419-9551-3_2`. arXiv: `arXiv:1011.1669v3`.

[25] X. Yao, J. Harms, A. Lyle, F. Ebrahimi, Y. Zhang, and J. P. Wang, "Magnetic tunnel junction-based spintronic logic units operated by spin transfer torque", *IEEE Transactions on Nanotechnology*, vol. 11, no. 1, pp. 120–126, 2012, ISSN: 1536125X. DOI: `10.1109/TNANO.2011.2158848`.

[26] T. Yagi, A. V. Kolobov, T. Fukaya, J. Tominaga, P. Fons, M. Krbal, and R. E. Simpson, "Interfacial phase-change memory", *Nature Nanotechnology*, vol. 6, no. 8, pp. 501–505, 2011, ISSN: 1748-3387. DOI: `10.1038/nnano.2011.96`.

[27] H. F. Hamann, M. O'Boyle, Y. C. Martin, M. Rooks, and H. K. Wickramasinghe, "Ultra-high-density phase-change storage and memory", *Nature Materials*, vol. 5, no. 5, pp. 383–387, 2006, ISSN: 14764660. DOI: `10.1038/nmat1627`.

[28] E. Eleftheriou. (2017). Neuromorphic computing based onphase-change-memory devices, [Online]. Available: `https://bigdata.uni.lu/wp-content/uploads/sites/5/2017/10/Evangelos-Eleftheriou_IBM-Research-Zurich_Presentation-Big-Data-Conference-2017.pdf` (visited on 03/28/2019).

[29] G. W. Burr, M. J. BrightSky, A. Sebastian, H. Y. Cheng, J. Y. Wu, S. Kim, N. E. Sosa, N. Papandreou, H. L. Lung, H. Pozidis, E. Eleftheriou, and C. H. Lam, "Recent Progress in Phase-Change Memory Technology", *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 6, no. 2, pp. 146–162, 2016, ISSN: 21563357. DOI: `10.1109/JETCAS.2016.2547718`.

[30] S. R. Nandakumar, I. Boybat, M. Le Gallo, A. Sebastian, E. Eleftheriou, and B. Rajendran, "A phase-change memory model for neuromorphic computing", *Journal of Applied Physics*, vol. 124, no. 15, p. 152 135, 2018, ISSN: 0021-8979. DOI: `10.1063/1.5042408`.

[31] S. Sardashti, K. Sewell, S. K. Reinhardt, A. Basu, D. A. Wood, T. Krishna, J. Hestness, G. Black, B. Beckmann, N. Vaish, D. R. Hower, M. D. Hill, N. Binkert, M. Shoaib, A. Saidi, and R. Sen, "The gem5 simulator", *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, p. 1, 2011, ISSN: 01635964. DOI: `10.1145/2024716.2024718`.

[32] J. Dennis, "The Memristor: Introduction to NanoElectronics", *Rowan University*, pp. 3–14, 2009.

[33] J. Daemen and V. Rijmen, *The Design of Rijndael*. 2001, p. 253.

[34] C. S. R. CENTER. (2001). Announcing approval of federal information processing standard (fips) 197, advanced encryption standard (aes), [Online]. Available: `https://csrc.nist.gov/news/2001/announcing-approval-of-fips-197-aes` (visited on 04/04/2019).

[35] K. Nyberg, *Perfect nonlinear S-boxes*, D. W. Davies, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, pp. 378–386, ISBN: 978-3-540-46416-7.

[36] D. Biolek and V. Biolková, "[09] SPICE Model of Memristor with Nonlinear Dopant Drift (2009).pdf", no. 1, pp. 210–214,

[37] Y. V. Pershin and M. D. Ventra, "Spice model of memristive devices with threshold", 2013.

[38] H. Krad and A. Y. Al-Taie, "A New Trend for CISC and RISC Architectures", *Asian Journal of Information Technology*, vol. 6, no. 11, pp. 1125–1131, 2007.

[39] (2019). X86 cpus' guide, [Online]. Available: `http://www.x86-guide.com/` (visited on 04/03/2019).

[40] B. Conte. (2015). Crypto-algorithms, [Online]. Available: `https://github.com/B-Con/crypto-algorithms` (visited on 11/29/2018).

[41] W. Tuchman, "Internet besieged", D. E. Denning and P. J. Denning, Eds., pp. 275–280, 1998. [Online]. Available: http://dl.acm.org/citation.cfm?id=275737.275754.

[42] R. L. Rivest and J. C. N. Schuldt, "Spritz — a spongy RC4-like stream cipher and hash function", 2014.

[43] J. Nechvatal, E. Barker, D. Dodson, M. Dworkin, J. Foti, and E. Roback, "Status Report on the First Round of the Development of the Advanced Encryption Standard", vol. 104, no. 5, pp. 435–459, 1999.

[44] B. Schneier. (1994). Description of a new variable-length key, 64-bit block cipher (blowfish), [Online]. Available: https://www.schneier.com/academic/archives/1994/09/description_of_a_new.html (visited on 04/04/2019).

[45] G. K. Wallace, "The JPEG Still Picture Compression Standard 2 Background : Requirements and Selec-", *IEEE Transactions on Consumer Electronics*, pp. 1–17, 1991.

[46] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, "Mediabench: A tool for evaluating and synthesizing multimedia and communicatons systems", MICRO 30, pp. 330–335, 1997. [Online]. Available: http://dl.acm.org/citation.cfm?id=266800.266832.

[47] M. Consortium. (1997). Mediabench, [Online]. Available: http://mathstat.slu.edu/~fritts/mediabench/ (visited on 04/04/2019).

[48] (2010). Ncbi c++ toolkit cross reference - readme for bzip2/libzip2, [Online]. Available: https://web.archive.org/web/19980704181204/ (visited on 04/05/2019).

[49] J. Seward, "Bzip2 and libbzip2", no. March, 2000.

[50] S. P. E. Corporation. (2003). 256.bzip2 spec cpu2000 benchmark description file, [Online]. Available: https://www.spec.org/cpu2000/CINT2000/256.bzip2/docs/256.bzip2.html (visited on 04/06/2019).

[51] I. Enthought. (2013). Bzip2-1.0.6, [Online]. Available: https://github.com/enthought/bzip2-1.0.6 (visited on 04/06/2019).

[52] S. P. E. Corporation. (2006). 999.specrand spec cpu2006 benchmark description, [Online]. Available: https://www.spec.org/cpu2006/Docs/999.specrand.html (visited on 04/06/2019).

[53] S. K. Park and K. W. Miller, "Random number generators: Good ones are hard to find", *Commun. ACM*, vol. 31, no. 10, pp. 1192–1201, Oct. 1988, ISSN: 0001-0782. DOI: 10.1145/63039.63042. [Online]. Available: http://doi.acm.org/10.1145/63039.63042.

[54] I. G. t. Free Software Foundation. (2019). Using the gnu compiler collection (gcc), [Online]. Available: https://gcc.gnu.org/onlinedocs/gcc/ (visited on 04/04/2019).

[55] A. Griffith, *GCC: The Complete Reference*, 1st ed. New York, NY, USA: McGraw-Hill, Inc., 2002, ISBN: 0072224053.

[56] J. Power. (2015). Gem5 tutorial 0.1 documentation » part i: Getting started with gem5, [Online]. Available: http://pages.cs.wisc.edu/~david/courses/cs752/Fall2015/gem5-tutorial/part1/cache_config.html (visited on 04/07/2019).

[57] J. Power. (2015). Gem5 tutorial 0.1 documentation » part i: Getting started with gem5, [Online]. Available: http://pages.cs.wisc.edu/~david/courses/cs752/Fall2015/gem5-tutorial/part1/simple_config.html (visited on 04/07/2019).