Novel Application Specific Architectures for Extreme Efficiency Under Harsh Operating Conditions

THESIS SUBMITTED TO THE FAULTY OF THE UNIVERSITY OF MINNESOTA
BY

Salonik Resch

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Ulya R. Karpuzcu

October 2022

Less than one hundred years ago, computation was limited to centralized and large scale machines that could barely fit in a single room, and could only be accessed by the scientists and engineers that worked on them. That world is hard to imagine today. Enabled by incredible technological advancement, and necessitated by ever increasing demand for its benefits, computing has become ubiquitous in the modern era. Modern life is filled with mobile and embedded computing devices. Additionally, the large, room sized computers have remained relevant, as powerful computers in data centers are accessible over the web, providing invaluable computational resources.

However, as computing has spread into ever more domains, the challenges associated in architecting these computing systems have increased. Computers no longer simply exist in stable, controlled environments, and they must be designed to handle any and all unique challenges imposed by their surroundings. The challenges involved vary wildly by location, and include operating under extreme power restriction, tolerating intermittent power supply, avoiding corruption due to radiation, and maintaining correctness across wide temperature ranges, to only name a few. Surprisingly, these challenges are not exclusive to mobile or embedded devices. Even computers in data centers are facing new challenges, as new, more advanced devices are becoming ever more sensitive to noise which was previously of no concern. Notably, computers which can exploit quantum mechanics now must deal with new forms of noise which can not be mitigated by traditional means. Hence, even computers in the middle of data centers can be considered to exist in "harsh" environments, and must adapt to the conditions they impose.

All of these additional challenges, along with increasing demands for performance, have caused a diversification of computer architectures. The "one size fits all" mindset of the microchip of the past few decades no longer universally applies. While initially these challenges seem like a burden, they have highlighted unique strengths and weaknesses of the available hardware technology and provided insight into potential new designs and computing paradigms. Hence, these new challenges are opportunities to explore alternative approaches and new architectures which may provide improved performance, energy efficiency, and robustness. These new architectures often benefit from extreme specialization, becoming exceedingly good at their niche task. Hence, these new architectures are moreso *accelerators* than traditional computers.

This thesis covers architectures which have been designed to handle the unique challenges of their environment and the analysis of how the corresponding challenges have impacted performance, efficiency, accuracy, or total lifetime of the device. We cover architectures in a wide range of environments, from at the core of data centers, to "beyond the edge" batteryless devices, and even to off-earth space applications. A common theme among all the chapters in this thesis is exploiting unique proprieties of the hardware or application in order to design a system which is particularly well suited for its environment. Prominently, we have shown that processing-in-memory architectures provide a promising alternative to traditional computing systems, and that these architectures are surprisingly robust and adaptable to harsh operating conditions. We show that architectures can be built on this technology which are resilient to power outages, robust to wide temperature ranges, and nearly immune to soft errors from radiation, all while providing high performance and energy efficiency. We also explore the limitations of these architectures, such as their finite endurance and limitations of their electrical properties, and provide insight on how these limitations can be mitigated. This thesis also covers accelerators which exploit the power of quantum mechanics, which can potentially solve problems that were previously believed to be impossible. These accelerators are highly susceptible to noise, and are fragile even in the most secure environments it is possible to construct. This is currently the limiting factor and it renders them impractical for useful applications. While noise in traditional computing systems is a highly manageable and well-studied problem, these new architectures face a whole new class of noise which much harder to overcome. We analyze the impacts of this noise, and we propose strategies which can be used to mitigate it in the near term and potentially over come it in the long term.

Acknowledgements

I would like to thank my academic advisor Ulya R. Karpuzcu for all the guidance and support she has provided me over over the years of my undergraduate and graduate career. As an undergraduate I was excited and interested in research, but did not know how to go about pursuing the career that I wanted. She gave me my first major break and opened the door to a world of opportunity. I appreciate her willingness to let me explore a wide range of topics and her encouragement to pursue projects I was interested in. I must specifically thank her for tolerating all of my bad ideas and redirecting my enthusiasm in more successful directions. Because of her, I have enjoyed (almost) everything I worked on. Looking back, I have come realize just how incredibly fortunate I was. A large part of me wishes this phase of my life would never end.

I would also like to thank all of the mentors and colleagues I've had here at the university. Along with my advisor, Professor Jian-Ping Wang and Professor Sachin Sapatnekar have led team CRAM, creating an incredible collaborative environment which has created numerous opportunities for myself and others. Much of the work in this thesis would not have been remotely possible without their help. It was a major honor being a small part of team CRAM. I would also like to thank Professor Moinuddin Quereshi and Professor Pen-Chung Yew for their valuable feedback, guidance, and collaborative opportunities. I am humbled by their wisdom and kindness and I am motivated to not only to be a better researcher but a better person.

I was also very lucky to have an incredible group of friends and colleagues during my time at the University. My senior colleagues Ismail Akturk and Karen Khatamifard created a welcoming environment when I first joined the lab. I specifically remember how calm Karen was under pressure, a characteristic I was very jealous of at the time and have tried to emulate since. Zamshed Chowdury is my fellow labmate and team CRAM member. You could not ask to meet a kinder person, and I was fortunate to spend most of my years here with him. I asked Zhengyang Zhao inumerable questions about MTJs, and his hard work is was an invaluable contribution to many of my projects. Masoud Zabihi was not only a great colleague who contributed greatly to my work but a great friend who I was incredibly lucky to have and one of the funniest people I ever met. I thank him for all the time spent together in and out of school, and for being a person you can rely on in tough times. I also want to thank Hari Cherupalli for all our long conversations, which happened at the times I needed them most. Finally, there's Husrev Cilasun. Despite being a dangerous threat to my reputation by getting a paper out in his first year, he's an incredibly nice guy. I was impressed by how fast he learned and I am grateful for key contributions he made during tough review cycles. I greatly appreciate the help and collaborations I had with Swamit Tannu and Yipeng Huang who taught me a lot about quantum computing. They not only very smart guys, but also kind and helpful.

I would like to thank my parents, Walter and Annette Resch, who have encouraged and supported me all of my life. This journey I am on would not have even begun without them.

Finally, I thank 小航 for her love and support, which means everything to me.

Dedication

To 小航, my everything

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Computational power has increased dramatically over the previous decades. However, the demand for computational power has grown even faster. Additionally, the number of computing domains has exploded as ever more potential uses of computation have been discovered. Today, no single technology or architecture is capable of adequately supplying the computational resources for every application. Each domain has its own requirements and unique challenges, which requires computing systems to be thoughtfully and specifically tailored in order to satisfy all constraints and be an effective solution to a given problem.

The increasing energy efficiency of computing systems has enabled them to move off the power grid and rely solely on batteries. This has enabled mobile and embedded applications, which operate on “edge” devices - those which are on the periphery of the power grid. However, operating on batteries has its own limitations. Batteries must be re-charged or replaced, requiring periodic human intervention. This is not suitable for devices which operate in remote or difficult to access locations. Hence, the concept of “beyond-edge” computing has been introduced. Beyond-edge refers to devices which do not even use a battery, and obtain energy exclusively from the environment by performing energy harvesting. This includes collecting energy from sunlight, heat, or ambient RF radiation. This enables devices to operate almost anywhere conceivable, operating autonomously for long periods of time. However, this introduces significant new challenges. Harvesting energy is not reliable, and such devices will need to frequently power off when there is not sufficient power available. As traditional computers are designed assuming a continuous power supply, re-starting a computer unexpectedly will likely corrupt the architectural state and cause undefined behavior. Hence, the architecture of beyond edge devices must be modified to efficiently and correctly handle a power outage. An obvious solution is to perform checkpointing, where the architectural state is repeatedly saved to non-volatile memory, allowing it to be restored on restart. Unfortunately, this introduces many additional write operations and significant energy overhead. As power is already extremely limited on such devices, energy costly checkpointing is not a viable solution.

The situation only gets worse when we consider that these devices are much more likely to be subjected to harsh conditions than their embedded and mobile counter parts. For example, beyond edge devices show great potential in the use of high earth atmosphere or satellite applications. In such an environment, they will be exposed not only to extremely hot and extremely cold temperatures, but increased levels of radiation. Wide temperature ranges and radiation create further threats to correctness, and providing hardware resilience to both imposes additional energy efficiency overheads. Designing devices which tolerate frequent power outages, wide temperature ranges, and high radiation while also maintaining good performance and extreme energy efficiency is a considerable challenge. Traditional architectures will be hard pressed to meet all of these criteria.

Beyond edge devices also introduce security risks. They are intended for deployment in a wide variety of locations, some of which may not be secure. If an application operates on sensitive data, encryption will be required. However, computing with encrypted data introduces extreme energy overheads, making it highly impractical. Hence, security beyond the edge remains an open problem.

Computers have not only expanded into new domains via improvements in their energy efficiency, they have also expanded by being reinvented at the most fundamental level. Within the last 30 years it has been realized that an entirely new class of computers can exist. By harnessing the power of quantum mechanics, computers can perform calculations that would otherwise be impossible. Such machines are referred to as *quantum computers*. However, the name is misleading as they are not intended for general purpose computation and are only useful for a narrow range of applications. Hence, they would be more appropriately called *quantum accelerators*. This new quantum domain has revolutionary potential, and is the only known potential solution to important scientific and industrial applications. However, it comes with a corresponding and daunting new set of challenges.

Surprisingly, despite some significant differences, quantum computers experience problems that are highly similar to traditional computers. The development of quantum computers will benefit from the previous development of traditional computers. It is likely that many insights and techniques developed for traditional computers will be applicable to designing quantum computers. However, as quantum computers are more complicated, such insights and techniques will need to be adapted.

Quantum computers are extremely fragile by nature, and need near perfect isolation from their environment in order to function. Hence, even the most controlled environment at the center of a data center represents a harsh and extreme environment for quantum computers. Traditional computers also experience noise and solve the problem with error correcting codes. Unfortunately, applying error correction for quantum computers is not easy, and currently is beyond the capability of modern systems. If quantum computers are to be of practical use, we must discover more efficient methods of mitigating and correcting noise.

In this thesis, we demonstrate how processing-in-memory architectures are well suited to solve the problems of beyond edge devices. We design unique architectures to handle each of the specific challenges. We also perform analysis of the challenges of quantum computers, and propose methods to overcome the limitations of noise. We show how techniques such as statistical fault injection and approximate computing can be re-applied to quantum computers.

1.2 Thesis Contributions

This thesis proposes application specific architectures and methodologies which overcome the unique challenges of the target domains, i.e. accelerators in extreme environments. We exploit both unique properties of hardware devices and characteristics of specific applications to reduce overhead and increase robustness. As such, we show thatunconventional architectures can be highly effective where more traditional architectures fail. Our work spans many domains, including high-cost specialized hardware within data centers, remote beyond edge embedded devices, and even off-earth satellites.

Prior work on beyond edge devices has struggled with the high complexity and low energy efficiency of traditional computing systems, typically in the form of a microcontroller [120, 121]. Such traditional systems have been a challenge to adapt to the harsh and unpredictable environments as they are susceptible to power outages, wide temperature ranges, and radiation. Adding resilience via hardware or software comes with considerable energy overheads [406], which is not acceptable for energy-constrained devices. We utilize non-volatile processing-in-memory (NV-PIM) [62]and design an unconventional accelerator architectures which have an inherent resilience to the challenges imposed by beyond edge domains. We show how NV-PIM architectures can achieve extremely low checkpointing overhead, operate over a wide range of temperatures with little modification to the operating semantics, and can be made resilient to radiation with minor hardware improvements. Due to NV-PIMs extreme energy efficiency, we also investigate it as a potential accelerator in cryogenic applications, where low power consumption is critical.

Also in the cryogenic environment are quantum accelerators. Such machines are capable of solving problems which are believed to be impossible by any other means [331]. However, these accelerators are extremely fragile and are often rendered useless due to excessive noise coming from their environment [45, 238]. Hence, it is of critical important to accuracy characterize and mitigate the noise. Unfortunately, as of this writing there has been no successful demonstration of non-trivial useful computation performed by a quantum accelerator. Our work analyzes the nature of this quantum noise and provides a perspective for engineers in how to overcome it. We also have recognized that the large body of work on traditional computer architecture research may prove useful, if properly adapted to quantum contexts. Consequently, our work borrows methods such as statistical fault injection and approximate computing concepts from the field of computer architecture and applies it to quantum accelerators. We show that these techniques can both improve the reliability of noisy, small quantum computers in the near term and help achieve large, fault-tolerant quantum computers in the future.

The contributions of this these is as follows:

- In Chapter 2, we show that using processing-in-memory architectures with non-volatile memory devices provides simple and effective solutions to tolerating frequent power outages when operating beyond the edge. When processing in the memory, data is effectively being constantly backed up. Hence, there is never an inconsistency between the processor and the data storage. As the memory is non-volatile, this constant backup process is persistent across power outages. We demonstrate that non-volatile processing-in-memory architectures effectively perform constant checkpointing, allowing for extremely fine-grained and low overhead restoration on restart. We design a full architecture exploiting this advantage by designing the necessary support circuitry and rigorously covering all corner cases providing correctness guarantees.

- Given the extreme energy efficiency of PIM, we consider it as a supporting accelerator in cryogenic systems in Chapter 3. Cryogenic computing is gaining traction as it enables a significant increase in the performance of CMOS transistors and it creates an environment where quantum computing becomes possible. Hence, it is the environment where the most powerful traditional and quantum computing systems can exist. However, cryogenic environments require extreme energy efficiency, as consuming power will increase the temperature. PIM is a promising candidate to supply both supplemental memory and computation while introducing very little heat. We evaluate the efficacy of different PIM technologies in this domain.

- We further the develop the architecture from Chapter 2 in Chapter 4, where we make it more programmable by extending the instruction set and adapting the architecture to be more robust and suitable for use as low-earth orbit satellite. We show how a large instruction set can be made tolerant to power interruptions with simple correctness guarantees and minimal hardware. We exploit the fact that non-volatile memory is inherently resilient to radiation and design an entire architecture which can tolerate large degrees of radiation with relatively low hardware and energy efficiency overhead. We also demonstrate the in-memory logic of PIM can function properly across a wide temperature range.

- We make progress on solving the correctness problem for beyond edge devices in Chapter 5. We again adapt the accelerator designed in Chapter 2 to act as a mini-server operating beyond the edge. We find that the extreme energy efficiency of NV-PIM enables it to perform fully encrypted computing, despite the large overhead it imposes. Tailoring the algorithm to increase efficiency allows our accelerator to finish computation within a competitive time frame. Hence, it can act as a secure server which other beyond edge devices can off load computation to. We demonstrate how this system can accelerate applications under certain power constrained circumstances and make progress towards fully-encrypted computation beyond the edge.

- We analyze one of the most concerning limitations of non-volatile PIM in Chapter 6. Non-volatile devices have a limited endurance, meaning they can only be written a finite number of times before failure. When used for PIM, such devices can fail within a relatively short amount of time due to the large number of write operations required. Our analysis highlights the need for further device level research, and we show the limits of the mitigation techniques available.

- We analyze the limitations of PIM cross-bar architectures in Chapter 7. We show how fundamental electrical limitations of such architectures make the unscalable. We propose that two transistors per cell should be used, and that this represents an ideal location in the design space. Such a configuration has sufficient density for nearly all applications, and it completely eliminates the electrical limitations of cross-bar architectures, making large scale designs entirely feasible.

- Chapter 8 contains an overview and analysis of quantum noise and a characterization of how different noise types impact different quantum programs. We test noise mitigation strategies and show how and when they apply. Our analysis provides a roadmap for engineers in how to tolerate noise and provide warning against common pitfalls and fallacies.

- We apply the computer architecture strategy of statistical fault injection to quantum computing in Chapter 9. We show that such a strategy can reveal regions of varying sensitivity in quantum programs which can be exploited to improve reliability. This work shows how quantum software sensitivity can be matched to quantum hardware reliability in order to reduce the impact of noise.
- In Chapter 10 we again use statistical fault injection along with the concept of approximate computing to lower the overhead of quantum error correction and increase the performance of quantum hardware. Statistical fault injection enables us to know regions of the program which are relatively less sensitive, where errors will be less likely to affect the output. This allows us to selectively apply lower strength error correction to these regions. The lower strength error correction has less overhead, which speeds up computation. The downside is that these regions have an increased risk of error. However, since we know they are less sensitive, the impact of these errors will be minimized. We show that this trade-off leads to a net win in performance and reliability for a wide range error rates.

Chapter 2

Non-Volatile Inference for Energy Harvesting Applications

2.1 Introduction

Machine learning is desirable for low-power, edge devices as it provides the capability to solve a wide variety of problems. As a result, much research has been devoted to optimizing hardware for machine learning inference on such devices [75, 216]. Going even further, energy harvesting techniques [181] remove the need for a battery, enabling the placement of such devices into almost any conceivable environment. There are many exciting possible applications, such as low power sensor networks [241], wearable tech, or even implants [133]. Previous work has already experimentally demonstrated machine learning capability on energy harvesting devices using commercially available hardware [120].

Energy harvesting applications present numerous and unique challenges. The energy harvested from the environment may be less than what can be supplied by a battery, making energy efficiency even more critical than in mobile applications. Significantly, the process of energy harvesting also introduces the requirement for *intermittent processing*. Energy sources (such as sunlight, heat, movement) may be unreliable, and a device will have to shut down when the power source goes away. Additionally, even when available, the power source may be insufficient to run the device continually. In order to operate within the power budget, the device must acquire energy over time and consume it in bursts [55].

Intermittent processing introduces new considerations and metrics for performance [219]. Significantly, correctness has to be guaranteed over shut down and restart operations. If the state is not properly stored – a process known as checkpointing – restarting a device can lead to memory inconsistencies and incorrect operation [68]. Additionally, the efficiency of these shut down and restart operations becomes critical, as they take away precious energy from operations that enable forward progress. Also critical, it has to be ensured that forward progress can be made during phases of power-on time. If the energy required between two checkpoints is too large, the device will be unable to complete the computation. This results in a program getting stuck, which is referred to as non-termination. Thus, effective energy harvesting devices must have efficient techniques which enable correctness and forward progress, all while remaining within a modest hardware budget.

A recently proposed spintronic processing in memory (PIM) substrate, CRAM [62], is uniquely well suited for energy harvesting applications. Operations on CRAM are highly energy efficient, enabling a low power budget. Further, as it is a PIM solution, it removes the need for energy hungry data transfers between processor logic and (volatile) memories. The main advantage, however, is that progress is automatically saved after every operation. CRAM consists entirely of non-volatile devices and the results of all computation are immediately stored in permanent memory. As there are very few variables required to maintain the architectural state, these can also be saved after each operation with minimal energy cost. Effectively, checkpointing occurs after every operation.

Checkpointing after each operation is not a new idea [231], and for most systems this would generally be considered inefficient [68]. However, as CRAM is a non-volatile PIM substrate and *all* of the computation occurs within the memory array, data backup for checkpointing happens automatically, i.e., non-volatile PIM is always performing data backup. Hence, CRAM can restart a program from the very last operation with fast and efficient shut down and restart. Additionally, CRAM is always in a state that can be recovered from. The power can be cut *instantly and unexpectedly*, and it will still restart correctly. The maximum penalty is repeating the last instruction. We refer to this capability as *instant restartability.* This provides a significant advantage, as shut down and restart procedures for more conventional energy harvesting devices introduce additional latency and energy, and significant complexity.

In this chapter, we introduce MOUSE (**M**inimal **O**verhead Accelerator **U**tilizing **S**pintronic RAM for **E**nergy Harvesting Applications) which is built using CRAM [62]. While based on CRAM, MOUSE has a different cell design which reduces energy consumption during computation. For our applications, we implement support vector machines (SVM) and binary neural networks (BNN), which are widely used machine learning algorithms, especially promising in the energy harvesting domain due to their small footprint. We demonstrate how MOUSE can provide high performance and energy efficiency on such applications while also having efficient shut down and restart procedures. Additionally, we consider how another modification to the CRAM cell, the addition of a spin-hall effect (SHE) channel [401], can further increase energy efficiency – by enabling independent optimization of the read and writes, which otherwise come with conflicting requirements.

The contributions of this chapter are as follows:

- We demonstrate that logic operations performed with magnetic tunnel junctions (MTJs) are inherently idempotent.
- We utilize this property with processing-in-memory to create an energy-efficient and intermittent-safe machine-learning inference accelerator.

2.2 Spintronic PIM

Spintronic memory in the form of STT-MRAM is an emerging technology, with a few products already commercially available [1]. Due to its non-volatility, high density, speed, and endurance, STT-MRAM is being considered as a universal memory replacement [91]. STT-MRAM arrays use one magnetic tunnel junction (MTJ) and one access transistor per cell. Being based on CRAM [62], MOUSE maintains the same basic cell structure. By making light modifications to the array, CRAM is able to connect MTJs in such a way to enable logic operations to be implemented within the array. Therefore, MOUSE is capable of being used as both a standard STT-MRAM array and as a computational substrate. CRAM is unique in that the computation does not require any external logic circuits or the use of sense amplifiers, making the computation contained *entirely* within the array. In the following, we explain MTJ basics and show how they can be used in logic operations. Then we demonstrate how these operations can be performed within the array structure.

2.2.1 Magnetic Tunnel Junction (MTJ) Basics

STT-MRAM arrays are built with magnetic tunnel junctions (MTJ). The MTJ is a resistive memory device which consists of two magnetic layers (fixed layer and free layer) which are separated by an insulator. The polarity of the free layer can change but the fixed cannot. When the fixed and free layers are aligned, the MTJ is in the parallel (P) state, which has a low resistance and corresponds to logic value 0. When the layers are opposing, the MTJ is in the anti-parallel (AP) state, which has a high resistance and corresponds to logic value 1.

The state can be determined by applying a voltage across the device and sensing the amount of current that travels through it. If a sufficient amount of current is driven through the device, it will change state. *Importantly, the state it changes to depends on the direction of the current*. This is key to our ability to ensure correctness in spite of power outages. When current flows from the free layer (fixed layer) to the fixed layer (free layer), it switches the MTJ to the AP (P) state.

2.2.2 Implementing Logic Gates in Memory

Before showing how logic can be implemented in the MOUSE array, we demonstrate how CRAM performs logic gates on MTJs in principle. The configuration for a two-input logic gate is shown in Figure 6.1. The two MTJs in parallel are the inputs to the logic gate, and the MTJ in series with them is the output. The output must be preset to a known value. For example, the output is preset to 0 (low resistance) for a NAND gate. To implement a NAND gate, a voltage is applied across the two terminals, V_1 and V_2, such that current flows from the input MTJs to the output MTJ. If either of the input MTJs is 0 (low resistance) there will be sufficient current to switch the output MTJ to 1. If both input MTJs are 1 (high resistance), there will be insufficient current to change the state of the output MTJ, and it will remain at 0. Therefore, the state of the output MTJ follows the truth table for a NAND gate, it is 0 only if both inputs are 1. *Most importantly, per basic MTJ physics, current flowing in the supplied direction can only cause the output MTJ to switch to 1; it cannot cause it to switch to 0.*

Many other common gates –including universal ones– can be implemented similarly, such as AND and (N)OR. In order to implement other gates, we can change the number of inputs, the preset value of the output, or the direction of the current.

More complex operations are broken down into these basic logic operations. For example, to perform a full-add in MOUSE, we can perform 9 NAND gates sequentially and use spare MTJs to hold 7 temporary bits. Using full-adds, full-subtracts, and other primitive operations we can perform integer or fixed-point arithmetic, thus enabling us to implement our benchmarks. Naturally, the latency for each complex operation is quite high, as it must be broken down into its constituent gates which are then performed sequentially. However, as we will show in later sections, this

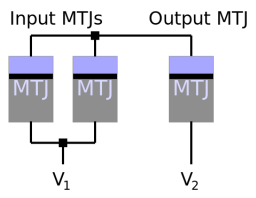


Figure 2.1: MTJs connected to implement a 2-input logic gate. The preset value of the output MTJ and the polarity and magnitude of the voltage applied between V_1 and V_2 determines the type of logic gate. The fixed layer is colored in grey and the free layer in light blue.

can be easily compensated for by performing many data independent operations in parallel, under intermittent power constraints. Due to space limitations, we will focus on MOUSE-specific CRAM adaptations next, but numerous papers[398, 401, 62] cover the details of using MTJs to perform more complex logic based on this basic CRAM principle.

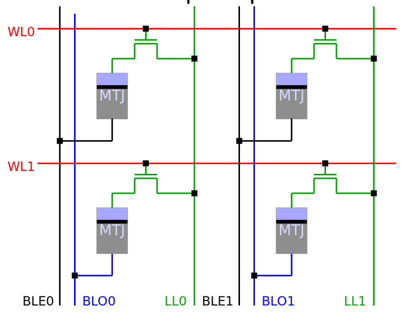


Figure 2.2: 4 cells in 2 columns and 2 rows in 1T1M (one access transistor, one MTJ) STT configuration.

2.2.3 MOUSE Array Architecture

Based on CRAM, MOUSE essentially is an STT-MRAM array with some additional hardware. As an example, four cells located in adjacent rows and columns are shown in Figure 2.2. Each memory cell consists of one MTJ and one access transistor. In each column there are two bit lines, bit line even (BLE) and bit line odd (BLO), and a logic line (LL). In each row there is a wordline (WL) that controls the access transistor. Each MTJ is connected to the LL through the access transistor and to one of the two bit lines. Cells in even rows are connected to BLE and cells in odd rows are connected to BLO. We now describe how memory and logic operations are performed in the array.

Memory Operation: To read or write from row n, activate WLn and apply a voltage differential across LL and the bitlines. Current will only travel through the bitline with the same parity as n. Current can be sensed on the bitlines to perform a read, or a large current can be driven through the MTJ to perform a write.

Logic Operation: To perform a logic operation with inputs in rows n_1, n_2 and with output in row m, preset row m by performing a write operation. n_1 and n_2 must have the same parity (i.e., both even or both odd) and m the opposite. Activate WLn_1, WLn_2 and WLm. Apply a voltage differential across BLE and BLO. Due to the parity requirement, in Figure 6.1, if V_1 is connected to BLE, V_2 must be connected to BLO, and vice versa. In this case, the junction connecting the free layers (in light blue) of the inputs and the ouput corresponds to LL. Current travels from one bit line (be it BLO or BLE, depending on the parity of the input cells), through the MTJs in rows n_1 and n_2, through the

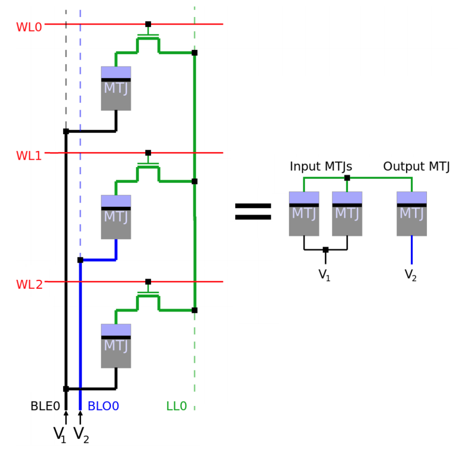


Figure 2.3: Demonstration of how a (2-input) NAND gate is performed within the array.

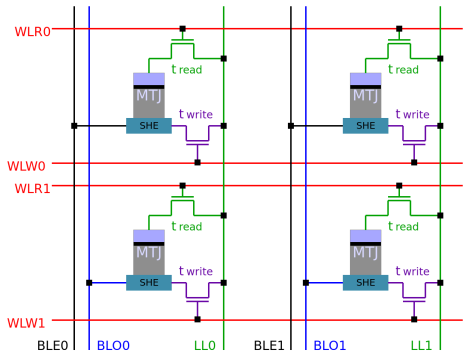


Figure 2.4: 4 cells in 2 columns and 2 rows in 2T1M SHE configuration.

LL, through the MTJ in row m, and back to the other bitline. Depending on the states of the MTJs in rows n_1 and n_2, the state of the MTJ in row m will either change or not. As an illustrative example, Figure 2.3 demonstrates the formation of a NAND gate.

Voltage which drives the operation is applied to every column (over the respective bitlines) in which the specified operation should take place. The peripheral circuitry determines which columns these are, which can be specified by dedicated instructions as will be described in Section 4.3.4. Hence, *while only one operation can be performed in a column at a time, an operation can be performed in many columns simultaneously*. This gives MOUSE *column level parallelism*, which bears some resemblance to bit-serial architectures.

2.2.4 Alternative Memory Cell Architecture

Augmenting each MTJ in the MOUSE cell with a Spin Hall Effect (SHE) channel can further improve energy efficiency. This is the same technology as Spin-Orbit Torque (SOT) MRAM [269, 114], which will likely replace STT-MRAM. The SHE channel provides separate paths for reads and writes (where optimization targets conflict), allowing for separate optimization, thereby better energy-efficiency. SHE channels are CMOS/MTJ-compatible and fabricated prototypes exist [114]. Technology details of SHE integration in CRAM is covered in [401]. Four augmented cells in two rows and two columns are shown in Figure 2.4.

In this case, there are two word lines per row, word line for read (WLR) and word line for write (WLW). WLR connects the cell to the read path, via t_{read}. WLW connects the cell to the write path, via t_{write}. When t_{write} is activated, current only passes through the SHE channel (and not the respective MTJ). This current, while not affected by the state of the MTJ, can still change its state. This configuration is used when writing to the MTJ or when the MTJ is the target output of a logic operation. When t_{read} is activated, on the other hand, current passes through the SHE channel *and* the MTJ. This allows the MTJ state to affect the current that travels through it. This is used when reading the MTJ state and when the MTJ is used as an input to a logic operation.

The SHE channel has important benefits. Due to the separation of read and write paths, the required current density to induce switching is lower, allowing for a reduction in the energy of write and logic operations. This increased energy efficiency can provide a decrease in the overall execution time in energy harvesting scenarios, as will be shown in our evaluation. Additionally, as the output MTJ resistance no longer is in series with the input MTJ resistances in a logic operation, different input values become easier to distinguish, increasing the robustness of logic operations.

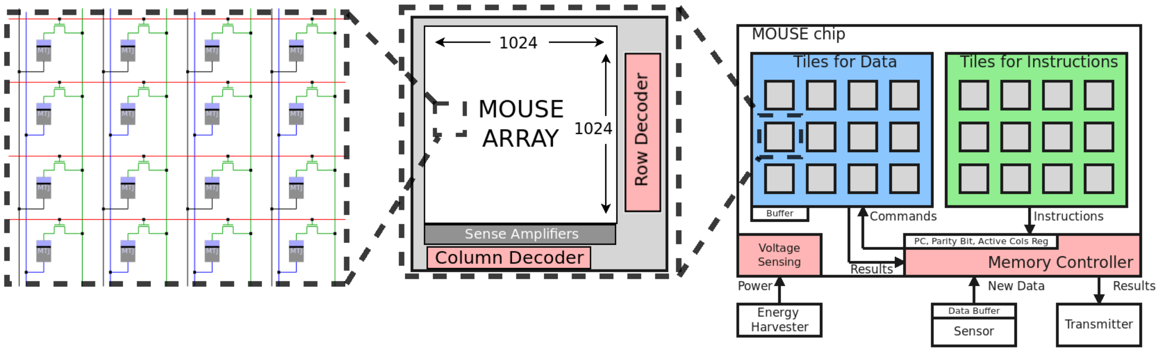


Figure 2.5: Overview of MOUSE. Each tile contains an array of MTJs along with a row and column decoder. Sense amplifiers are required for read/writes but aren't used in computation. Shown here is the STT (1T1M) configuration.

2.3 Case Studies

To show the capability of MOUSE, we implement Support Vector Machines (SVM) and Binary Neural Networks (BNN). Both are widely used machine learning algorithms. Generally speaking, whether SVMs or neural networks are a superior choice depends on the target problem, but applications overlap considerably and both are applicable in the energy harvesting domain, where both the energy and the area budget is stringent.

SVMs are effective and simple classifiers for typically smaller data sets. Particularly, we found SVMs to perform well on MNIST image recognition and human activity recognition. However, there is a trade-off, as SVMs can struggle with some problems. For example, we were unable to achieve reasonable accuracy on the speech recognition data set, which neural networks have performed well on [120].

For all SVM benchmarks we use a polynomial kernel with a degree of 2. For inference, the main computation is effectively performing the dot product between an input vector and each of the support vectors. The results of these dot products are then squared, multiplied by a set of coefficients, and finally summed together. By design, SVMs have two class outputs, where the sign of the output value is the classification.

In this work, we opt for the simplest extension to multi-class problems: we train a separate SVM for each possible output class. Each SVM has the task of identifying its assigned class. For example, MNIST has 10 different classes for digits 0-9. We train 10 SVMs each identifying each digit. The output is 10 scores for “how similar” the input is to each digit. We take the highest-score output of the 10 classifiers to be the final classification. We perform training offline in software and only consider inference acceleration on MOUSE.

BNNs are neural networks that have neurons and weights represented by a single bit each [76]. This enables multiplications to be replaced by XNOR operations and addition is simplified to a popcount operation. As a result, BNNs are much more energy efficient than full- or fixed-precision networks. They have been implemented efficiently in FPGAs in FINN [362] and FP-BNN [211]. We mimic their network configurations, modified only in transforming them to run on our PIM substrate. Hence, our accuracy is identical. Neural networks [390, 59] and BNNs [349, 396] have been previously mapped to PIM substrates for acceleration, including on CRAM [301]. However, those designs rely on continuous power and have not considered correctness in intermittent computing and thus are not capable of functioning in the targeted energy harvesting domain.

2.4 MOUSE Design

Energy harvesting systems are powered by their environment. If the environment does not provide enough power, the system will have to accumulate energy over time and consume it in bursts [120]. Therefore, such devices must consume as little energy as possible and be capable of tolerating power outages while maintaining program correctness. MOUSE is a natural fit for such a paradigm as logic operations are highly energy efficient and the memory is entirely non-volatile. Additionally, all computation occurs within the memory so progress is effectively saved after each operation. This greatly simplifies strategies to maintain correctness. In this section, we detail a basic MOUSE design which is tightly tailored to energy harvesting applications.

2.4.1 Hardware Organization

MOUSE has a tiled architecture. Certain MOUSE tiles are dedicated for instructions, while all others are dedicated for data and computation, as shown in Figure 4.1. MOUSE has a larger storage capacity than is typical for energy harvesting devices. This is due to two reasons. First, MRAM is dense and has extremely low standby power, giving the memory a low area and energy impact. For example, NVSIM [92] reports the size of 64MB STT-MRAM array –which is nearly twice the size of our largest configuration– as 15.12mm^2. 256MB and 1GB STT-MRAM memory manufactured by Everspin [1, 2] comes in a package that is 130mm^2. For reference, just the MSP430FR5994 micro-controller itself, commonly used as a sub-component of energy harvesting systems [120, 70, 153, 154, 155, 311], consumes over 100mm^2. Second, as there is no need for external processor logic or area costly volatile memory (such as SRAM), and due to minimal peripheral circuitry, nearly the entire area budget of MOUSE is available for memory arrays. That said, the SHE configuration has more area overhead than the STT configuration due to the presence of a 2nd transistor, which we expand upon in Section 7.5. However, the area budget still remains modest.

There are only five components of MOUSE that are not memory arrays:

1) A memory controller that reads instructions from the instruction tiles and issues all instructions;

2) An 128B memory buffer that facilitates reads and writes to the tiles;

3) A non-volatile register for Program Counter (PC);

4) A non-volatile register for buffering a single instruction;

5) Voltage sensing circuitry for monitoring the power source.

The memory controller only needs to differentiate between three instruction types as will be described in Section 4.3.4. All computation and memory operations are performed in the tiles, hence the controller need only broadcast the appropriate command to the tiles. The memory buffer is the same size as one row of the MOUSE tiles and is used for intermediate storage when transferring data to and from the tiles. The non-volatile registers are used for maintaining correctness during power outages, as will be described in Section 2.4.4. Finally, the voltage sensing circuitry is standard for energy harvesting systems, and is as described in [219].

2.4.2 Instructions

Instructions for MOUSE are 64-bit and the formats are shown in Figure 4.2. There are three types of instructions, logic operations, memory operations, and column activation. Memory operations are the same as standard read and write operations for MRAM. Instructions for logic operations specify the type of operation (which determines the applied voltage level) and the rows on which input and output cells reside. When a logic instruction is issued, it will be applied to every column that is currently active. Columns are activated by the *Activate Columns* instruction, which provides a list of column addresses to a column decoder. Once columns are activated they are held active by a latching mechanism as proposed by [209]. This allows columns to remain active over multiple instructions. As columns need to be changed infrequently, typically staying active for many instructions, the peripheral cost for activation is amortized. This cost is further reduced by modifying the encoding to allow for bulk addressing, similar to the procedure in [326].

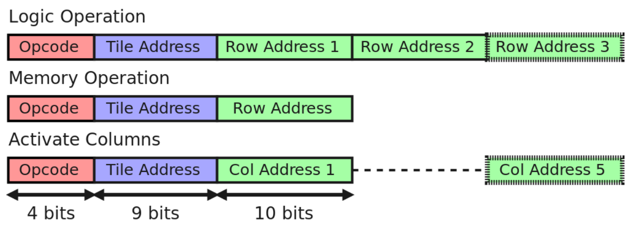


Figure 2.6: MOUSE instruction formats. There are three types of instructions, logic, memory, and an additional activate columns instruction for configuration. Opcodes are 4 bits; tile addresses, 9 bits; and row and column addresses, 10 bits each. Dashed items are optional.

Compiling instructions for MOUSE requires some knowledge of the hardware to make efficient use of potential parallelism. This situation is analogous to compiling for GPU architectures from Open-CL or CUDA code. Unfortunately there is no generic equivalent for PIM. In the following we will provide pointers for efficient compilation, but inevitably leave detailed exploration of this rich design space to future work. Otherwise, architecture and data layout for MOUSE is similar to a number of other works which have mapped applications to PIM substrates [209, 326], including BNN implementations [301].

While operations can occur in multiple tiles simultaneously, tiles do not operate autonomously. All operations are triggered by the memory controller (discussed in more detail in Section 4.4). Effectively, there is a single controlling "thread", and hence there are no concurrency concerns between individual tiles.

A subset of the tiles are dedicated to store the instructions. In the prototype MOUSE implementation, instruction and data tiles are homogeneous in design. The instructions are written into these tiles before deployment. Once active, the memory controller fetches each instruction from the instruction tiles, decodes it, and then broadcasts it to the tiles storing data. Instructions vary in the amount of time they take to complete. This is because specifying row and column addresses has an associated latency, and different instructions have different numbers of addresses. Logic operations can use 2 or 3 rows and column activation can specify up to 5 columns at a time. To ensure that every instruction finishes in time, the memory controller waits longer than the longest taking instruction needs before issuing the next. This time lapse forms a *cycle*. While this conservative approach comes at some cost of performance (more complex techniques could potentially issue instructions faster, in an event-driven fashion), MOUSE already is capable of extreme performance relative to other devices in this domain, as shown in Section 7.5. Additionally, energy efficiency (rather than throughput) is the limiting factor for energy harvesting devices. Hence, we opt for simplicity at the cost of some performance loss.

Finally, as we are only performing inference in MOUSE, the sequence of instructions performed doesn't change as a function of inputs at runtime. Instructions are performed in sequential order one by one until the program repeats.

2.4.3 Power Draw

Most energy harvesting devices utilize an energy buffer (capacitor), rather than having the power source directly attached [225]. This prevents the need to match the power consumption with the power source. A switched-capacitor voltage converter can be used to apply the all appropriate voltages to the device [148, 173, 289], including the voltages required to perform all logic gates (explained further in Section 7.5). MOUSE performs a single type of operation in each cycle. A portion of each cycle must be dedicated to changing the output voltage of the converter, if consecutive operations require different voltage levels. The converter may have an efficiency anywhere between 35-80%, hence the energy harvesting power source will have to provide energy in addition to that which MOUSE consumes.

By utilizing an energy buffer, MOUSE acquires energy over time and then consumes it in bursts. Hence, MOUSE could consume more power during power-on time than the energy-harvesting power source provides. However, we note that it is possible to reconfigure MOUSE to consume a specified power (to stay within a specified power budget), if this is known prior to deployment. By adjusting the amount of parallelism in the computation, the power consumption of MOUSE can be finely tuned. This enables a trade-off between latency and power draw. However, this can place strict limitations on potential parallelism. For example, if the power source can only deliver low power, e.g., $60\,\mu$W (an efficiency of 35% from a $171\,\mu$W power source), MOUSE would only be able to perform logic operations in 4 columns simultaneously (using the least energy efficient configuration described in Section 7.5).

Table 2.1: Four possible cases for re-performing an interrupted AND gate. The output MTJ either should or should not switch for correct operation, and it either did or did not prior to the power being cut.

	Output did not switch before interrupt	**Output did switch before interrupt**
Output should not switch	Repeating the operation is the same as performing it for the first time; no switching will occur (correct output).	Not possible. There cannot be sufficient current to induce switching at any point of the operation (be it before of after the interrupt). Repetition cannot induce switching by construction.
Output should switch	Repeating the operation is the same as performing it for the first time, and will now result in switching (correct output).	The output has already switched to 0 (correct output). Repetition, i.e., re-applying the same voltage will result in a larger current. Due to the *direction* of the current, however, staying the same (as before the interrupt), the output will remain at 0 (and cannot switch back to 1).

2.4.4 Intermittent Processing

As energy harvesting systems frequently experience power outages, they must be designed to perform intermittent processing. This involves addressing the challenge of maintaining correct state while repeatedly shutting down and restarting. The mechanism for maintaining correct state also needs to be efficient, as to avoid consuming the precious energy available for program execution. A number of techniques have been designed to ensure correctness [68, 305, 233, 119]. These studies have devised sophisticated techniques to ensure correctness while introducing minimal backup and restart overhead. In contrast, MOUSE maintains correctness with just a program counter (PC) and an additional non-volatile status bit. While extremely simple, and would be crude for other architectures, it is a natural fit for MOUSE. The simplicity of this technique is enabled by our novel architecture. More sophisticated techniques are unsuitable and unnecessary as MOUSE has no volatile data to backup. As MOUSE performs all computation within the non-volatile memory, progress is saved after each operation. This makes restarting after the last instruction possible and ideal.

When MOUSE restarts, only two pieces of information are required: the last instruction that was performed and the columns that were active. In order to restart from the last instruction, MOUSE writes; i.e., checkpoints, the PC into a non-volatile register after each instruction. When MOUSE gains sufficient power to restart, it simply reads the next instruction from the address in the PC. In the worst case, the power is cut after the last instruction is issued and performed, but before the update to the PC register. This does not break correctness as the same result is obtained if a single instruction is repeated multiple times, i.e., each such repetition is *idempotent*[369, 157] as will be shown in Section 2.5.1. The only requirement is that the PC checkpoint happens strictly after each instruction is performed. Restarting after the very last instruction not only minimizes the amount of work potentially lost on shutdown, but it also simplifies the restart process. The simple correctness guarantee, an operation being *idempotent*, does not hold if we were to repeat multiple instructions. This is because over the course of multiple instructions, temporary values can be created. These temporary values may be used later in the computation or periodically overwritten. Repeating multiple instructions on startup would require some method for ensuring correctness of these temporary values, such as performing additional presetting operations. This is certainly possible to do, but it introduces additional (and unnecessary, as we will see shortly) complexity.

The second requirement is to restore the previously active columns, for which we use a similar procedure. Whenever an *Activate Columns* instruction is issued, it is stored in an additional instruction register. Reissuing this last *Activate Columns* instruction is the first action on restart.

This scheme gives MOUSE minimal backup and restart overhead. To summarize, the cost is 1) continuous checkpointing of the program counter and *Activate Columns* registers and 2) an additional issue of an *Activate Columns* instruction on every restart. Both of these actions incur far less energy than a typical logic instruction. We make sure that operations happen in the correct order by performing them sequentially; updates to architectural state occur only after the current instruction is performed. It is noteworthy that MOUSE is always in a state which is safe to shut down in. Hence, MOUSE maintains correctness even if power is cut unexpectedly. We provide more detail on maintaining correct state in Section 4.4.2.

There is an efficiency trade-off in the frequency of checkpointing [231]. Doing so more often results in less work potentially lost on shut-down, however this also increases the checkpointing overhead. The optimal approach will depend on the power source. MOUSE consumes energy on every cycle to perform checkpointing. If energy-harvesting is able to supply sufficient power, making power interruptions less frequent, it is possible that MOUSE would be more energy efficient performing checkpointing less often. However, we opt to checkpoint on every cycle as this keeps the design complexity minimal, which is enabled by the energy efficiency of MOUSE's checkpointing.

2.4.5 System Integration

During inference, MOUSE itself holds all static data required and performs all the computation. To be integrated into an energy harvesting system, MOUSE needs to receive energy from an energy harvester, receive input from a sensor, and send output to a transmitter. In this work, we assume input data is stored in a non-volatile buffer in the sensor prior to inference. The sensor's buffer is assigned a tile address and is treated as one of the tiles. Additionally, the buffer contains a non-volatile valid bit indicating that new input is ready. When MOUSE is ready for new input, the memory controller can check the valid bit and trigger a memory transfer. The memory transfer then consists of reads from the buffer and writes to the MOUSE data tiles. These reads and writes can be controlled by instructions at the beginning of the program. When MOUSE finishes inference, the memory controller reads out the data from the tiles. This data is then available to be transferred to the transmitter. In this work, we focus only on the accelerator and do not consider any overhead for the sensor or transmitter.

MOUSE can also handle potential sensor data corruption due to power outage. A dedicated non-volatile register along with an instruction to orchestrate sensor reads achieves this. When sensor read begins, this instruction stores current PC in a dedicated register. If power goes out during reading sensor data, on restart, MOUSE checks the valid bit in the sensor buffer (which stays zero under corruption). If zero, MOUSE goes back to first instruction handling sensor read (getting PC from the dedicated register). MOUSE can checkpoint such PC at any code location. As an alternative design point, MOUSE can offload this orchestration to software, as well. Otherwise, if power outage happens during computation, MOUSE does go back in time, but at most by one operation. Going further back is unnecessary. The most recent checkpoint is guaranteed to be correct. Idempotency ensures that neither following operation nor following checkpoint can corrupt it. All data remains consistent, hence there cannot be corruption on reboot.

2.5 Correctness Guarantee

We show that correctness is guaranteed in spite of power outages, even when unexpected. There are two components, the correctness of individual operations when interrupted or re-performed (Section 2.5.1) and correctness of state variables in transitions between states (Section 4.4.2).

2.5.1 Operation Level Correctness

In this section we show that correctness is maintained if a single operation is repeated, i.e., that repeating any single operation is idempotent [369, 157]. Given that the power may be cut at any moment, we must consider what happens when an operation is interrupted in all its possible stages. Since all operations in MOUSE are threshold operations, the two stages are pre- and post-switching. Additionally, switching of the output MTJ either should or should not occur depending on the inputs. To be explicit, we use AND as an example, however, our observations here apply to all gates.

The preset value for the output of an AND gate is 1, meaning the MTJ has a high resistance. During operation, current is applied in a direction that could change the output state to 0. If either of the two inputs is 0, there will be a sufficient current to change the state, otherwise it will remain at 1. We show the four possible cases in Table 2.1: If, due to the inputs, the output is not supposed to switch, the output MTJ will not switch before the power is cut or after the power is restored. On the other hand, if the output is supposed to switch, it does not matter if it switches before the power outage or after.

If the output MTJ does not switch before the power outage, it will switch once power is restored and the operation is re-performed. If the output MTJ does switch to 0 before the power outage, re-performing the operation once the power is restored will leave the output at 0. This is because the direction of the current can only change the output to 0, it cannot revert it back to 1 due to basic MTJ physics.

The catch here is that repeating a logic gate is effectively the same as performing the gate for a longer duration. Doing so results in an identical outcome, regardless of whether the output MTJ switched before interruption (i.e., power outage) or not. The case for writes is even simpler. The result of a write operation does not depend on the preset value, hence repeating a write is effectively writing the value twice. Such power interruptions can lead to wasted energy (as we may end up re-performing unnecessary work) but cannot result in corruption of logical values.

We do not require idempotency beyond a single logic gate as we perform only one logic gate per cycle (per column). More complex operations (such as additions or multiplications) are broken down into individual gate operations, which are then performed sequentially in consecutive cycles (hence separated by checkpoints, as will be explained in Section 4.4.2). Our Boolean gate set is universal, allowing arbitrary computation.

2.5.2 Maintaining Correct State

It must also be ensured that the memory controller can tolerate unexpected interruptions and that MOUSE can maintain correctness as it transitions from one operation to the next during program

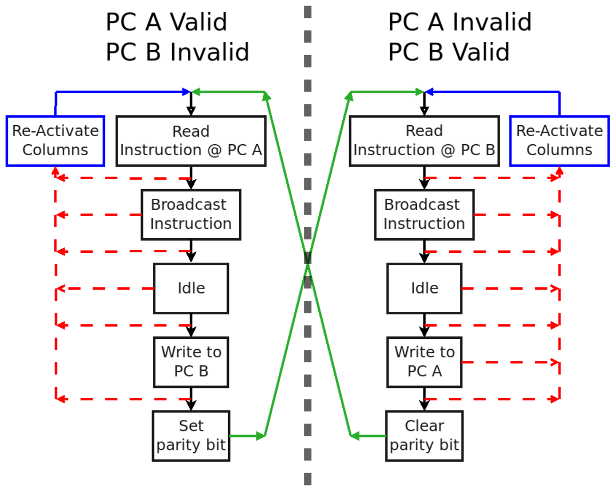


Figure 2.7: Memory controller's state transitions to ensure correctness of the program counter as MOUSE transitions from one instruction to the next. Effect of interrupts are dashed and highlighted in red, corrective measures in blue, and forward progress (guaranteed completion of an instruction) in green.

execution. Here, we describe how correctness of the architectural state variables and data is guaranteed during this process.

Architectural State

The memory controller reads instructions from the address held in the non-volatile program counter (PC), decodes them, and broadcasts them to the data tiles. It then updates the PC. If power is cut during a write operation to the PC, the value may be corrupt. We solve this by using two PC registers and maintaining a parity bit. We refer to these two registers as PC-A and PC-B. If the parity bit is 0 then PC-A is valid and if the parity bit is 1 then PC-B is valid. The valid PC register points to the instruction currently being executed.

After an instruction is completed, the value stored in the valid PC register is read, updated (to point to the next instruction), and the new value is then written into the invalid PC register. At this point the invalid (valid) PC register keeps the address of the next (current) instruction that is to be executed (completed). After the PC register update, the parity bit is flipped. This process is depicted in Figure 2.7.

With this scheme, a write is never performed on the currently valid PC, hence, a valid copy of the PC is maintained at all times. If power is cut after the update to the invalid PC but before the parity bit is flipped, the memory controller will consider the old PC to be valid on restart. This results in the previous instruction being re-performed, and cannot introduce errors (since individual instructions are idempotent), as explained in Section 2.5.1. The register holding the last *Activate Columns* instruction is also duplicated, and is handled in an identical fashion to the PC. Hence, power can be cut at any point during the execution of an instruction and the memory controller can always guarantee correct operation upon restart.

Data

The broadcast from the memory controller –which initiates an operation in the data tiles, and is depicted as *Command(s)* in Figure 4.1– is not atomic, and thus can be interrupted at any stage. However, this cannot cause corruption as the broadcast itself is idempotent. There are two cases to consider, 1) a broadcast initiating memory and logic instructions and 2) a broadcast initiating *Activate Columns* instructions.

As explained in Section 2.5.1, data in the MOUSE tile cannot be corrupted by an interruption during memory and logic operations –no matter what stage in its progression the operation gets interrupted. As a direct result, the broadcast cannot cause corruption as it's only effect is the initiation of the operation. Power can be cut before the broadcast reaches a tile, while the operation is being performed, or after the operation has finished –none of these cases can introduce error. The second case of *Activate Columns* instructions cannot result in any corruption either, as the peripheral circuitry is always re-configured after restart, which overwrites the action of the previous *Activate Columns* instruction. More fundamentally, no corruption can be the case if power was cut during an *Activate Columns* instruction, simply because no logic or memory operation can take place as an *Activate Columns* is in progress in the same tile.

2.6 Putting It All Together

Energy-harvesting devices need energy efficient execution and checkpointing capabilities. MOUSE uses non-volatile PIM to provide both.

A high-level program can be converted to computational blocks, such as multiplications or additions. These blocks can be broken into individual gates. For example, n-bit addition can be implemented by performing n full-adds, each of which can be performed with 9 NAND gates. These individual gates can be performed in the columns of MOUSE's tiles. The gates are highly energy-efficient, and applications can exploit high-degrees of paralellism available in the MOUSE tiles to achieve performance.

Scheduling these gates in the tiles is a two-dimensional problem in space and time, where we leave the rich design space for automation and optimization for future work. A multi-dimensional trade-off exists between parallelism, data-transfer, area consumption, and energy efficiency. Mapping computation to use more columns can increase parallelism, as computation can proceed in each column simultaneously. However, not only does this increase memory usage, but it also incurs a potential data-transfer overhead. Intermediate values will have to be transferred between columns, via reads and writes, in order to produce the final result. This decreases the energy efficiency. Without loss of generality, in this paper we stick to greedy scheduling in minimizing energy consumption and area usage by using the minimal number of columns; at a cost of latency.

For example, vector dot-products constitute the majority of SVM classification. We place as many as possible bits of the elements of two vectors into a single column, with extra rows available for scratch bits. The elements that do not fit are placed (aligned) into other columns. The vectors are next element-wise multiplied and then summed together with a sequence of gates. Finally, the partial sums are moved, via reads and writes, to a single column, where they are summed to produce the end result. By using many columns and multiple tiles, this can be performed for many vectors simultaneously.

Hence, a program on MOUSE consists of a sequence of logic gates in-memory, along with reads and writes to perform I/O and transfer data between tiles. These operations can be fully specified by instructions shown in Fig. 4.2. Memory instructions are either read or write. Logic instructions correspond directly to logic gates, such as NAND, NOT, etc. Activate Columns is a single instruction. Instructions of this format are stored in MOUSE's instruction tiles.

A simple memory-controller is responsible for reading the instructions, decoding the opcode, and then broadcasting the instruction (and necessary addresses) to the data tiles. After waiting a sufficient period of time (for instruction completion), the memory-controller updates the PC and "commits" the instruction by flipping the parity bit. The memory-controller needs only basic logic circuitry (for decoding) and a clock to keep track of time. Its functionality is analogous to the 1st, 2nd, and 5th stages of the classic 5-stage pipeline. It performs 1-Instruction Read, 2-Instruction Decode, and 5-Write Back (setting parity bit). The memory handles 3-Execution and 4-Memory Access.

MOUSE's strength is in its simplicity. All operations performed in-memory are inherently idempotent and automatically stored in non-volatile memory. Hence, MOUSE can be made intermittent-safe with lightweight additions to the memory controller. This includes duplicated, non-volatile copies of the PC and active columns registers.

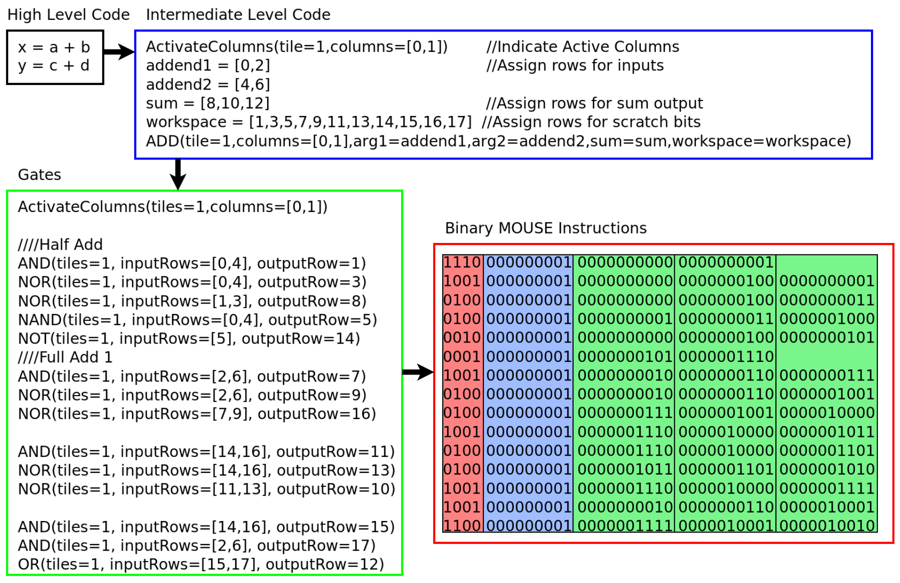
2.7 Application Mapping

We next provide a basic illustrative example for application mapping: 2-bit addition. Figure 2.8 shows the stages of converting high-level code to MOUSE instructions. The first step is conversion from high-level to intermediate-level, i.e., functional translation to required logic and memory operations for the underlying computation along with basic spatio-temporal optimization. Next, all of the specified operations get converted directly into MOUSE format instructions.

In Figure 2.8 two 2-bit integers are added to form a 3-bit integer. We can perform these additions in parallel, and choose for these to occur in columns 0 and 1 of tile 1. The first addends (a and c) are assigned to rows 0 and 2, and the second addends (b and d) are placed in rows 4 and 6. The sums (x and y) are chosen to be placed in rows 8, 10, and 12. The computation will require additional scratch bits (workspace), for which we assign the odd rows between the addends and sum, and some additional even and odd rows at higher row addresses, picked based on availability.

Given the location of the addends, sum, and workspace, the ADD function will generate the sequence of gates required to compute the sum. It does this by performing a half-add, and then as many full-adds required to complete the addition; in this case just 1. Note that all gates have inputs and outputs on opposite parity rows. The parallelism of these gates is determined by the active columns, which are set by the *Activate Columns* instruction. Since the operands are in columns 0 and 1, a single *Activate Columns* instruction is issued to activate these two columns for computation. Prior to this computation, the preset value for outputs for gates will need to be written into the respective columns. For simplicity we do not show this step, but it consists only of write instructions. After finishing all instructions, the sums reside in rows 8, 10, and 12, with x in column 0 and y in column 1.

Once the sequence of gates has been fully specified, it gets converted directly to instructions by replacing the gate with the corresponding opcode and inserting the tile and row addresses. The opcode specifies how many row or column addresses are required. As all instructions are 64-bit, a number of bits remain as don't care.



```
x = a + b
y = c + d
```

```
ActivateColumns(tile=1,columns=[0,1])       //Indicate Active Columns
addend1 = [0,2]                             //Assign rows for inputs
addend2 = [4,6]
sum = [8,10,12]                             //Assign rows for sum output
workspace = [1,3,5,7,9,11,13,14,15,16,17]  //Assign rows for scratch bits
ADD(tile=1,columns=[0,1],arg1=addend1,arg2=addend2,sum=sum,workspace=workspace)
```

```
ActivateColumns(tiles=1,columns=[0,1])

////Half Add
AND(tiles=1, inputRows=[0,4], outputRow=1)
NOR(tiles=1, inputRows=[0,4], outputRow=3)
NOR(tiles=1, inputRows=[1,3], outputRow=8)
NAND(tiles=1, inputRows=[0,4], outputRow=5)
NOT(tiles=1, inputRows=[5], outputRow=14)
////Full Add 1
AND(tiles=1, inputRows=[2,6], outputRow=7)
NOR(tiles=1, inputRows=[2,6], outputRow=9)
NOR(tiles=1, inputRows=[7,9], outputRow=16)

AND(tiles=1, inputRows=[14,16], outputRow=11)
NOR(tiles=1, inputRows=[14,16], outputRow=13)
NOR(tiles=1, inputRows=[11,13], outputRow=10)

AND(tiles=1, inputRows=[14,16], outputRow=15)
AND(tiles=1, inputRows=[2,6], outputRow=17)
OR(tiles=1, inputRows=[15,17], outputRow=12)
```

```
1110 000000001 0000000000 0000000001
1001 000000001 0000000000 0000000100 0000000001
0100 000000001 0000000000 0000000100 0000000011
0100 000000001 0000000001 0000000011 0000001000
0010 000000001 0000000000 0000000100 0000000101
0001 000000001 0000000101 0000001110
1001 000000001 0000000010 0000000110 0000000111
0100 000000001 0000000010 0000000110 0000001001
0100 000000001 0000000111 0000001001 0000010000
1001 000000001 0000001110 0000010000 0000001011
0100 000000001 0000001110 0000010000 0000001101
0100 000000001 0000001011 0000001101 0000001010
1001 000000001 0000001110 0000010000 0000001111
1001 000000001 0000000010 0000000110 0000010001
1100 000000001 0000001111 0000010001 0000010010
```

Figure 2.8: An application mapping example of parallel 2-bit integer addition. Variables are assigned to rows and columns. Independent operations are mapped to separate columns for parallel execution. Computation is broken down into individual logic gates, which directly correspond to individual MOUSE instructions. During operation the memory controller issues each instruction in sequence. The MOUSE instructions shown are in the formats specified in Figure 4.2.

2.8 Evaluation Setup

Benchmarks: Energy harvesting systems are ideal for applications in which the system is difficult or inconvenient to power directly or with batteries. Examples include remote sensors and wearable tech. We choose benchmarks which are representative of different possible use cases, along with an additional standard benchmark.

MNIST [198], as an example small-scale image recognition for sensor networks, is a digit recognition data set, where there are 10 classes for digits 0-9. The input is a grey scale 28×28 pixel image with 8-bit precision. We use both BNNs and SVMs on this benchmark. For the SVM, the pixels are placed row wise into a 784 element vector. We also use a binarized version, where pixels that are greater than a threshold value are set to 1 and others to 0. This allows us to replace multiplications with AND gates for most parts of the computation. For BNNs, we tailor the network configuration of FPGA-based FINN [362] and FP-BNN [211] to function properly on MOUSE, by converting it to sequences of logic gates. Our logical configuration is exactly the same. Hence, our accuracy is identical. The FINN configuration only uses binarized input. It has three hidden layers of 1024 neurons (bits) each, and an output layer of 10 neurons with 10-bit precision. The FP-BNN configuration only uses 8-bit inputs. It has three hidden layers of 2048 neurons and the output layer has 10 neurons with 16-bit precision.

Human Activity Recognition (HAR) [14], as an example for wearable tech, is a data set con-

Table 2.2: Parameters for MTJ devices.

Parameter	Modern	Projected
P State Resistance	$3.15\,\mathrm{k\Omega}$	$7.34\,\mathrm{k\Omega}$
AP State Resistance	$7.34\,\mathrm{k\Omega}$	$76.39\,\mathrm{k\Omega}$
Switching Time	3 ns [307, 268]	1 ns [398, 167]
Switching Current	$40\,\mu$A [307]	$3\,\mu$A [398]

taining measurements from an accelerometer and gyroscope embedded in a smartphone, which is carried by participants performing a variety of activities. The task is to classify each set of readings to which activity is being performed. We represent the input with fixed point integer format with 8-bit precision. Each input is a vector of 561 elements.

ADULT [188] is a commonly used benchmark for SVMs that contains census information and the task is to classify whether an individual makes greater than $50K per year or not. We use a reformatted version of the data set from libSVM [57]. Each input is a 15 element vector where each element is an 8-bit integer.

Our SVMs are trained and tested in R [288]. They are custom designed, however we do compare our results with libSVM [57] with the same inputs and obtain similar accuracy. In our custom implementation we do not use any operations that would be inefficient in MOUSE; all programs consist of bit-wise and integer arithmetic.

Performance and Energy Model: We simulate the benchmarks on MOUSE with an in-house simulator, also implemented in R. MOUSE has a tiled architecture. We set each tile to have a capacity of 128KB, which is an 1024x1024 array. We chose this size as it is a commonly recommended subarray size for non-volatile memories from NVSIM [92]. While only one logic gate can be performed in one column in each cycle, the gate can be performed in all 1024 columns simultaneously (column-parallelism) and in each tile simultaneously (tile-parallelism). Note that operating 1024 columns in parallel would require approximately 15 mW (on the least energy-efficient MOUSE configuration), which may exceed available energy. Additionally, high levels of parallelism can increase the restart cost during intermittent computing. This is because re-performing the last instruction on restart would cost more energy if it is highly parallel.

We experiment with both modern MTJ parameters [308] and projections of MTJ parameters in the next few years [398, 401]. Projected improvements in MTJ devices are expected to significantly increase energy efficiency. The MTJ parameters we use are shown in Table 7.1. For projected MTJs, two techniques enable a reduction in the switching current, 1) decreasing the damping constant of ferromagnetic materials [315, 252, 97] and 2) using a dual-reference layer structure [161, 89]. To be conservative, we assume $3\,\mu$A, however, switching currents as low as $1\,\mu$A are possible. For projected MTJs, we test with both the STT and SHE based architectures. The main benefit of SHE is providing a write path through the SHE channel, rather than through the MTJ itself. To capture this effect, we assume a $1\,\mathrm{k\Omega}$ resistance for the SHE channel, which is in series with the input MTJs in logic operations. This provides a conservative estimate of the SHE energy efficiency.

For Modern MTJs MOUSE operates at 30.3 MHz and for projected MTJs MOUSE operates at 90.9 MHz. This enables sufficient time for MTJ switching and peripheral circuitry latency.

To estimate latency and energy cost due to peripheral circuitry, we take data from NVSIM [92] which reports results for modern MRAM memories. We set our peripheral circuitry costs so that they consume the same percentage share of the total latency and energy as reported by NVSIM. In addition to the latency and energy required for performing the instructions, we also account for the overhead involved in reading the instructions from the tiles, updating the program counter and valid bits, specification of row and column addresses, storing the most recent *Activate Columns* instruction, and the re-issuing of the last *Activate Columns* instruction whenever the system restarts.

We first evaluate the performance of MOUSE under continuous power. Then, we evaluate MOUSE under energy harvesting conditions. We model our energy harvester as a constant power source which is filling an energy buffer (capacitor). MOUSE will start executing when the voltage on the capacitor is sufficiently high, and will shutdown when the voltage drops to a pre-determined level.

For Modern MTJs, the voltage on the capacitor fluctuates between $320\,\text{mV}$ and $340\,\text{mV}$, and for Projected MTJs the range is $100\,\text{mV}$ to $120\,\text{mV}$. We use switched-capacitor converters for up-conversion and downconversion [148] to supply the required voltage for the operations. By using conversion ratios of 0.75, 1, 1.5, and 1.75 [173, 289], we can supply all voltages required. We evaluate MOUSE on the power supplied by the converter, the evaluation does not include regulator efficiency overhead. The converter may have an efficiency anywhere between 35-80%, hence the energy harvester may need to provide roughly 1.25-2.85$\times$ the energy that MOUSE consumes.

While energy harvesters can fluctuate in the amount of power they provide (e.g., amount of sunlight), this model captures a representative operation. We sweep the power source over a wide range, from levels well below the operating power of MOUSE, incurring numerous power outages, up to levels where MOUSE can nearly be continuously powered. Additionally, MOUSE assumes no knowledge when power will run out or when it will be restored. Effectively all outages are “unexpected”.

Following metrics provided in [312], we report energy dedicated to different components. In addition to total energy, we report Backup energy, Dead energy, and Restore energy. Backup captures operations performed prior to shut down to save state. For us, this is the continual writing of the PC, parity bit, and storing each *Activate Columns* instruction in an additional instruction register. Dead energy is energy spent re-performing work that was lost during shut down, which in this case is repeating the last instruction on restart. Restore energy includes any operation needed to prepare MOUSE for computation on restart. For us, this is issuing the most recent *Activate Columns* instruction.

We also report Dead latency, which is the latency associated with re-performing instructions, and Restore latency, which is the time it takes to re-activate the columns on restart. There is no Backup latency, as backup operations occur during the same cycle as each instruction.

Area Overhead: MOUSE tiles have a similar area overhead to MRAM arrays. MOUSE has an extra bit line per column for the STT configuration. For the SHE configuration, it has an extra transistor and a SHE channel for each cell. The impact of the additional bit line is minor but the additional transistor has significant overhead. To estimate area overhead for MOUSE, we create estimates for cell size keeping access transistor resistance less than $1\,\text{k}\Omega$. The access transistors dominate the area overhead. This is for two reasons: 1) the MTJs and SHE channel can be placed on a separate layer from the access transistors and 2) the access transistors are much larger. As the SHE design has twice as many access transistors, the cell area is approximately twice as large. To estimate peripheral circuitry area overhead, we take NVSIM [92] results for area efficiency for the same sized

Table 2.3: Area required for MOUSE for different benchmarks and configurations. Units are in mm^2.

Benchmark	Total Memory	Modern STT[400]	Projected STT [400]	SHE
SVM MNIST	64MB	50.98	38.67	77.35
Binarized	8MB	5.43	4.13	8.24
SVM HAR	16MB	10.86	8.24	16.48
SVM ADULT	1MB	0.71	0.53	1.06
BNN FINN MNIST	8MB	5.43	4.13	8.24
BNN FPBNN MNIST	16MB	10.86	8.24	16.48

Table 2.4: Continuously powered MOUSE (using STT design and modern MTJ devices) and related work under continuous power. The CPU does not benefit from MNIST binarization as it still performs 64-bit integer multiplication.

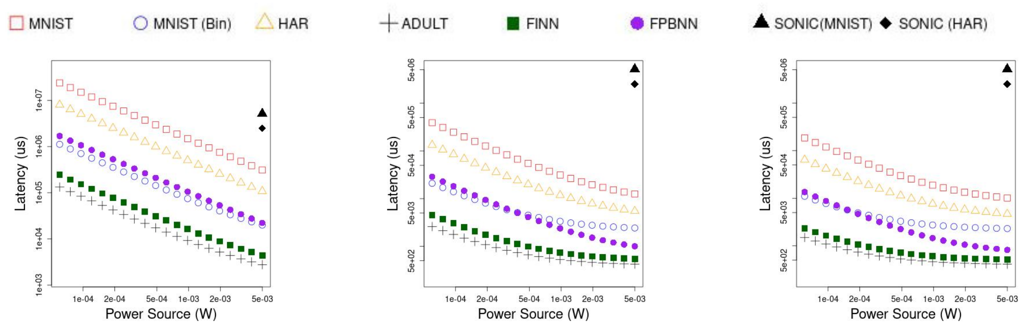
Benchmark	Latency (μs)	Energy (μJ)	#SV	I/D Mem (MB)	Area (mm^2)	Accuracy
SVM (CPU)						
MNIST	169,824	5,094,702	11,813	-	-	97.55
MNIST (Binarized)	192,370	5,771,085	12,214	-	-	97.37
HAR (integer) [14, 360]	127,494	3,824,822	2,809	-		95.96
ADULT	4,368	131,052	1,909	-	-	76.12
MOUSE SVM (Modern STT)						
MNIST	23,936	1,384	11,813	4.5 / 30.0	50.98	97.55
MNIST (Binarized)	6,575	65.49	12,214	1.25 / 6.0	5.43	97.37
HAR (integer) [14, 360]	11,805	468.6	2,809	2.25 / 10.0	10.86	94.57
ADULT	1,189	7.24	1,909	0.25 / 0.5	0.71	76.12
MOUSE BNN (Modern STT)						
MNIST (Binarized) FINN	1,485	14.33	NA	3.15/1.71	5.43	98.4
MNIST FP-BNN	2,007	99.9	NA	4.20 / 8.00	10.86	98.24
libSVM [57]						
MNIST	7,830	234,900	8,652	-	-	98.05
MNIST (Binarized)	19,037	571,116	23,672	-	-	92.49
HAR (integer)	1,701	51,042	2,632	-	-	93.69
ADULT	379	11,370	15,792	-	-	78.62
SONIC [120]						
MNIST	2,740,000	27,000	NA	0.256	>100	99
HAR	1,100,000	12,500	NA	0.256	>100	88

arrays and adjust our estimates by the same ratio. As NVSIM only works with memory capacities that are a power of 2, we assign the smallest memory size for which the entire benchmark will fit. For example, SVM MNIST requires only 34.5MB yet we assume MOUSE will consume 64MB of memory to perform this benchmark. Our conservative area estimates are shown in Table 4.3.

2.9 Evaluation

Continuous Power: Results for MOUSE under continuous power are summarized in Table 4.4. Also reported are results for the same benchmarks using both our custom SVM and libSVM on a CPU, and a representative energy harvesting system SONIC [120] under continuous power. The CPU implementations are run on a supercomputing cluster using Intel Haswell 5-2680v3 processors. To be conservative, we account only for the processor power consumption and assume it operates at its idle power. SONIC uses a TI-MSP430FR5994 microcontroller and is powered by a Powercast P2210B energy harvester.

Overall, MOUSE shows significant energy efficiency advantages over other implementations,



(a) Modern STT (b) Projected STT (c) SHE

Figure 2.9: Latency (μs) vs. Power Source (W) for each MOUSE configuration and SONIC [120].

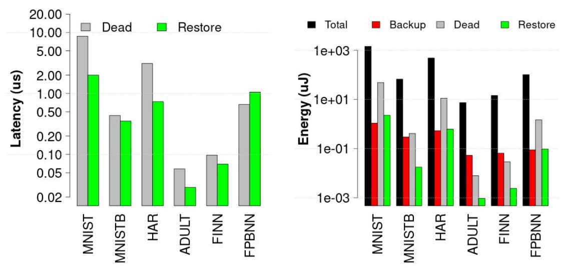
and competitive latencies. MOUSE does require more memory than SONIC, however, we believe this to be reasonable given that MOUSE is implemented in high density MRAM and does not need external processing logic or area costly volatile memory. MOUSE benefits greatly from binarizing the MNIST input. One bit inputs enable us to replace multiplications with AND gates, which significantly reduces the amount of computation required. This comes at a small cost in accuracy. The libSVM implementation struggles on the binarized MNIST inputs, and attempts to increase accuracy by adding many more support vectors. This increases the latency and energy of inference.

Let us next look into the significant difference in performance between MOUSE and SONIC [120]. SONIC is implemented on a conventional, low performance microprocessor. That design is highly economical, makes use of very scarce memory capacity, uses currently commercially available hardware, and has been proven experimentally. Additionally, the authors note that there is room for significant improvement in the efficiency. While we are reporting a significant latency and energy advantage, MOUSE is not fabricated yet. However, MTJ based logic has been experimentally demonstrated [378]. That said, MOUSE uses roughly the same area budget –that SONIC allocates for energy-hungry volatile memory and relatively complex logic– for more non-volatile memory (within which computation can be performed)[1].

Energy Harvesting: Now we consider MOUSE in a more realistic energy harvesting scenario. We test MOUSE with a range of power sources, from $60\,\mu$W, approximately what can be harvested from a 1cm^2 thermal energy harvester running on body heat [202, 181], up to $5\,$mW, the power harvested by SONIC [120]. The power source charges an energy buffer (capacitor) on chip. We allow the voltage to fluctuate between $320\,$mV and $340\,$mV when using Modern MTJs and between $100\,$mV and $120\,$mV when using Projected MTJs. When the voltage drops below the desired range, MOUSE shuts down and waits until the voltage reaches the upper end of the range. We assume that MOUSE starts with a capacitor with less charge than what would correspond to the shutdown voltage. Hence, all benchmarks begin with an initial charging time. We use a $100\,\mu$F capacitor (energy buffer) with Modern MTJs and a $10\,\mu$F capacitor for Projected MTJs. The optimal capacitor size depends on the technology and the program being executed. When deployed, a system such as Capybara [71] could be used to tune the parameters of the energy buffer.

Results for latency are plotted and compared to SONIC for Modern STT, Projected STT, and SHE in Figure 4.7. Consistent with and as noted by [120], the latency is mostly determined by

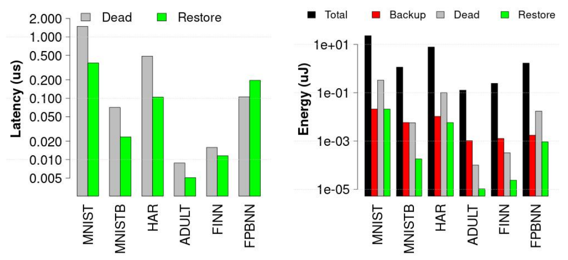
[1]Our largest configuration uses 35MB (fits in an 64MB array). Everspin's commercially available 64MB STT-MRAM device (similarly sized) is 130mm^2 [1]. Everspin's new 1GB product is the same (package) size [2]. In comparison, SONIC's microcontroller takes 100mm^2, which is only a sub-component.



(a) Latency (b) Energy

Figure 2.10: Latency/Energy Breakdown: Modern STT.

energy efficiency. This is because the majority of the latency is spent powered off, waiting for the capacitor to charge. The fewer recharges required, the lower the latency. Hence, latency increases significantly as the power source is reduced. Because the SHE design is more energy efficient, it draws significantly less power and thus drains the capacitor less often. This results in fewer power outages, fewer shutdown and restart operations, and hence also requires fewer operations to complete the program. This gives SHE a latency advantage over STT while in energy harvesting conditions. However, with all configurations, MOUSE achieves a significantly lower latency than SONIC, even with a much lower power budget.



(a) Latency (b) Energy

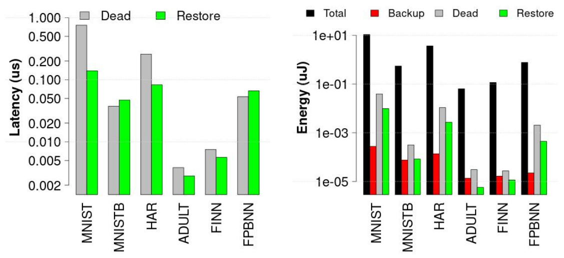
Figure 2.11: Latency/Energy Breakdown: Projected STT.

The dependency of latency on energy efficiency results in a cross-over of the latency between FP-BNN and SVM MNIST (Bin) benchmarks. FP-BNN costs more total energy, and hence has a higher latency at lower power sources. However, due to a higher degree of exploited parallelism, FP-BNN has a lower latency if sufficient power can be supplied.

As MOUSE spends negligible amounts of energy while powered off, the energy consumption is nearly independent of the power supply. The total energy is plotted in Figure 4.5(b) for Modern STT; in Figure 4.6(b) for Projected STT; and in Figure 4.7(b) for SHE; assuming a $60\,\mu$W power source.

Also of interest, as noted by [312], is the Backup, Restore, and Dead latency and energy. These are also reported in Figure 4.5 for Modern STT; in Figure 4.6, for Projected STT; and in Figure 4.7, for SHE. Note that the y-axis is log scale. The total energy encapsulates all energy used for computation, as well as Backup, Restore, and Dead energy. An efficient intermittent computing system will have low Backup, Restore, and Dead energy relative to the total energy, which can be seen is the case for MOUSE. Also note the total latency is provided for all architectures in Figure 4.4 – where the breakdown figures capture the data for the $60\,\mu$W power source.

Dead latency and energy is due to the possible re-execution of the previous instruction on



(a) Latency (b) Energy

Figure 2.12: Latency/Energy Breakdown: SHE.

restart. Dead latency and energy also increase with the number of restarts required and is also variable on when the interrupt occurred. The best case is if an interrupt occurred immediately after the parity bit is flipped (indicating the completion of an instruction per Section 4.4.2 and Figure 2.7). In this case, there is effectively no penalty. The worst case is if the interrupt happened just before the flipping of the parity bit, where the current instruction has been fully performed but not considered complete. In this case, the entire instruction will be re-performed on restart.

As Modern STT is the least energy efficient, it must restart the most and hence has the largest relative Dead energy. At the extremely low power of $60\,\mu$W, on average, across all benchmarks, Dead energy is 7.4% of the total energy. Higher energy efficiencies reduce this significantly, where Dead energy (on average) becomes 2.52% of the energy for Projected STT and 0.61% of the total for SHE. Dead latency, on the other hand, is 0.47% of the total for Modern STT, 0.09% of the total for Projected STT, and 0.044% of the total for SHE.

Restore is the time and energy required to re-activate the columns every time MOUSE restarts. Restore latency and energy naturally increase with the number of restarts required, but is also variable depending on where in the program an interrupt occurred. The more columns that were active at the time of an interrupt, the higher the respective Restore cost of re-activating them will be. However, overall, as the restore process is fast and energy efficient, the Restore latency and energy remain a small fraction of the total. On average across all benchmarks, Restore is only 0.91% of the latency and 0.50% of the energy for Modern STT; 0.14% of the latency and 0.13% of the energy for Projected STT; and 0.04% of the latency and 0.13% of the energy for SHE. As Restore latency and energy is due to peripheral circuitry, SHE has no advantage over STT for an individual restart. However, SHE still requires fewer restart operations due to its overall increased energy efficiency.

Backup energy entails the continual writing of architectural state variables (PC and parity bit). Backup energy corresponds to writing only a few bits on every cycle. An interrupt can cause at most a single additional write. Hence, Backup energy is not significantly affected by interrupts and is determined mostly by the length of the program. Backup energy is, on average across all benchmarks, 0.24% for Modern STT; 0.27% for Projected STT; and 0.007% for SHE. Backup has no associated latency as it is performed at the same time as each instruction on every cycle.

Restore and Dead latency and energy are all zero for the case of a continuously powered system. This is because there are no power outages and, hence, never a need to restart the system or re-perform any potentially unfinished instructions.

2.10 Related Work

Non-volatile processors (NVP) [231, 219] are uniquely designed for intermittent computing by integrating non-volatile memory near the compute units. Unlike MOUSE, these devices have a structure similar to traditional CPUs. MOUSE has three advantages over these architectures. Due to PIM, it is capable of high degrees of parallelism in order to achieve performance. Additionally, MOUSE doesn't need to perform energy-hungry loads and stores in order to operate on data. Finally, MOUSE doesn't perform additional backup operations prior to shut-down, making it effectively immune to unexpected power outages. The authors of [219] propose a system using a THU1010N non-volatile processor for energy harvesting applications. They describe trade-offs in designing such a system and demonstrate its capability on a number of benchmarks. There is follow up work in [230, 229] which makes the NVPs more resistant to power interruptions. The NVP in [229] can complete the FFT benchmark from MiBench [142] in $4.2\,\mathrm{ms}$. A recent paper [80] has evaluated FFT implementations on CRAM, the same substrate which MOUSE uses. Performing a similarly sized problem, the best latency they were able to achieve is $1.63\,\mathrm{ms}$. Naturally, adapting this implementation to be intermittent safe in the same manner in MOUSE would introduce a latency penalty. Another non-volatile processor is presented in [340] which features PIM components. There is a controlling CPU that performs logic and control. A few RRAM arrays are used to accelerate computing in neural networks. In this case, the PIM is a sub-component of the system, which also contains more traditional logic circuitry and an external processor. Due to this complexity, this implementation cannot make use of the same checkpointing strategy used in MOUSE.

A recent paper, ResiRCA, proposes an adaptable RRAM crossbar accelerator for MAC (multiply+accumulate) operations for CNNs in energy harvesting environments. It proposes clever methods to adapt the power consumption to varying power sources. While the crossbar is powered by an energy harvester, it assumes a battery powered host processor. Unlike MOUSE, computation also occurs outside the memory array (only MACs are processed by the memory). Hence, the automatic check-pointing mechanism we use is not applicable to this design. A number of RRAM PIM technologies also exist [396, 390, 350, 342]. However, the RRAM array is used as an accelerator as a sub-component of the system. Hence, there is much additional circuitry and logic that occurs outside the memory. This significantly increases the difficulty to adapt to intermittent processing. Additionally, many RRAM accelerators rely heavily on ADCs (analog to digital converters), which have a significant area and energy overhead. RRAM typically suffers from a lower endurance, as well.

Capybara uses a re-configurable hardware energy storage mechanism and a software interface that allows the specification of energy needs for different tasks. This gives the system more flexibility in satisfying the requirements of different kinds of tasks. While we do not focus on the power delivery system in this work, systems such as Capybara could be used to optimally supply MOUSE with power. Hibernus [23], on the other hand, is a system that reactively hibernates and wakes up. This is a similar shutdown policy to MOUSE. However, Hibernus performs an additional back-up operation before shutting down, whereas MOUSE does not need to.

A number of techniques have been developed to enable intermittent computation on more traditional hardware. For example, CleanCut [68] works with LLVM to compile programs with checkpoints, and uses a statistical energy model to find potential non-terminating paths. Chinchilla [233] uses adaptive checkpointing, where the frequency of checkpoints is a function of the number of interrupts. Coati [305] developed methods to ensure correctness in the presence of interrupts for intermittent systems. The What's Next Intermittent Architecture [112] uses approximation to improve performance. Rather than following an all-or-nothing approach, What's Next computes approximate results and continually improves the output. If an acceptable output is achieved it will skip to processing the next input. This enables the device to process more inputs as it does not waste time and energy achieving unnecessary accuracy.

The EH model [312] facilitates early design space exploration for energy harvesting architectures. It helps finding a good balance to achieve minimal overhead for allowing maximal forward progress. As noted by the authors of [312], energy harvesting systems can generally be divided into two types, multi-backup, which perform many backups between power outages, and single back-up, which only save state once before a power outage. Multi-backup systems include Mementos, [290], DINO [227], Chain [67], Alpaca [232], Mayfly [156], Ratchet [369], and Clank [157]. Single-backup systems include Hibernus [22], QuickRecall [170], and many others [15, 21, 31, 218, 228]. According to this categorization, MOUSE fits under a multi-backup system as we are constantly saving the architectural state.

PIM has been studied for non-volatile memories with Pinatubo [209], for DRAM with Ambit [326], and for SRAM with Neural Cache [98]. These technologies are meant to be integrated into the memory hierarchy of traditional CPUs and have not been considered for energy harvesting applications. Ambit and Neural Cache are not suitable for energy harvesting as they are volatile technologies. Pinatubo could be adapted and used similarly as CRAM in MOUSE. However, Pinatubo uses logic external to the memory array for some operations. This adds complexity as these circuits would need to be protected against errors in intermittent computing. Additionally, Pinatubo uses sense amplifiers to perform computation, which is less energy efficient than the logic operations in CRAM.

The Phoenix processor [324] is an extremely low power processor with a sophisticated sleep strategy. However, it is not designed to be safe for intermittent processing. Similarly, while not safe for intermittent computing (and where adding this functionality would likely incur a significant performance and efficiency cost), a number of accelerators have demonstrated high performance and energy efficiency on inference. PuDianNao [216] is an ASIC accelerator which also targets SVM. A microcontroller based system was used as a BNN accelerator in [75]. An in/near memory SRAM substrate is proposed in [379], which performs bit-serial arithmetic, and which was shown to have high performance and efficiency on the AlexNet [190] network. A few analog PIM accelerators also exist. For example, the accelerator in [172] uses BNN to perform Cifar-10 image classification. The accelerator in [403] uses SRAM cells and analog computation to achieve high energy efficiency while classifying MNIST. Another example is [366] for MNIST and Cifar-10 recognition. Generally, adapting such accelerators to support safe intermittent computing is not straight-forward and would likely come –if at all possible– at significant performance and efficiency cost.

Orthogonal to our work, recent papers have made progress on problems relevant in the energy harvesting domain. Low power and accurate time keeping was developed in [84]. SRAM was used as a an efficient check-pointing memory, being able to maintain state for short periods of power off time [385]. A new platform for intermittent computing is proposed in [189] which simplifies the task of adapting pre-existing embedded applications to work in intermittent environments.

2.11 Conclusion

In this paper we presented MOUSE, a machine learning accelerator in (non-volatile) memory for energy harvesting applications. The requirements for energy harvesting applications are extreme energy efficiency, efficient shut down and restart procedures, and correctness during intermittent execution. MOUSE provides all of these by having highly energy efficient logic operations with simple and effective shut down and restart procedures. The non-volatility combined with processing in memory provides a natural progress saving mechanism which demands very little overhead. By simulation, we demonstrated that such a device would provide significant latency and energy efficiency advantages over state of the art approaches, and is a promising candidate to bring machine learning to new domains.

Chapter 3

Cryogenic PIM: Challenges & Opportunities

3.1 Introduction

While Moore's Law has lasted longer than expected, nothing lasts forever. Transistor scaling will continue to become more challenging as the years go on. This limitation makes it difficult for computer architects to continue designing systems with higher performance. A possible solution to this problem is *cryogenic computing*, where the processor and supporting memory structures are cooled to very low temperatures. The boiling point of liquid nitrogen (77K) is a common temperature target. This may seem like an extreme solution, but it offers some very attractive advantages. Electrical circuits operate faster and more energy efficiently than at room temperature, enabling further increases in computer performance.

However, a challenge for cryogenic systems is cooling cost. Heat dissipation can result in inordinate cooling costs, unless the architecture is re-designed to reduce power consumption [50]. This raises the question if Processing-in-Memory (PIM) architectures –specifically, architectures which fuse logic and memory functions within an array– are ideal candidates for cryogenic operation. PIM architectures exhibit extreme energy efficiency, due to avoiding energy-costly data transfers between the CPU and memory [257, 83]. Hence, PIM can operate within a very low power budget, which can significantly lower the cooling cost.

PIM can also provide performance improvement. A current limitation for computer performance is the *memory wall*. As CPU performance has increased over recent decades, memory demand has outgrown the increases in memory performance [152]. Combined with recent trends of increased data usage, modern applications are typically *memory bound*, meaning their performance is limited by the memory latency. PIM architectures alleviate the memory wall by performing computation where the data resides, avoiding much of the latency due to data transfer. Cryogenic operation will actually reduce the impact of the memory wall, as cold temperatures significantly improve the performance of memory - DRAM latency is reduced by a factor of $4\times$ at 77K [200]. However, increases in logic frequency, roughly 30-40% [276], and the corresponding increase in CPU frequency at cryogenic temperature, will increase the memory request rate, counteracting some of the benefit. Hence, it likely that cryogenic architectures (consisting of a CPU with supporting memory hierarchy) will still suffer performance loss due the memory wall, leaving room for potential performance increases by utilizing PIM.

Yet another benefit of PIM is that the architecture is easier to adapt to cryogenic temperatures. As they use predominately (slightly modified) memory structures, they are highly homogeneous and simple relative to more traditional systems. This makes any redesign required much easier than for more complex architectures where logic, memory, and their interface need to be optimized separately.

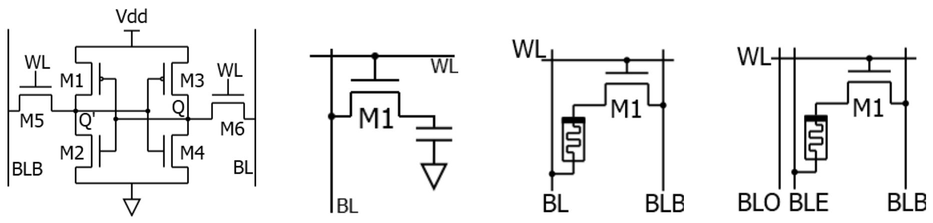
Numerous PIM substrates exist, including in SRAM [9, 8] and DRAM [325]. There is also a number of non-volatile PIM (NV PIM) substrates, including PCRAM [210], RRAM [110], and STT-MRAM [62], which use resistive memory devices. All are suitable candidates for cryogenic operation. More detail is provided in Section 3.3, but generally speaking SRAM and DRAM will stand to improve the most at cryogenic temperatures, relative to their room temperature performance. However, non-volatile PIM substrates have shown high performance and extreme energy efficiency [210, 298] at room temperature, which makes them even more promising candidates. Here, we discuss how cryogenic operation can improve the performance of PIM. As a case study, we evaluate an STT-MRAM based architecture at room temperature and cryogenic operation.

It is noteworthy that even if the performance benefit from cryogenic operation by itself is insufficient to justify the development of cryogenic architectures, such architectures will be required in order to support emerging technologies. Emerging cryogenic technologies include single flux quantum [240] and quantum computing [258], both of which will require classical hardware support [109].

3.2 PIM Substrates

Cell designs for different PIM-capable memory technologies are shown in Fig.3.1. Each technology follows similar operating semantics when it comes to memory access to perform reads and writes[1]. Each technology contains bitlines (BL), which are used to access the memory cell to perform operations. The cells are connected to the BLs via access transistors, which are controlled by rowlines (RL). A second bitline, bitline bar (BLB) is used in SRAM and NV technologies, which is set to the opposite value as BL during write operations. The memory storage device for each technology is different. SRAM uses a latch which is constructed with transistors, shown in Figure 3.1a. Due to a circuit feedback loop, there is a stable state when M1 and M4 are ON and M2 and M3 are OFF, and vice versa. To read a state, the access transistors (M5 and M6) connect Q to BL and Q' to BLB. If Q is 1 (0), the cell with pull BL (BLB) up and BLB (BL) down. To write, the same process is performed, except the voltage on the bitline is set by the bitline drivers, which will be strong enough to switch the state the of cell. Shown in Fig.3.1a is a 6T cell, however 8T is another common design. DRAM cells, shown in Figure 3.1b, use capacitors to hold the state. The presence of a charge indicates "1" and the absence indicates "0". The access transistor connects the capacitor to the bitline, allowing the charge on the capacitor to be read or to be overwritten. RRAM and STT-MRAM both use resistive memory devices, which are devices which can have varying levels of resistance. For

[1]Non-volatile devices are also commonly used in cross-bar architectures, which have significantly different characteristics. We omit their consideration as they are not random access memory.



(a) SRAM [8] (b) DRAM [325] (c) NV [210] (d) NV [301]

Figure 3.1: Representative PIM technologies considered.

example, a high resistance can be logic “1” and a low resistance be can logic “0”. For both devices, driving a current of sufficient magnitude will set the state. The direction of the current determines which state it is set to. A typical NV memory cell, such as used in [210], is shown in Fig.3.1c. A voltage across BL and BLB will drive a current through the resistive memory device.

The most common method of performing PIM in SRAM and DRAM uses the sense amplifiers and logic in the array periphery. Multiple rows are activated simultaneously, connecting multiple cells to the bitline. The sense amplifiers are then used to sense the voltage and differentiate between different inputs. Immediately after, basic logic circuitry operates on the output of the sense amps, and the result is written back into the array. If a NV PIM substrate uses the cell in Fig.3.1c, it must also use this same approach. A voltage applied on BL will drive a current through multiple resistive devices (in parallel) and onto the BLB, which can then be sensed. However, the NV cell in Fig.3.1d uses a double bitline, bitline even (BLE) and bitline odd (BLO), to enable computation in the array itself [301], bypassing the sense amps. This technology is called computational RAM (CRAM) [301, 62]. BLE connectes to even rows and BLO connects to odd rows (not shown). In this case, voltage is applied across BLE and BLO. Current flows through input memory cells (in parallel) which are in series with an output memory cell (connected over BLB)[301]. The current will set the state of the output cell, depending on the states of the input cells.

3.3 Adaptation to Cryogenic Temperatures

Cryogenic operation affects how the memory devices and supporting circuitry perform. In this section, we go over each of these changes and how they impact the performance of PIM. We highlight the differences between NV PIM and the more traditional PIM based on SRAM and DRAM.

3.3.1 Low level impact of Cryogenic Temperatures

Wire resistance and capacitance both decrease. Specifically, wire resistance is a linear function of temperature [200]. As this has the most impact on the bitline, which all technologies share, this provides a universal benefit to all PIM. For SRAM and DRAM, the benefit is reduced latency for the bitline pre-charge [325, 9, 8]. This will significantly reduce latency for both memory and logic operations, as wire latency dominates memory access time [200]. The main benefit for NV PIM is energy savings. When performing a read, write or logic operation, the wire resistance is in series with the resistance of the memory devices. A low bitline resistance will decrease unwanted energy dissipation. An additional benefit for NV PIM is increased reliability. This is because NV PIM uses resistive memory devices. To perform operations, a specified voltage needs to be applied across a connected set of resistive memory devices [301, 399]. Some of the applied voltage will drop over the bitline, which reduces the margin of error. Hence, cryogenic NV PIM will be more resilient to voltage fluctuations and process variation.

CMOS transistors perform better at 77K in a number of ways. The one drawback is that the threshold voltage increases slightly with decreasing temperature [29]. Otherwise, an increase in the charge carrier mobility results in a higher ON current [393], and both the transconductance and the sub-threshold slope are higher [329]. The steep sub-threshold slope drastically lowers leakage current. Logic built from transistors has a lower latency, roughly 30-40% [276]. These improvements benefit every aspect of SRAM and DRAM PIM. Logic performed by CMOS in the array periphery will be faster and more energy efficient. The reduction in leakage current will nearly eliminate static power for SRAM and reduce the refresh overhead for DRAM [377]. The benefits are less for NV PIM, which already has near zero leakage current and no refresh overhead. The main benefit for NV PIM will be the impact on the row-decoder, which is CMOS based. The row-decoder has considerable latency overhead, as it needs to be activated 1-3 times for every operation [298, 210]. Hence, superior transistor performance will have a lesser, but noticeable positive impact on NV PIM.

Resistive memory devices: apply only to NV PIM, and have a number of changes at cryogenic temperature, some positive and others negative. Magnetic Tunnel Junctions (MTJs) are widely used and are the basis of STT-MRAM. MTJs have a higher endurance at cryogenic temperature [196]. As noted in Section 3.2, resistive memory devices store logic values in their resistivity, having both a high and low resistance state. The resistance ratio is the relative difference of resistance of the two states. MTJs have a higher ratio at cryogenic temperatures [392, 397]. This increase can be quite significant, greater than 30% relative to room temperature in some cases [397]. A high ratio is desirable, as it makes them easier to discern during read operations. The high ratio also increases the robustness of logic operations, making them less susceptible to process variation and voltage fluctuations [301]. A negative impact is that the absolute resistance for both states increases with lower temperature as well [392, 397]. Different fabrication processes can be used to create MTJs with varying parameters. The increase in resistance can be different for different types of MTJs and can also be different for each state of the MTJ. The resistance can increase anywhere from approximately 10% up to 40% [397]. This is undesirable, as a higher resistance requires a higher voltage to perform the same write or logic operation [196], leading to more energy consumption. RRAM is an alternative resistive technology which has a significantly higher ratio than MTJs but also a lower endurance. RRAM will also function properly at 77K [365], however it will have a further reduced endurance [380]. For example, an endurance of 10^{10} write cycles at 298K was reduced to 10^{8} at 100K [163]. Additionally, it was demonstrated that they have a slightly higher operating

Table 3.1: Positive and negative effects of cryogenic operation on different technologies. + = increased, - = decreased, L=Latency, E=Energy, R=Resistance

Effect	DRAM	SRAM	MTJ	RRAM
Positive	(-)Bitline R (-)Peripheral L (-)CMOS Logic L (-)Refresh Overhead	(-)Bitline R (-)Peripheral L (-)CMOS Logic L (-)Static Power	(-)Bitline R (-)Peripheral L (+)Logic Robustness (+)Endurance	(-)Bitline R (-)Peripheral L (-)CMOS Logic L
Negative	Timing Changes	Timing Changes	(+)Write E	(-)Logic Robustness (-)Endurance

voltage and a narrower switching voltage window [365], making them more susceptible to voltage fluctuations.

3.3.2 System Level Impact and New Challenges

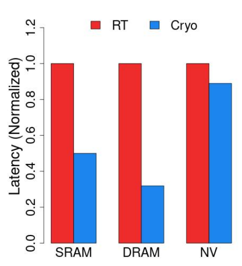
Overall, the impact of cryogenic conditions is generally positive for PIM technologies, as summarized in Table 3.1. That said, cryogenic operation also introduces correctness concerns which must be addressed.

SRAM and DRAM PIM: Both off the shelf DRAM [353, 377] and SRAM [250] have been demonstrated to work at cryogenic temperatures. However, these did not consider PIM. The change in relative transistor strengths and timing of the analog circuitry may affect the correctness of logic operations. The impact will depend heavily on the specific architecture, but we provide a few illustrative examples. An example of 8T SRAM PIM is X-SRAM [9], where logic is performed by pre-charging the read bitline and then connecting multiple cells to the read path simultaneously. Depending on the value stored in the SRAM cells, the voltage on the bitline decays at a known rate. At a specified time, a logic buffer reads the value of the read bitline and then writes it to the write bitline. The delay between the read and the write determines the type of logic that is implemented. Cryogenic temperatures change this timing. Compute Cache [8] uses a similar approach, but with 6T SRAM cells. 6T introduces the concern that SRAM cells may destructively interact, since the read and write paths share the same bitlines. This can be avoided by lowering the wordline voltage [171, 8] at room temperature. However, at cryogenic temperatures the transistors connecting the bitline to both the supply voltage and ground will be stronger, and this may increase the susceptibility. Ambit [325] uses a sense amplifier to read multiple DRAM cells simultaneously to differentiate between different input combinations. The amount of current that is drawn from each DRAM cell through the access transistor could be significantly different from that at room temperature.

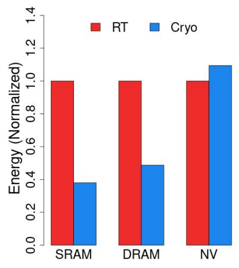
NV PIM: While NV PIM has some correctness concerns as well, these are easier to account for. The voltages applied to perform writes and logic operations change, due changes in the MTJ resistance, and the peripheral circuitry latency decreases. Changing the supply voltage and the frequency can account for this, no circuit re-design is required. The main disadvantages for cryogenic NV PIM are the non-ideal device characteristics, reduced energy efficiency for MTJs and reduced endurance for RRAM. A reduction in energy efficiency may be tolerable, given NV PIM demonstrates extreme energy efficiency at room temperature.

3.4 Quantitative Analysis

To determine the efficacy of cryogenic PIM we evaluate SRAM, DRAM, and NV-PIM both at 300K and at 77K. To estimate the performance of SRAM PIM we take data from Min et.al. [250] and for DRAM PIM we take data from Lee et.al [200]. These studies provide the latency and energy of memory accesses at 77K relative to 300K. We assume that a single memory access and logic operation have the same overhead. For NV PIM, we use MTJs as a representative case study, as they have a higher endurance at cryogenic temperatures. We take peripheral circuitry estimates from NVSIM [90] and match them with MTJ parameters from [399]. This provides the latency and energy for each PIM operation at 300K. To estimate performance at cryogenic temperatures, we



(a) Latency



(b) Energy

Figure 3.2: Benchmark characterization at room temperature (RT) vs. cryogenic temperature (Cryo). Values are normalized relative to the RT performance.

modify the peripheral circuitry latency and energy based on experimental data for CMOS operation [393], and MTJ latency and energy based on experimental data from [392, 397]. The peripheral circuitry and CMOS logic latency reduces by 35% and the MTJ write energy increases by 15% at 77K relative to 300K.

Machine-learning inference has been demonstrated efficiently in memory [172, 366, 301]. It is also ubiquitous in the cloud/server environment, making it a notable candidate for cryogenic acceleration. Hence, we use neural network inference as a case study with the CIFAR-10 image recognition dataset as the input. We take the network configurations provided by [211] and map them by hand to run on the PIM substrate.

We assume the PIM substrates consist of 1024x1024 memory arrays. Each memory array can perform logic operations within the columns, i.e., they have *column-level parallelism*. Such PIM architectures have been used with SRAM [9], DRAM [325], and NV technologies [210, 301]. Operations are driven by an external controller. We use a data layout similar to that used in PIMBALL [301]. Multiplications, additions, and subtractions are performed in a bit-serial manner within the columns of the memory arrays. Data is moved between columns and arrays with read and write operations (orchestrated by the external controller). All network parameters, including weights and thresholds, are stored in the memory prior to inference and kept constant. Input data (images) and the neurons of hidden layers are moved between memory arrays as needed throughout the run of the program.

To estimate the total latency and energy, we sum the latency and energy of all logic operations, reads, and writes required to perform the program. For every operation, we account for the overhead due to the peripheral circuitry and row decoder activation, which vary depending on the operations and the sequence they are performed in.

Results are shown for latency in Figure 4.4 and for energy in Figure 5.12. The energy consumption reported does not include cooling costs, which can be significant [50]. Cryogenic operation provides a latency advantage for all technologies due to improvements of the peripheral circuitry. Latency is reduced to 50% of the room temperature counterpart for SRAM, to 32% for DRAM, and to 89% for NV PIM, respectively. NV PIM's latency does not improve as much because the MTJ write latency remains the same at cryogenic temperatures. SRAM and DRAM PIM feature significantly reduced energy consumption, as well, down to 38% for SRAM and to 48% for DRAM PIM, respectively, of the room temperature counterparts. This is largely due to a reduction in leakage current, which enables a lower operating voltage. NV PIM actually is less efficient at cryogenic temperatures, with an energy consumption of 109% relative to the room temperature counterpart.

This is due to the increase in the MTJ write energy, which dominates energy consumption. Additionally, NV PIM already has near zero leakage current at room temperature, and hence this is not an added benefit at cryogenic temperatures. While the energy increase relative to room temperature is a detriment, it may be tolerable. At room temperature, NV PIM has demonstrated superior energy efficiency to more traditional architectures. For example, a Xeon E5-2640 CPU consumes $129\,$J to perform the CIFAR-10 benchmark [211], where an NV PIM solution only consumes $31.9\,\mu$J [301], superior even to specialized FPGAs consuming $299\,\mu$J [211]. Hence, a reasonable increase in its energy consumption will still result in an overall energy efficient operation. Note that MTJs optimized for room temperature were used in this analysis; MTJs designed specifically for cryogenic temperatures may display superior efficiency.

3.5 Conclusion

PIM technologies are well suited for the cryogenic domain. Their modular architectures facilitate ease of transition and their energy efficiency should enable lower cooling budgets. SRAM and DRAM based PIM show significant improvements in both performance and energy efficiency (when compared to their room temperature counterparts). NV PIM also exhibits increased performance but suffer from reduced energy efficiency when compared to its room temperature counterpart. This does not rule out NV technologies at cryogenic temperatures, however, as at room temperatures NV PIM tends to be typically much more energy efficient than SRAM or DRAM based PIM.

Chapter 4

Reliable Inference Under Extreme Operating Conditions

4.1 Introduction

In this Chapter, we expand the architecture presented in Chapter 2 to handle a wider temperature range and tolerate radiation. The goal of this is to enable such a device to operate as a satellite, where it serves a cheap alternative to more traditional space-based systems. We also increase the programmability of the architecture by extending the instruction set, and re-designing the architecture to be more efficient.

Beyond edge devices collect energy from the environment, allowing them to operate off the grid and without a battery [56, 180]. This enables them to function in environments that were previously considered as impossible, such as in the remote wilderness [241], within the human body [133], and out in space [225]. This capability opens up many opportunities for new applications. Running machine-learning algorithms on beyond edge devices is particularly attractive due its versatility [120]. Utilizing neural networks (NN) or support vector machines (SVM), a wide variety of problems can be solved.

However, engineering devices to operate beyond the edge is difficult. As they must collect energy from their environment, the power source by construction is unreliable. The devices must frequently power off, turning back on when energy is available. This is referred to as *intermittent* computing, which comes with performance and energy efficiency overheads. In order to prevent total loss of information during intermittent operation, beyond edge devices must do additional work in 3 categories [113]: 1) *Backup* refers to saving data and the current architectural state (which often entails writes to non-volatile memory); 2) *Dead* corresponds to re-performing work which couldn't be saved on the last shutdown; and 3) *Restore* encapsulates all work associated with re-starting the device after a shutdown. Beyond the additional latency and energy overheads, intermittency also makes it a challenge to guarantee correctness of a program. Power interuptions can introduce memory inconsistencies which can easily lead to incorrect operation [225, 69]. For conventional embedded systems, sophisticated software strategies are required to ensure that an interruption at any point during operation does not induce corruption [306, 120].

Previous work has shown that non-volatile processing-in-memory (PIM) architectures are promising for use in beyond edge devices. As a representative example, MOUSE [298] is a PIM architecture which delivers high performance and extreme energy efficiency using low complexity checkpointing mechanisms. It has 3 advantages over traditional architectures:

1. Inherently intermittent safe logic operations;
2. Automatic and instantaneous data backup;
3. Highly energy efficient and highly parallel operations.

Advantage (1) enables MOUSE to simplify its checkpointing strategies. Operations performed in the memory can be interrupted or performed multiple times without introducing corruption. Hence, data remains consistent as long as operations are performed sequentially. Advantage (2) comes from processing in non-volatile memory, and directly reduces the overhead for checkpointing. Typically a device has to write volatile data back to memory to save progress before shutdown. Since MOUSE does all its computation in non-volatile memory, progress is saved automatically after every operation. Finally, Advantage (3) enables high performance within low power budgets.

Recently, there has been much excitement about the use of beyond edge devices as nanosatellites [226] deployed in low earth orbit (LEO). Such devices can provide valuable services such as security along with agricultural [370], environmental or structural monitoring [226]. Nanosatellites can be much more cost effective than traditional monolithic satellites. However, orbital deployment for use as a satellite further challenges engineering such devices. For example, the cost of communication now becomes much greater than the cost of computation (even more so than for terrestrial deployment) [226, 120]. This shifts emphasis towards performing more computation and holding more data on the device, and away from frequent communication [86]. MOUSE is well-suited for this challenge as it has a large memory capacity (due to consisting nearly entirely of high density non-volatile memory), enabling it to potentially go long periods of time and store many results before data transmission becomes necessary.

An additional challenge for beyond edge devices deployed in LEO is that they must operate in a wide range of temperatures. Satellites can get both very cold (-170°C) and very hot (123°C) [201]. Large scale satellites can be engineered to perform temperature modulation [25]. However, small, cheap beyond edge devices can typically not use such strategies. Fortunately, CMOS can perform well across this wide temperature range, and the performance of CMOS circuits actually tends to increase with decreasing temperature [393, 329]. However, cold operation can have an adverse effect on non-volatile memory, where the energy efficiency degrades [296, 163, 196, 392, 397].

Another complication of orbital deployment is radiation. Even in terrestrial deployment, radiation can induce soft errors in CMOS circuits [26], potentially corrupting the architectural state. Without the earth's atmosphere to shield radiation from space, satellites are exposed to much higher levels of radiation. Non-volatile memory is at an advantage in this domain, as the memory devices it uses are highly resistant to radiation [255]. However, non-volatile memory still relies on CMOS circuitry for memory access and external control, which also applies to non-volatile PIM. Circuit level strategies can be used to mitigate the impact of radiation on CMOS hardware [317], which can incur significant power, latency, and area overheads.

In this work, we extend the design of MOUSE [298] to be suitable for orbital deployment. We demonstrate that MOUSE can operate over a wide temperature range (despite non-ideal impacts on non-volatile memory) and evaluate its performance at the extremes. MOUSE has an inherent resilience to radiation due to ideal properties of the non-volatile memory it uses [116, 186], but still requires CMOS circuitry to drive operations. We show that even in the presence of the overhead for the hardening of CMOS circuitry to radiation [406], MOUSE remains highly energy efficient and performant. We also extend the MOUSE PIM instruction set [298] and add architectural support for branch instructions, which increases the programmability of the device. Finally, we introduce more hardware efficient column activation mechanisms for enabling logic in the memory. The result is a programmable, high performance, and extremely energy efficient beyond edge device which is suitable for deployment in space. In summary, using MOUSE [298] as a representative case study, our contributions are as follows:

1. Demonstration that the non-volatile in-memory logic works under a wide operating temperature range and evaluation of the impact on performance.
2. Evaluation of the overhead in adapting all PIM circuitry to withstand high radiation.
3. Extension of the MOUSE instruction set for enhanced programmability.
4. Addition of more efficient hardware mechanisms for enabling logic in the memory.

4.2 Type of Operations Supported

Read: To read from row n, activate WLn. Apply a voltage differential, V_{read}, across LL and the BLE/BLO. Current can be sensed on the bitlines. V_{read} should be lower than V_{switch}.
Write: To write to row n, activate WLn. Apply a voltage differential, V_{write}, across LL and the BLE/BLO. To write 0 (1), the voltage on BLE/BLO should be higher (lower) than on LL. V_{write} should be larger than V_{switch}.

Logic Operation: To perform a logic gate with two inputs in rows n_1 and n_2, and the output in row m, preset row m by performing a write[1]. n_1 and n_2 must have the same parity (i.e., both even or both odd), and m, the opposite. Activate WLn_1, WLn_2 and WLm. Apply a voltage differential, V_{logic}, across BLE and BLO. Due to the parity requirement, in Figure 6.1, if V_1 is connected to BLE, V_2 must be connected to BLO, and vice versa. LL connects the free layers (in light blue) of the input and the output MTJs. Current travels from one bit line (either BLO or BLE, depending on the parity of the input cells), through the MTJs in rows n_1 and n_2, through the LL, through the MTJ in row m, and back to the other bitline. Depending on the states of the MTJs in rows n_1 and n_2, the state of the MTJ in row m will either change or not. Figure 2.3 shows how a NAND gate can be performed inside the array. V_{logic} must be within a specified range for each type of logic operation [301, 400].

Only one operation (read, write, or logic) can be performed in each column at a time. However, operations can proceed in many columns simultaneously. The restriction is that (within a single array) it should be the same operation (i.e., type of logic gate) on the same row designation for inputs and outputs. For example, a NAND gate can be performed in all columns with the inputs in rows n_1 and n_2 and the output in row m.

[1]This write needs only to be performed if the initial state is different than the corresponding preset value.

It may be desirable to perform computation in all columns, or in just a subset of columns. The peripheral circuitry determines which columns participate in every operation. The mechanism for activating columns is covered in Section 4.3.3, and the instructions which control column activation, in Section 4.3.4.

Effectively, each column acts as an independent thread that has access to the memory cells within the column. This is highly analogous to the SIMD lanes of a GPU architecture, where each lane (column) performs the same operation on different data. The *active* columns act like the bitmask in a GPU, where only the active subset columns perform the operation. However, each column can only perform Boolean logic gates (whereas a GPU has full access to an ALU). Complex arithmetic/logic translates into performing a sequence of Boolean gates in each column, where each gate operation can proceed in parallel, in a lock-step fashion. Hence, computations in each column are relatively slow, but performance is achieved via a high degree of parallelism.

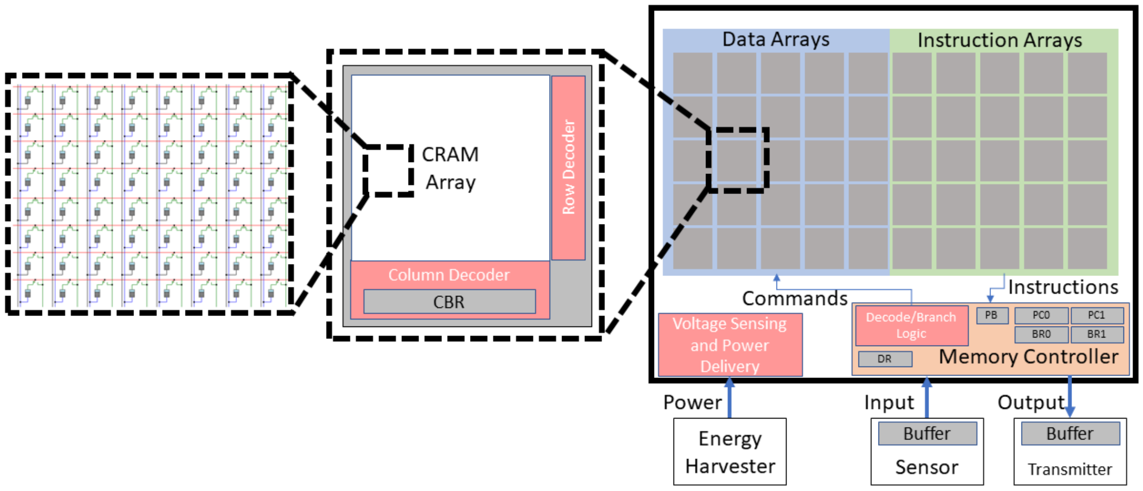


Figure 4.1: Overview of modified MOUSE. Each memory array contains an array of MTJs, a row and column decoder, and a non-volatile register storing the column bitmask. Sense amplifiers are required for reads but are not used in computation. The memory controller contains non-volatile registers to maintain the architectural state.

4.3 Revised MOUSE Design

In this section, we describe the architecture of MOUSE and show how it is uniquely well suited for beyond edge deployment. MOUSE utilizes CRAM arrays and minimal support circuitry. The computations performed in MOUSE are energy efficient, highly parallel, and have an inherent robustness to intermittent operation. The basic architecture and operation semantics are the same as our previous work [298]. However, we expand the instruction set, improve the method of activating memory to perform computation, and add hardware support for branch instructions.

4.3.1 Hardware Organization

The hardware remains highly similar to MOUSE with a few modifications. Most notably is additional hardware in the memory controller and additional registers in each CRAM array. Figure 4.1 shows the architecture of MOUSE. It consists predominantly of CRAM arrays.[2] MOUSE can afford to have a large number of arrays (and hence more memory than is typical for a beyond edge device) due to the ideal properties of MRAM. Each CRAM array contains 1024 rows and 1024 columns. In addition to the arrays, MOUSE requires the following minimal hardware to drive operations and maintain the architectural state:

1) A memory controller that reads, decodes, and issues instructions;

2) A non-volatile register for the Program Counter (PC);

3) A 128 byte data register (DR) that facilitates reads and writes;

4) Two non-volatile registers, BR1 and BR2, for branch evaluation;

5) Voltage sensing circuitry for monitoring the power source.

Minimal hardware is required for the memory controller.[3] With the exception of resolving branches (covered in Section 4.3.4) and updating architectural variables, its sole responsibility is repeatedly reading instructions, decoding them, and broadcasting them to the CRAM arrays. We use a highly simplified instruction set, covered in Section 4.3.4, hence decoding requires very little computation. The DR is the same size as one row of the MOUSE arrays and is used for intermediate storage when transferring data to and from different arrays. BR1 and BR2 hold data near the memory controller, enabling quick comparison tests for branch resolution. Finally, the voltage sensing circuitry is standard in beyond edge devices and is as described in [219].

4.3.2 Row Activation

In standard memory, a row decoder activates wordlines for read and write operations. As depicted in Figure 2.3, logic operations require the activation of multiple rows (up to 3) simultaneously. To avoid increasing complexity of the row decoder, we use a latching mechanism which holds wordlines high after a row activation [209]. In this manner, the row decoder can activate rows sequentially with normal operation. The hardware cost is two transistors per row. Additionally, each logic operation must wait for the three sequential activations, which increases latency.

4.3.3 One-Hot Column Decoder

Typically it is desirable to drive logic operations in every column simultaneously. However, it is frequently preferable to perform computation only in a subset of the columns, leaving data in other columns unperturbed. Hence, in addition to a row decoder, we also need a column decoder which will select which columns participate in each operation.

Column activation patterns are different than for rows. With rows, 1-3 rows are activated for every instruction, and the rows which are activated are typically different for each consecutive instruction. When it comes to column activation, typically many columns are activated simultaneously (commonly all columns or a large subset). Additionally, columns tend to remain active for long periods of time – (de)activating columns is a rare event.

[2]Each array also contains a row decoder and column decoder.

[3]Our memory controller is a standard memory controller which has been augmented with the capability to read, decode, and issue PIM instructions. We simply refer to it as the memory controller for the remainder of the paper.

The original MOUSE design relies on a decoder which allows bulk addressing [298, 326]. Activation for the column decoder follows the same semantics as for the row decoder, except that there are reserved addresses which correspond to groups of columns, and up to 5 column addresses can be specified simultaneously [298]. In this work, we propose One-Hot bitmask decoding to reduce the complexity of the decoder.

Rather than an address, a bitmask is supplied to the column decoder. In a 1:1 bit to column scheme, each bit indicates whether a specific column should be activated or not. The bitmask is stored in a 1024-bit non-volatile register (corresponding to the 1024 columns in each array) within each CRAM array. We call this register the *column bitmask register* (CBR). The CBR can be written with a standard write operation. The advantage of One-Hot bitmask activation is that the column decoder complexity is low: no addresses need to be resolved. The activation signal of each bit can be supplied directly to the columns. The disadvantage is that each activation of the columns (if the column addresses are changing) must be preceded by a write operation to CBR.

It is possible to use different bit-to-column ratios. Fore example, a 1:32 scheme could be used, where each bit activates a consecutive set of 32 columns. This would allow a 32-bit mask to activate all 1024 columns. The drawback is reduced flexibility as such a bitmask cannot activate less than 32 columns at a time. If computation requires less than 32 columns, additional (and unnecessary) operations will be performed in all 32 columns. This wastes energy. As a 1024-bit bitmask is easily handled by the CBR, we maintain a 1:1 ratio.

4.3.4 Instructions

Instructions for MOUSE are 64-bit and have the formats shown in Figure 4.2. The expanded instruction set has four categories of instructions, which we explain here.

Memory Instructions

Reads and writes are the standard memory operations, but have additional overhead due to support for intermittent operation. The data register (DR) is a non-volatile register the same size as the rows of the CRAM arrays (128B) that holds data between read and write instructions. A read instruction reads from a CRAM array (at the specified address) and writes the data into the DR. A write instruction reads data from the DR and writes it into a CRAM array (at the specified address). Hence, if there is a power interruption between consecutive reads and writes, the DR will maintain the data being transferred – circumventing the need to re-perform the prior read operation. In addition to memory operations which use the DR, there is also a write immediate instruction, which allows instructions to write data directly into memory.

Logic Instructions

Logic instructions correspond 1:1 with logic gates (as covered in Section 2.2.2). For example, NOT, (N)AND, and (N)OR are all individual instructions. The instruction specifies the CRAM array address the operation is to be performed in and the row addresses of the logic gate (which rows the inputs and outputs reside in). NOT requires two row addresses (1 input, 1 output) and all others require three row addresses (2 inputs, 1 output). For example, a NAND instruction may specify that it is to be performed in CRAM array 15, with inputs in rows 7 and 9, and the output in row 12. We restrict logic operations to at most two inputs, which are shown to be reliable [301, 400]. As COPY, XOR, and XNOR gates are not natively supported in CRAM, there are only 5 unique logic instructions. Analogous to vector instructions, many logic gates can be performed in parallel (triggered by a single instruction), as long as their inputs and output reside in the same rows. The number of parallel gates depends on which columns are active, discussed in Section 4.3.4.

The CRAM array address can specify a single array, or multiple arrays with bulk addressing [326]. There are reserved memory addresses which correspond to groups of memory arrays. For example, it may be desirable to trigger an operation in all CRAM arrays. We designate array address 111111111 as a reserved address, to send an instruction to all arrays.

Activate Columns Instruction

It is necessary to specify which columns should participate in each operation. As noted in Section 4.3.3, consecutive operations typically use the same columns. Hence, which columns to activate changes infrequently. To take advantage of this, we use a strategy where columns are activated and then held active. All following logic operations will be performed in the columns which are held active. To (de)activate columns we use a dedicated instruction, the activate columns (AC) instruction. As described in Section 4.3.3, the column decoder simply activates the columns depending on the values in CBR. Hence, a column activation consists of two components:

1. A write to the CBR (*set*);
2. Triggering of the column decoder to activate the corresponding columns (*activate*).

Typically an AC instruction handles both components. However, when restarting the device it is only necessary to do the second. Hence there are two variants of the AC instruction, one which does both parts (set and activate) and one which only does the second (activate).

The write to CBR acts like a standard write. As noted in Section 4.3.4, a write can use the value in the DR or an immediate field in the instruction. Hence, there are a total of 3 unique versions of the AC instruction:

1. Re-activate: Activate using pre-existing value in CBR;
2. Set and Activate: Use data in DR to set CBR and then activate;
3. Set and Activate (Immediate): Use data in the immediate field of instruction to set CBR and then activate.

Branch Instructions

Branch instructions involve an update to the program counter (PC) in the event a logical condition holds true. As the logic required to evaluate the condition (e.g., checking equality of two numbers) is not complex, it can be implemented efficiently within the memory controller.

Non-volatile registers, BR1 and BR2, reside in the memory controller and are used for condition evaluation. We support simple standard branches based on BR1 and BR2:

1. beq BR1 BR2: branch if BR1 and BR2 contents are equal;

2. bge BR1 BR2: branch if BR1 is equal or greater than BR2;

3. beqz BR1: if BR1 equal to 0.

Hence, the memory controller evaluates a simple condition based on BR1 and/or BR2 and updates the PC accordingly. Additional instructions are required to write values to BR1 and BR2. This follows the same semantics as writing to the CBR. A dedicated instruction writes to BR1 or BR2, and the value can come either from the DR or an immediate field in the instruction.

Branch instructions increase programmability by enabling function calls, repetitions of computational blocks, and handling I/O events. However, as the computation for branch instructions happens in the memory controller, it cannot capitalize on the extreme energy efficiency and the high degree of parallelism provided by the CRAM arrays. Therefore, efficiency tends to decrease with larger share of branch instructions in the instruction mix. [4]

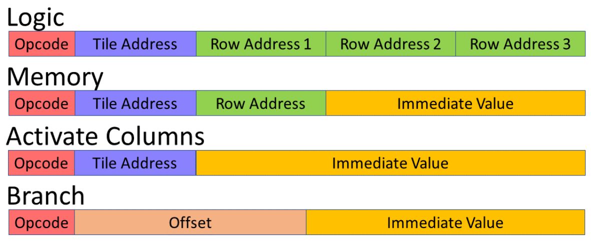


Figure 4.2: MOUSE instruction formats. There are three types of instructions, logic, memory, and an additional activate columns instruction for configuration. Opcodes are 5 bits; array (tile) addresses, 9 bits; and row addresses 10 bits each. Branch offset is 20 bits. Remaining bits are used by instructions which use an optional immediate value.

Compilation

Compiling high-level code to MOUSE instructions (or any PIM substrate) requires knowledge of the PIM hardware in order to make efficient use of available parallelism. This is similar to compiling Open-CL or CUDA code for GPU architectures. Unfortunately, no equivalent standard software compiler exists for PIM. There is a rich design space, where a multi-dimensional trade-off exists between efficiency, area, power, and performance. Higher degrees of parallelism are possible by spreading computation out over more columns. However, this increases power, consumes more area, and adds communication overhead which reduces energy efficiency. Our strategy rather is to minimize area by using as few columns as possible, to maximize energy efficiency. Our data layout is similar to a number of other works which have mapped applications to PIM substrates [209, 326], including machine learning algorithms [301].

Issuing Instructions

While operations can occur in multiple arrays simultaneously, arrays do not operate autonomously. All operations are triggered by the memory controller (discussed in more detail in Section 4.4). Effectively, there is a single controlling "thread", and hence there are no concurrency concerns between individual arrays. CRAM arrays in MOUSE hold both data and the instructions. For clarity,

[4] Avoiding branch instructions is easy for machine learning applications. For our benchmarks, we do not need any branch instructions during a single inference pass. Branches are used only to repeat inference or to handle I/O.

we categorize arrays into *instruction arrays* and *data arrays* based on their contents. However, as all arrays have identical hardware, arrays can be re-categorized to fit the programmer's needs.

Instructions and required data are written into the arrays prior to deployment. During operation, the memory controller repeatedly fetches an instruction from the instruction arrays, decodes it, and then broadcasts it to the data arrays. Instructions are performed entirely sequentially. The next instruction does not start until the previous has finished. This is to guarantee correctness, which we will cover further in Section 2.4.4 and Section 4.4.

Generally, different instructions can take different time to complete. This is mainly because instructions can activate different numbers of rows. To guarantee that all instructions complete in time, for each instruction, the memory controller conservatively allocates as much time as the the longest instruction takes before starting the next instruction. This time lapse forms a *cycle*. This conservative approach to issuing instructions comes with a performance cost, as opposed to a more complex event-driven strategy. We opt for the conservative approach for three reasons: (1) MOUSE already delivers higher performance than representative alternatives for beyond edge computing (as we cover in Section 7.5), hence aggressive optimization is unnecessary. (2) Complex issue logic would be less energy efficient and make it more difficult to guarantee correctness under intermittent operation. (3) Simplicity is a strength for beyond edge devices. In the end, energy efficiency (rather than high performance hardware) is the limiting factor for performance beyond the edge [121]. Designs consuming less energy complete programs faster because they spend less time waiting for the harvested energy to reach sufficient levels for forward progress in computation.

4.4 Intermittent Correctness Guarantee

Beyond edge devices need to ensure program correctness in the presence of power outages. If not handled carefully, interruption due to power outage can corrupt the architectural state. In this section we show how MOUSE remains correct, even in the presence of unexpected power outages. There are two crucial components: the correctness of individual in-memory operations when interrupted or re-performed (Section 2.5.1) and the correctness of architectural state variables in transitions between states (Section 4.4.2). As MOUSE checkpoints after every instruction, we need to only show that each instruction and the transitions between instructions remain correct when interrupted. In the following, we show that all instructions and transitions are *idempotent* [369, 157], which means that they produce the same results, even if repeated multiple times. The key to remaining idempotent is not over-writing data that is required on restart (or if the data gets overwritten, it should be in a manner that does not change the computation outcome). The architectural state variables and their protection mechanisms are listed in Table 4.1. Note that the correctness guarantee covered in this section applies only to interruptions and power outages. It does not cover errors in the computation itself or perturbations due to soft errors from radiation.

4.4.1 Operation Level Correctness

In this section we cover the correctness of individual operations performed in the memory when interrupted and re-performed. We assume the most general case, where the power can be cut at any moment (unexpectedly). Hence, we need to consider all possibilities for when (during its processing) an operation can get interrupted.

Table 4.1: Architectural state variables and how they are protected under power interruptions.

Variable	Volatility	Protection Mechanism
Program Counter	Non-Volatile	Duplicated, valid copy is read only
Parity Bit	Non-Volatile	Only flipped after instruction has finished. Flip is an atomic operation
BR1 and BR2	Non-Volatile	Write operation guarantee (Section 4.4.1)
CBR	Non-Volatile	Write operation guarantee (Section 4.4.1)
DR	Non-Volatile	Read and Write operation guarantee (Section 4.4.1)
Active Columns	Volatile	Bitmask stored in CBR. Re-actived on restart with AC instruction (Section 4.3.4)
Active Rows	Volatile	Activated by every instruction
Data	Non-Volatile	Idempotent logic operations (Section 2.5.1), Read and Write operation guarantee (Section 4.4.1)

Logic Operations

Logic operation correctness is identical to MOUSE, and is exactly as described in Section 2.5.1. There are no changes introduced in this chapter.

Memory Operations

Re-performing a read operation has no effect on the read data, reading it a second time will produce the same results. Re-performing a write will overwrite whatever was written the first time. If the write was not successful the first time (due to interruption), it will be successful the second time. If the write was successful the first time, the same value will be written twice. As noted in Section 4.3.4, read (write) instructions can involve a write to (read from) the *DR*. These are protected by the idempotency of both read and write operations - a memory operation does not write to any address/register that it also reads.

Column Activation

Column activation involves a write to the CBR in a data array and then a triggering of the column decoder. The write to the CBR is kept correct by the same semantics as memory operations (a write can be performed multiple times). The column activation by the column decoder does not change any non-volatile data and hence cannot introduce corruption. The volatile state is entirely lost on shutdown and will be overwritten on restart.

Summary

Power interruptions can waste energy (due to re-performing instructions) but cannot corrupt the data in memory. Idempotency of all instructions guarantees that they produce the same results, even if performed multiple times. Moreoever, idempotency is not required beyond a single instruction as only one instruction is performed between each checkpoint.

4.4.2 Maintaining Correct State

The previous section showed that the individual operations performed in the memory are idempotent. This is necessary but not sufficient for correctness. MOUSE has to guarantee that the memory controller can drive the operations and maintain the architectural state in an intermittent safe fashion. This extended architecture of MOUSE has more instructions and more considerations due to radiation.

Memory Controller

The memory controller repeatedly reads instructions from the *instruction arrays*, decodes them, and broadcasts them to the *data arrays*. The memory controller waits for a sufficient time for each instruction to complete before updating the program counter (PC). The PC must be stored in a non-volatile register to prevent loss on shutdown. However, concern remains if the update to the PC gets interrupted. If power is lost during a write to the PC register, it can be corrupted - resulting in incorrect behavior on startup.

We circumvent this issue by maintaining two PC registers, PC0 and PC1, and an additional non-volatile parity bit. If the parity bit is 0, PC0 is valid, and if the parity bit is 1, PC1 is valid. When the memory controller updates the PC, it takes the value in the valid PC register, updates it accordingly, and stores it into the invalid PC register. Then it flips the parity bit, indicating the advancement to the next instruction. Therefore, the memory controller never writes to the valid PC register, and there is no risk of corruption.

The setting of the parity bit is analogous to the committing of an instruction in traditional architectures. As the parity is a single bit, the operation is *atomic* and cannot be interrupted mid-way through. The parity bit either is set or not. If an interruption occurs before the parity bit is set, the memory controller re-issues the same instruction on restart, which is safe to do (Section 4.4.2). If the interruption occurs after the parity bit is set, the instruction is completed and the memory controller issues the next instruction on restart, as depicted in Figure 2.7.

There are other non-volatile registers that hold the architectural state. These include the data register (DR), branch registers (BR1 and BR2), and the column bitmask registers (CBR) in each array. These registers are protected by the same semantics as in Section 4.4.1. Updates (i.e., writes) to these registers are guaranteed to be completed before the memory controller commits the corresponding instruction. No register is both read and written by the same instruction, so no required data can be corrupted.

Active columns is part of the architectural state. When MOUSE restarts, these columns need to be re-activated. The non-volatile CBR in each memory array maintains the currently valid bitmask for active columns. All that is required is for the column decoders to re-activate these columns. To achieve this, the memory controller issues a re-activate columns instruction to all arrays on restart (Section 4.3.4).

Data in Arrays

The previous section showed that the memory controller itself remains correct during intermittent operation. We must also ensure that the memory controller does not generate any signals which corrupt the data residing in the memory arrays.

The memory controller broadcasts instructions to the data arrays. This broadcast is not atomic, and thus can be interrupted at any stage. However, all the operations that it can trigger are idempotent (Section 2.5.1), meaning they can safely be interrupted at any point in their progression. As a direct result, the broadcast cannot cause corruption as its only effect is the initiation of the operation. Power can be cut before the broadcast reaches a memory array, while the operation is being performed, or after the operation has finished –none of these cases can introduce error.

4.5 Impact of Orbital Deployment

An exciting domain for beyond edge devices is low earth orbit (LEO) where they can act as nano-satellites [226]. One aspect of LEO deployment is that the cost of communication becomes much greater than computation (even more so than for terrestrial deployment) [226, 120]. MOUSE is especially well suited for LEO deployment as it has a much larger memory capacity relative to other beyond edge devices as MOUSE nearly entirely entails high density non-volatile memory. Due to the large memory capacity, MOUSE can go long periods of time without offloading data. However, orbital deployment also introduces challenges related to operating temperature and radiation. We discuss here how MOUSE can tolerate such conditions.

4.5.1 Temperature

Satellites in LEO can experience a wide range of temperatures, from -170°C to 123°C [201]. Maintaining the proper temperature on large scale satellites is an important engineering challenge [25]. Nano-satellites, on the other hand, typically do not have sufficient resources for any temperature modulation yet they need to operate properly across a wide range of temperatures. Non-volatile memory characteristics are very sensitive to temperature [296], which further challenges this situation for MOUSE.

MTJ resistance increases with decreasing temperature, to the extent that the resistance at -170°C can be 30% higher than at room temperature [393, 196]. This increases the voltage required to write MTJs, and consequently, energy consumption. The SHE architecture is less sensitive than STT, as the SHE channel is metallic (to be more specific, the resistance does not increase with decreasing temperature). Therefore, write operations with SHE remain largely unaffected. However, SHE still requires current to travel through the body of input MTJs for read and logic operations. Hence, the overall energy efficiency of SHE still decreases. As a result, PIM using MTJs (or other non-volatile technologies, as well) is less energy efficient at cold temperatures [296]. However, the change is modest, within approximately 10% more energy consumption (relative to room temperature), even at cryogenic (77K) temperatures [296]. Given that MTJs are extremely energy efficient [398, 401], this increase in energy consumption remains tolerable. Additionally, there is a benefit of cold temperature. The ratio between the high and low resistance state increases [397, 196]. This leads to more robust logic gates which are less susceptible to voltage fluctuations [400, 301].

The inverse is true at high temperatures. The overall MTJ resistance and the ratio between the high and low resistance states are both lower. MTJ resistance at 123°C is roughly 86% of its resistance at room temperature. This translates into more energy efficient MTJ logic gates, which at the same time are more susceptible to voltage fluctuations. The latter challenges the power delivery system. Power systems such as Capybara [71] become necessary to ensure that the proper voltage is applied across a variety of temperatures. Switched-capacitor voltage converters [173, 289, 289] can generate necessary voltages to facilitate correct operation. We cover the overhead of voltage generation/conversion in Section 7.5.

In contrast to the resistive memory, the peripheral circuitry in MOUSE benefits from cold temperatures. At colder temperatures CMOS transistors have higher ON current [393], switch faster [276], feature a higher trans-conductance, and incur a lower leakage current due to a steeper subthresh-

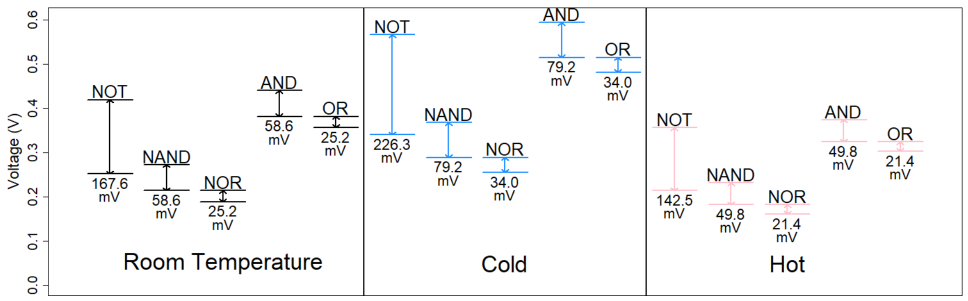


Figure 4.3: Voltage ranges for correct operation of each logic gate at different temperatures, room temperature (27°C), Cold (-170°C), and hot (123°C).

old slope [329]. However, MOUSE does not benefit significantly from these characteristics. This is because the non-volatile memory already has extremely low static power, and the latency is limited by the switching time of the MTJs. CMOS performance can degrade with increasing temperature due to increased leakage current, while typical CMOS devices can operate well up to 175°C [176]. Radiation hardened bulk CMOS technology can increase this range further to 250°C [224, 176]. Hence, CMOS technology is well suited to operate within the expected temperature range of LEO satellites. We discuss the overhead of our CMOS components further in Section 7.5.

Voltage Margins

Logic operations with MTJs discussed in Section 2.2.2 require proper voltages to be applied across the inputs and output. Correct operation can only be the case if each gate-specific voltage is within a specified range [398, 301]. The required voltage depends on the logic operation (which determines the number of inputs and the output preset), the resistances of the MTJs (R_P and R_{AP}), and the switching current (I_{switch}). Because the MTJ resistance changes as a function of the temperature, the proper voltage ranges do, as well. Figure 4.3 shows the voltage ranges for different logic operations at different temperatures. MTJ resistance is higher at cold temperatures, making the required voltages higher. At high temperatures, the MTJ resistance is lower. In addition to the voltages being lower, the ranges are also smaller. This reduces the margin for error in the voltage supply. MOUSE relies on a power delivery system which can reliably supply the appropriate voltage. This becomes an even more challenging task considering temperature fluctuations.

4.5.2 Radiation

Radiation can cause errors in electrical circuits. When a stray energetic particle strikes the hardware, it induces a voltage spike which can travel along the circuit and potentially alter output voltages or flip logical values [317]. Corrupted logic values give rise to *soft errors*. Technology scaling already makes any circuit ever more susceptible to soft errors[391]. However, when deployed in orbit, beyond edge devices are exposed to significantly higher levels of radiation which translates into frequent particle strikes.

CMOS components of MOUSE (the memory controller and peripheral circuitry, respectively) are susceptible to soft errors. Radiation can disrupt the CMOS logic in the memory controller or the voltage supplied by the power delivery circuitry (required to drive logic operations in memory). The consequences vary greatly depending on the location. The impact of soft errors in computation will likely be minor due to the resilience of machine learning to noise. Hirtzlin et. al. [159] showed that frequent bitflips in binary neural networks implemented in STT-MRAM result in negligible degradations in accuracy. However, a bit flip in the memory controller (corrupting the architectural state) can lead to undefined behavior. For example, if the memory controller experiences an error where it incorrectly updates the program counter, the device could attempt to read instructions from invalid addresses. Such an error could permanently disable the device. Hence, hardening MOUSE to radiation is imperative.

Many mitigation techniques exist for soft errors, which can be categorized into-system level, device-level, or circuit-level [317, 319, 316]. As MOUSE must stay correct in transitions from each instruction to the next (due to check-pointing after each instruction), and as increasing complexity also increases the overhead for correctness guarantees, system-level mitigation is less appropriate – i.e., soft errors should be caught before they manifest themselves at the system (architecture) level. Additionally, MOUSE relies on pre-existing CMOS devices, making device-level mitigation impossible.[5] Hence, circuit-level mitigation is the most suitable approach.

MOUSE relies on switched-capacitor circuits to supply appropriate voltages to the memory, which can be particularly susceptible to radiation. Circuit-level techniques can be especially helpful in this case, e.g., in the form of additional feedback paths to counteract the impact of particle strikes [106]. Thereby, if a single path experiences a voltage spike (or drop), an alternate path can take over in order to compensate. When properly designed, the impact on the final output can be minimized even if a large disturbance is experienced at the input of a circuit. This comes at a cost in area and energy efficiency due to the larger number of transistors per circuit.

MOUSE also relies on CMOS logic circuits within the memory controller to decode instructions and send commands to the memory. CMOS logic can be hardened to soft errors with a variety of circuit-level techniques. Redundancy can be added, either in space [270] (with area overhead) or time [244, 263] (with latency overhead), where outputs are checked for consistency. Increasing the node capacitance and transistor drive current (at a cost of area and energy) can also reduce the electrical susceptibility to particle strikes [406]. More sophisticated strategies of lower area overhead involve creating "transmission gates" between stages of a circuit, filtering out pulses from particle strikes [194, 405, 318].

While MOUSE does critically rely on CMOS circuitry, the vast majority of MOUSE's computation and all of its memory involve MTJs. Fortunately, MTJs are considerably more robust to soft errors than alternative technologies While short-lived voltage pulses suffice to change the state of CMOS circuits, MTJs require a significant current (a few microamps) for a sustained period of time (a few nanoseconds) in order to switch states. Hence, particle strikes are highly unlikely to flip MTJ states. In fact, MTJs are shown to be highly resilient to radiation from heavy ions [74, 187], neutrons [293], protons [164], and gamma rays [293, 164]. Recently, Montoya et. al. [255] demonstrated that MTJs are even resilient to radiation that is $100\times$ greater than what is observed on particularly harsh interplanetary travel [402, 27]. Therefore, MTJs represent leading candidates for space applications [116, 186]. Since MOUSE consists mostly of MTJs, it is less susceptible to radiation than traditional architectures. As only minimal CMOS circuitry is required external to the memory arrays, circuit level strategies to increase CMOS resilience to radiation [406] incurs a relatively low overhead.

[5]A major exception is pre-existing properties of MTJ devices, discussed later.

4.6 Evaluation Setup

Benchmarks: The exact use case of beyond edge devices can vary significantly, applications include agricultural monitoring [226, 370], security, and structural and environmental monitoring [86]. However, general sensor processing algorithms can be used to solve a wide variety of problems. We use benchmarks which are representative for many possible use cases - machine learning inference on data sets which are tenable for beyond edge devices. The specific input problem will vary depending on the user, however, the computation involved should remain highly similar.

We implement two machine learning algorithms, Support Vector Machines (SVM) and Binary Neural Networks (BNN). Both are widely used and light weight, which makes them highly suitable for the beyond edge domain. For both, we used only operations that are efficient in MOUSE, all bit-wise and integer arithmetic. Machine learning applications are well-suited for integer arithmetic as they remain robust under approximation. Fixed-point representation using integer arithmetic is sufficient to achieve high accuracy [165]. We designed customized SVM implementations and trained and tested them in R [288]. We were able to achieve similar accuracy as standard SVM implementations from libSVM [57]. For inference, the main computation is effectively performing the dot product between an input vector and each of the support vectors. The results of these dot products are then squared, multiplied by a set of coefficients, and finally summed together. By construction, SVMs have two class outputs, where the sign of the output value is the classification. We extend to multi-class classification by training a separate SVM for each possible output class, where each has the task of identifying a single class. BNNs are neural networks that use neurons and weights represented by a single bit each [76]. This enables multiplications to be replaced by XNOR operations and addition is simplified to a popcount operation. This gives BNNs extreme energy efficiency. Previous work has efficiently mapped BNNs onto FPGAs, including FINN [362] and FP-BNN [211]. We copy their network configurations exactly. We modify the algorithms only in transforming them to run on our PIM substrate. Hence, our accuracy is identical.

Data Sets: For small scale image recognition we use MNIST [198]. The task is digit recognition, where a 28×28 pixel image with 8-bit precision is to be classified into one of ten digits (0-9). We use both BNNs and SVMs on this benchmark. With SVM, the pixels are a 784 element vector. We also create a binarized version, where pixels that have a value below $255/4 \approx 63$ are assigned 0 and those above are assigned 1. This allows us to replace multiplications with AND gates, significantly reducing the time, energy, and area overhead. For BNNs, we use the network configurations of FPGA-based FINN [362] and FP-BNN [211]. FINN [362] uses binarized input, has three hidden layers of 1024 neurons (bits) each, and the output layer has 10 neurons with 10-bit precision. FP-BNN [211] 8-bit inputs, has three hidden layers of 2048 neurons each, and the output layer is 10 neurons with 16-bit precision.

Human Activity Recognition (HAR) [14] is a data set which has accelerometer and gyroscope measurements from a smartphone, which is carried by participants performing a variety of activities. The problem is to classify the physical activity the individual is performing. Each input is a vector of 561 elements. We convert the input to fixed point representation with 8-bit precision.

ADULT [188] contains census information. The problem is to classify whether an individual makes greater than $50K per year or not. We use a reformatted version of the data set from libSVM [57]. Each input is a 15 element vector where each element is an 8-bit integer.

Table 4.2: Parameters for MTJ devices.

Parameter	Modern	Projected
P State Resistance	$3.15\,\text{k}\Omega$	$7.34\,\text{k}\Omega$
AP State Resistance	$7.34\,\text{k}\Omega$	$76.39\,\text{k}\Omega$
Switching Time	3 ns [307, 268]	1 ns [398, 167]
Switching Current	$40\,\mu$A [307]	$3\,\mu$A [398]

Performance and Energy Model: We use an in-house simulator to evaluate MOUSE. We set each array in MOUSE to 1024x1024, which is a recommended subarray size for non-volatile memories from NVSIM [92]. Our simulator tracks all instructions issued by the memory controller and accounts for the time and energy consumed by each. An instruction can consume energy by performing the following actions:

1. Reading the instruction from the instruction array.
2. Sending the instruction to the data arrays.
3. The activation of rows for computation.
4. The activation of columns for computation.
5. The switching energy of the MTJs in memory.
6. Update of the program counter and parity bit

Items 1 and 6 always occur, while the remaining items occur depending on the instruction type. All energy consumption comes either from the MTJ devices or from the peripheral circuitry. The models for both are discussed below.

We simulate with both modern MTJ parameters [308] and with projections of MTJ parameters expected to be possible within a few years [398, 401]. MTJs are expected to be significantly more energy efficient as the technology matures. Two techniques will enable a reduction in the switching current, 1) decreasing the damping constant of ferromagnetic materials [315, 252, 97] and 2) using a dual-reference layer structure [161, 89]. It is possible switching currents will be as low as $1\,\mu$A, however, we assume $3\,\mu$A to be conservative. The parameters we use are shown in Table 7.1. For Modern MTJs we use only the STT architecture, for projected MTJs we use both the STT and SHE architectures. The benefit of SHE is providing a more efficient write mechanism. We model the SHE channel as a $1\,\text{k}\Omega$ resistance. This provides a conservative estimate of SHE energy efficiency.

Due to the different switching times of modern and projected MTJs, we clock MOUSE at different speeds for each. With Modern MTJs MOUSE operates at $30.3\,\text{MHz}$ clock rate (33ns per cycle) and for projected MTJs MOUSE operates at $90.9\,\text{MHz}$ clock rate (11ns per cycle). This enables sufficient time for instruction read, decode, and the peripheral circuitry latency and MTJ switching time.

For modeling peripheral circuitry, we take data from NVSIM [92] which reports the relative overhead of peripheral circuitry for modern MRAM memory. We set the overhead of MOUSE so that it consumes the same relative share of total latency and energy as reported by NVSIM.

We first evaluate MOUSE with continuous power (using a power source which can supply as much power as MOUSE desires). Then, we evaluate with an energy harvesting power source where MOUSE will have to operate intermittently. We model the energy harvester as a (small) constant power source which is filling an energy buffer (capacitor). When MOUSE is off, the power source charges the capacitor and the voltage will rise. When MOUSE is on, it will consume the energy and the voltage will drop. MOUSE will shut off when the voltage hits a pre-defined minimum value, hence the voltage on the capacitor will fluctuate within a specified range. When the voltage hits the lower end of the range, power is instantaneously cut - MOUSE does not do any preparation for the shutdown. We start all benchmarks with a capacitor that has voltage just below the cutoff, hence all benchmarks begin with an initial charging time. Modern MTJs and Projected MTJs have different operating voltages [398], so we use a different voltage range for each technology. We let the voltage fluctuate between between $400\,\mathrm{mV}$ and $420\,\mathrm{mV}$ when using Modern MTJs and between $100\,\mathrm{mV}$ and $120\,\mathrm{mV}$ when using Projected MTJs. Switched-capacitor converters are used for upconversion and downconversion [148] to supply the required voltages for all operations. All required voltages can be acquired by using conversion ratios of 0.75, 1, 1.5, and 1.75 [173, 289]. We evaluate MOUSE on the power supplied by the converter, the evaluation does not include regulator efficiency overhead. The converter may have an efficiency anywhere between 35-80%, hence the energy harvester may need to provide roughly 1.25-2.85$\times$ the energy that MOUSE consumes. As noted in Section 4.3.4, a single instruction is performed in every cycle. A portion of the cycle must be dedicated to changing the output voltage of the converter (if consecutive operations require different voltage levels). The time overhead can be overlapped with the row activations.

It is desirable to match the capacitor size to the expected energy consumption. Hence, we also use different capacitor sizes for modern and project MTJs. We use a $100\,\mu\mathrm{F}$ capacitor (energy buffer) with Modern MTJs and a $10\,\mu\mathrm{F}$ capacitor for Projected MTJs. The optimal capacitor size depends on the technology and the program being executed. When deployed, a system such as Capybara [71] could be used to tune the parameters of the energy buffer.

Given that energy harvesting power sources can vary significantly in how much power they can provide, we sweep the power source over a wide range. At the low end, we test from $60\,\mu\mathrm{W}$ which is approximately what can be harvested from a 1cm^2 thermal energy harvester running on body heat [202, 181]. This is well below the operating power of MOUSE. At the high end we use $5\,\mathrm{mW}$, which is the same power harvested by the beyond edge device SONIC [120]. This can nearly power MOUSE continuously. Beyond edge devices deployed as satellites will likely use solar cells as power sources [226]. The amount of power which can be harvested will depend on the size of the cells (typically very small) and their orientation which is likely to change over time.

Area Overhead: The CRAM arrays used in MOUSE have a similar area overhead as MRAM arrays. The extra overhead of STT CRAM is an extra bit line per column, which is a minor impact. For SHE CRAM, a second transistor and SHE channel is required in each cell, which has a significant impact.

We base our cell area estimates on Zabihi et. al. [400]. We use configurations where the access transistors have a resistance less than $1\,\mathrm{k}\Omega$ and give an extra 10% to account for spacing and layout issues. The access transistors and MTJs can be placed on separate layers. As the transistors are much larger, they dominate the area overhead. As the SHE architecture has twice as many transistors, it is approximately twice as large. We use NVSIM [92] to estimate the area overhead of peripheral circuitry. NVSIM reports the percentage of chip area which much be dedicated to the peripheral circuitry for different memory sizes. We increase the area overhead for each benchmark accordingly. Our conservative area estimates are shown in Table 4.3.

Impact of Temperature: MTJs have been demonstrated to function over a wide range of temperatures [397, 196]. However, the MTJ resistance increases at colder temperatures, which will increase energy consumption. We test MOUSE both at -170°C (cold) and 123°C (hot). To model

Table 4.3: Area required for MOUSE for different benchmarks and configurations. Units are in mm^2.

Benchmark	Total Memory	Modern STT[400]	Projected STT [400]	SHE
SVM MNIST	64MB	28.04	21.27	42.54
Binarized	8MB	2.99	2.27	4.53
SVM HAR	16MB	5.97	4.53	9.06
SVM ADULT	1MB	0.39	0.29	0.58
BNN FINN MNIST	8MB	2.99	2.27	4.53
BNN FPBNN MNIST	16MB	5.97	4.53	9.06

the impact on MTJs we take data from Yuan et. al. [397]. For cold temperatures, we conservatively estimate the MTJ resistance increases by 30%. For the STT architecture, this increases the write, read, and logic energy consumption by 30%. For SHE, the write energy remains unaffected as the SHE channel (which is metallic) is use for write operations. However, energy consumption still increases for read and logic operations. The CMOS circuitry will generally perform better at cold temperatures, having a lower latency and potentially lower energy [329, 276, 393]. However, to be conservative, we assume no additional efficiency of the peripheral circuitry. The latency improvement of CMOS does not benefit MOUSE as we choose to maintain the same clock rate across temperature ranges. Hence, the latency of each instruction remains the same. At hot temperatures, the MTJ resistance drops by approximately 13% [397]. We model this in an identical fashion to cold temperatures, where we change the energy efficiency of each operation.

Impact of Radiation: As noted in Section 4.5.2, MTJs have an inherent resilience to radiation. However, the CMOS components of MOUSE remain vulnerable. Errors in the CMOS circuits can lead to undefined behavior. Hence, in order for MOUSE to work in orbital deployment, these errors must be suppressed. Circuit-level strategies can make CMOS circuits resistant to soft errors [406, 317]. These strategies come with a power and delay cost. We choose to be conservative and assume a large overhead. We assume a 60% increase in CMOS energy and a 10% increase in CMOS latency, one of the largest overheads reported by Zhou et. al. [406]. Hence, the performance and efficiency of an orbitally deployed MOUSE will be reduced, relative to that of its counterpart designed for terrestrial deployment.

Baseline for Comparison: We compare MOUSE with SONIC [120], a beyond edge device which performs machine learning inference on the same benchmarks we use. As SONIC was evaluated at room temperature, we must estimate its performance at different temperature ranges. To be conservative, we assume SONIC can fully exploit the benefit of CMOS operation at cold temperatures, increasing performance by 30% [276]. We also assume it suffers no negative consequences of varying temperature (hot or cold) and it pays no overhead resilience to radiation. We also compare against estimates of the vector architecture MANIC [121]. We also give MANIC overly optimistic assumptions, a 30% boost in performance and no overhead for temperature or radiation. MANIC was not evaluated on end-to-end inference, rather on computational kernels required for inference (i.e. convolution). Hence, we rely on rough estimates of its performance on the same benchmarks. We follow the authors' statement, that MANIC 9.6$\times$ more energy efficient than SONIC [121].

4.7 Evaluation

Continuous Power: Continuously powered MOUSE at room temperature and related work is reported in Table 4.4. MOUSE implements both BNNs and SVMs. SONIC [120] is beyond edge

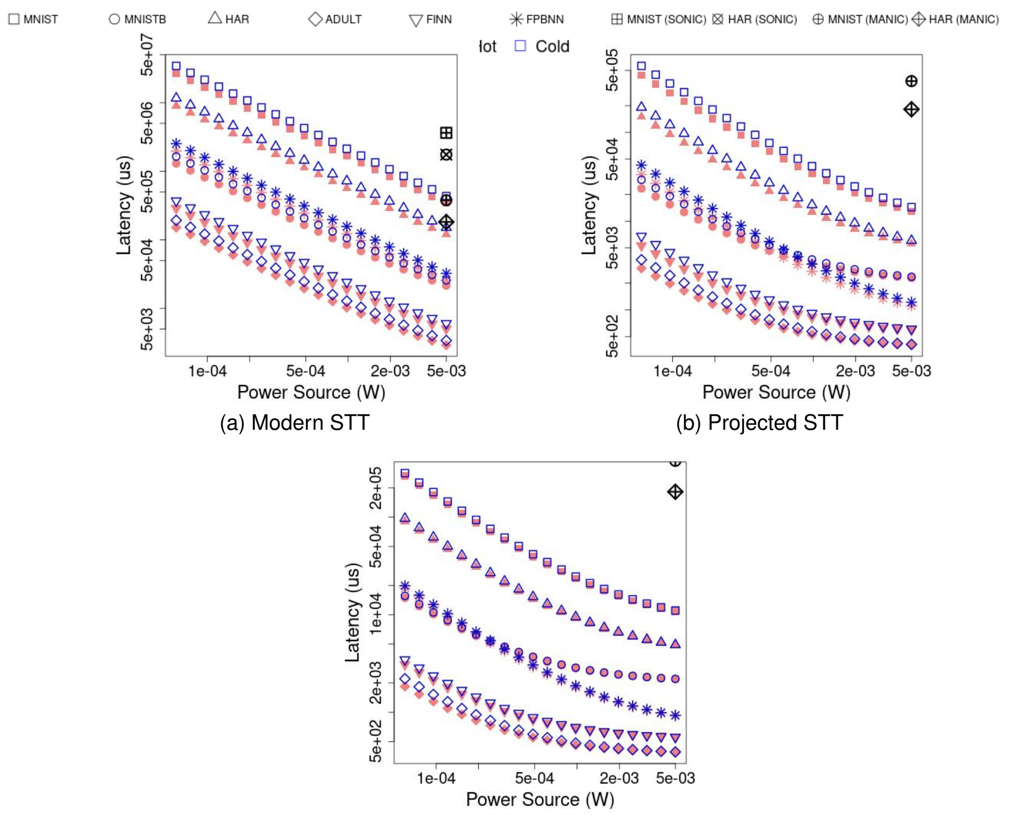
Table 4.4: Continuously powered MOUSE at room temperature (using STT design and modern MTJ devices) and related work under continuous power. The CPU does not benefit from MNIST binarization as it still performs 64-bit integer multiplication.

Benchmark	Latency (μs)	Energy (μJ)	#SV	I/D Mem (MB)	Area (mm^2)	Accuracy
SVM (CPU)						
MNIST	169,824	5,094,702	11,813	-	-	97.55
MNIST (Binarized)	192,370	5,771,085	12,214	-	-	97.37
HAR (integer) [14, 360]	127,494	3,824,822	2,809	-		95.96
ADULT	4,368	131,052	1,909	-	-	76.12
MOUSE SVM (Modern STT)						
MNIST	23,116	1,700	11,813	4.5 / 30.0	28.04	97.55
MNIST (Binarized)	6,071	81.43	12,214	1.25 / 6.0	2.99	97.37
HAR (integer) [14, 360]	11,312	575.8	2,809	2.25 / 10.0	5.97	94.57
ADULT	1,104	9.06	1,909	0.25 / 0.5	0.39	76.12
MOUSE BNN (Modern STT)						
MNIST (Binarized) FINN	1,605	18.04	NA	3.15/1.71	2.99	98.4
MNIST FP-BNN	2,150	125.4	NA	4.20 / 8.00	5.97	98.24
libSVM [57]						
MNIST	7,830	234,900	8,652	-	-	98.05
MNIST (Binarized)	19,037	571,116	23,672	-	-	92.49
HAR (integer)	1,701	51,042	2,632	-	-	93.69
ADULT	379	11,370	15,792	-	-	78.62
SONIC [120]						
MNIST	2,740,000	27,000	NA	0.256	$>$ 100	99
HAR	1,100,000	12,500	NA	0.256	$>$ 100	88

device which uses TI-MSP430FR5994 microcontroller to run neural networks on the same benchmarks. For reference, our custom SVM implementation and optimized SVMs from libSVM [57] are run on a Intel Haswell 5-2680v3 processor. To be conservative, we account only for the processor power consumption and assume it operates at its idle power. Overall, MOUSE has a significant energy efficiency advantage and a competitive latency. Notably, MOUSE consumes more memory than SONIC. However, this is reasonable as MOUSE consists nearly entirely of non-volatile memory, which has high density. MOUSE does not require external logic or area costly volatile memory.

Intermittent Operation: We now evaluate MOUSE with intermittent computation, where a small power source is charging a capacitor that MOUSE can draw energy from. The latency (including time powered off) of all benchmarks with each MTJ device (and different operating temperatures) over the range of power sources ($60\,\mu$W - $5\,$mW) is plotted in Figure 4.4, along with a comparison to SONIC [120] and MANIC [121]. All MOUSE configurations are able to significantly outperform SONIC for the same power budget. Despite conservative estimates of MTJ performance, conservative estimates of peripheral circuitry, and very optimistic estimation of MANIC across the temperature range (30% boost in performance and no overhead for temperature or radiation) MOUSE has a similar performance with MANIC. On the MNIST data set, if MOUSE uses 8-bit inputs, its latency is 0.91$\times$ (1.15$\times$) that of MANIC at hot (cold) temperatures. On the HAR data set, MOUSE has a latency that is 0.66$\times$ (0.83$\times$) that of MANIC at hot (cold) temperatures. Hence, MOUSE has better performance on average, with better results at warmer temperatures.

At cold temperatures MOUSE has a higher latency than when hot. At $60\,\mu$W, overall cold is 23.4% slower on average. While MOUSE has the same clock rate and issues instructions at the same rate, the instructions consume more energy when cold. Hence, MOUSE will run out of energy and have to power off more frequently. Temperature has a varying level of impact on each MTJ technology. Modern STT has a 33.3% higher latency and Projected STT has 28.5% higher latency at cold temperature, across all benchmarks. SHE is less effected by temperature because write



(a) Modern STT

(b) Projected STT

(c) SHE

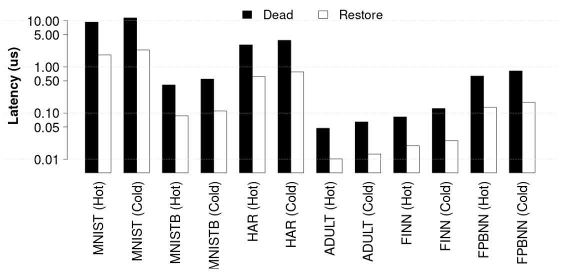
Figure 4.4: Latency (μs) vs. Power Source (W) for each MOUSE configuration and SONIC [120]. MOUSE at hot temperature is shown in Red/Filled shapes and at cold temperature is shown in Blue/empty shapes.

and logic operations use the SHE channel, which is not only more energy efficient but less affected by temperature. SHE has an 8.6% higher latency across all benchmarks at cold temperature.

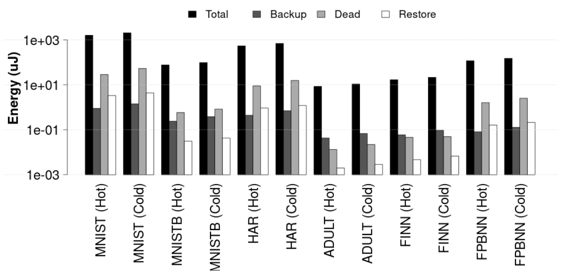
Independent of temperature, SHE is the most energy efficient. Because of this it drains the capacitor less often, and hence has fewer power outages leading to the overall lowest latency. Projected STT has a lower latency than Modern STT, as it can operate at higher frequency (11ns per instruction vs. 33ns) and it is more energy efficient.

MOUSE spends negligible amounts of energy while powered off. Hence, the energy consumption is nearly independent of the power supply. The vast majority of the energy is dedicated to normal program execution. A small portion is dedicated to overhead for intermittent execution, which will vary depending on the number of interruptions (which is determined by energy efficiency and the capacitor size). The total energy is plotted in Figure 4.5(b) for Modern STT; in Figure 4.6(b) for Projected STT; and in Figure 4.7(b) for SHE; assuming a $60\,\mu\mathrm{W}$ power source.

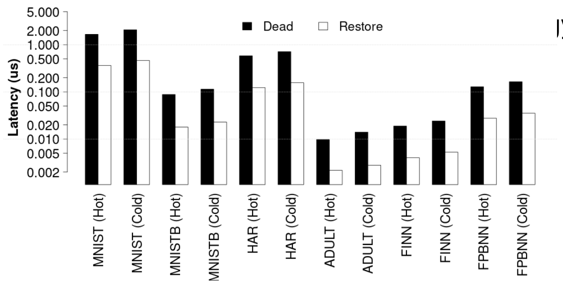
There are metrics specific to beyond edge devices which indicate how efficient the checkpointing strategy is [312]. In addition to the total energy, we report the Backup energy, Dead energy, and the Restore energy. Backup refers to any actions required prior to shutdown to save the state. For more traditional architectures, this involves writing data back to non-volatile memory. For MOUSE, the only backup operations are saving the PC, flipping the parity bit, and writing values into the CBR (to indicate which columns are active). MOUSE does the first two on every instruction, and the second only AC instructions. Dead refers to any computation that must be re-performed after



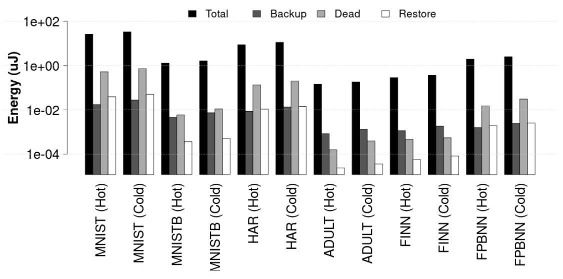
(a) Latency



(b) Energy



(a) Latency



(b) Energy

Figure 4.6: Latency/Energy Breakdown: Projected STT.

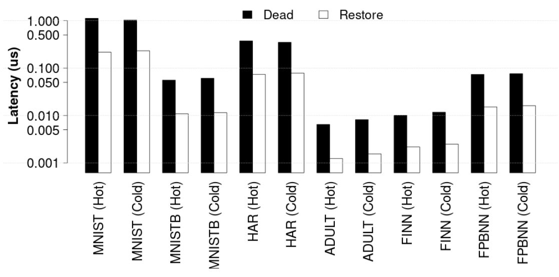
restart (which was lost due to the shutdown). For MOUSE, this is at most re-performing the very last instruction. Restart is any actions required to put the device back into operating condition after a shutdown. For MOUSE, this is the re-activation of columns with an AC instruction.

Backup has no associated latency, as MOUSE's backup operations are overlapped with normal program execution. However, we do report Dead latency, which is the time it takes to re-perform the last instruction, and the Restore latency, which is the time it takes to re-activate columns. To remain efficient, the Backup, Restore, and Dead latency and energy should be low.

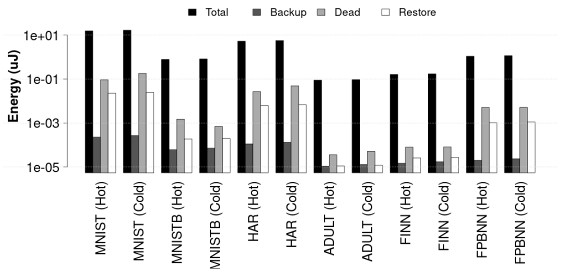
Overhead for Backup, Restore, and Dead are reported in Figure 4.5 for Modern STT; in Figure 4.6, for Projected STT; and in Figure 4.7, for SHE. Note that the y-axis is log scale. The total energy encapsulates all energy used for computation, as well as Backup, Restore, and Dead energy. Also note the total latency is provided for all architectures in Figure 4.4 – where the breakdown figures capture the data for the $60\,\mu$W power source.

The overheads for Backup, Dead, and Restore increase with cold temperature. This is for two reasons. The first is that the actions required for each will cost more energy due to the MTJ characteristics. For exampling, writing the PC value or re-performing the last instruction will involve MTJ operations, which will take more energy at cold temperatures. The second reason is that the overall lower energy efficiency at cold temperatures leads to more power outages. At cold temperature, across all benchmarks and technologies, MOUSE restarts 24.4% more often than at hot temperatures. This increases the number of instructions that need to be re-performed and the number of times architectural state variables will be saved.

Modern STT is the least energy efficient, which means it must restart the most. Because of this it has the largest relative Dead energy. At the extremely low power of $60\,\mu$W, on average, across all benchmarks, Dead energy is 0.98% (1.09%) of the total energy at hot (cold) temperature. The projected MTJs have lower overhead, where Dead energy (on average) becomes 0.796% (0.804%) of the energy for Projected STT and 0.194% (0.323) of the total for SHE at hot (cold) temperatures. Dead latency, on the other hand, is 0.068% (0.084%) of the total for Modern STT, 0.040% (0.050%) of the total for Projected STT, and 0.020% (0.020%) of the total for SHE with hot (cold) temperatures.



(a) Latency



(b) Energy

Figure 4.7: Latency/Energy Breakdown: SHE.

Restore is only 0.013% (0.016%) of the latency and 0.066% (0.069%) of the energy for Modern STT; 0.008% (0.010%) of the latency and 0.048% (0.049%) of the energy for Projected STT; and 0.0037% (0.0040%)of the latency and 0.0436% (0.0436%) of the energy for SHE with hot (cold) temperatures. As Restore latency and energy is due to peripheral circuitry, SHE has no advantage over STT for an individual restart. However, SHE still requires fewer restart operations due to its overall increased energy efficiency. Backup energy is, on average across all benchmarks, 0.304% (0.337%) for Modern STT; 0.350% (0.340%) for Projected STT; and 0.009% (0.009%) for SHE. Backup has no associated latency as it is performed at the same time as each instruction on every cycle. Overall the Backup, Dead, and Restore overheads increase only modestly at cold temperatures. Hence, the checkpointing mechanisms remain efficient across the wide temperature range and MOUSE is suitable for use as an intermittent accelerator in the harsh environments of LEO.

Restore and Dead latency and energy are all zero for the case of a continuously powered system. This is because there are no power outages and, hence, never a need to restart the system or re-perform any potentially unfinished instructions.

CMOS hardening decreases efficiency due the peripheral circuitry consuming more energy. The overhead varies across benchmarks and technologies, but tends to be higher at lower power (due to requiring more restarts, which incurs re-activation of the peripheral circuitry) and higher temperature (due to peripheral circuitry having a larger percentage share of the total energy). At $60\,\mu$W, the lowest power tested, CMOS hardening increases energy consumption by 26.9% on average (44.8% at worst) in cold and by 32.3% on average (49.2% at worst) at hot temperature.

4.8 Conclusion

In this paper we improve the hardware efficiency and programmability of MOUSE [298], a non-volatile processing-in-memory (PIM) accelerator for beyond edge computing to to enable orbital deployment. Specifically, we expand the PIM instruction set and add architectural support for branch instructions for enhanced programmability. We develop more efficient mechanisms for column activation, reducing the complexity of the peripheral circuitry. We show that MTJ devices and supporting CMOS circuitry operate correctly across a wide temperature range. Even accounting for the overhead to maintain resilience against radiation, this advanced architecture features high performance and extreme energy efficiency. Combined with the guarantee for intermittent safe operation and inherent low-cost checkpointing mechanisms, the final result is a design well suited for use as a nanosatellite in low earth orbit.

Chapter 5

Towards Homomorphic Inference Beyond the Edge

5.1 Introduction

Devices are now being built which can operate without batteries or other constant power sources [120]. This allows these devices to be deployed in a wide variety of environments, where traditional power sources are typically unavailable. Such environments are referred to as *beyond the edge*, and can reside inside the structures of buildings [242], in the remote wilderness [120], in outer space [226], inside the human body [134], or simply scattered throughout a city. The extreme deployment capability opens up many new applications.

A significant limitation of these devices, however, is the very strict power budget. Available power sources range from less than $1\,\mu$W from RF energy [180, 394, 279], to $60\,\mu$W from thermal energy harvesters [239, 180], up to $100\,$mW per square centimeter from solar panels [81, 73, 129, 166, 180]. Depending on the specific deployment location, devices could be limited to only the weakest power sources. For example, if embedded within the walls of a building, a device will have no access to sunlight and will have to rely on either RF or thermal energy. Hence, beyond edge devices have to operate effectively while consuming minimal power. Energy efficiency determines the performance [120], as typically most time is spent waiting for sufficient energy.

A potential solution to this power limitation is to perform the computation in a remote server, rather than on the device itself. Beyond edge devices can collect sensor data and then send it to a server to be processed, rather than performing computation locally. This can provide a benefit if intense processing is required. This approach can even make use of un-trusted servers by leveraging *homomorphic encryption*, where the device can encrypt data and the untrusted server can process it without decrypting [368].

Unfortunately, even the this data transmission itself can be excessively costly for beyond edge devices. If the server is a few kilometers away, the cost can be as much as $400\,\mu$J per bit [40]. Even sending a minimal amount of unencrypted data can consume unreasonable amounts of energy. For example a single sample of the MNIST data set [199] would consume $2.5\,$J. The situation is much worse or encrypted data, where, for most practical purposes, the smallest homomorphic encryption configuration will consume approximately 400,000 bits [321]. In this case, transmission would cost 160 J, well beyond what is practically obtainable by beyond edge devices.

However, the severe limitation of transmission cost can be dramatically decreased by reducing the distance. Within tens of meters Bluetooth Low Energy (BLE) can consume as little as 158 pJ per bit [303]. Hence, *offloading computation remains feasible if the server is nearby*.

However, it is clearly impossible to deploy a typical server (eg. a cloud computing data center) within tens of meters of all beyond edge devices. Hence, to enable secure beyond edge computing, an entirely different type of server will be required. has previously been noted that beyond edge devices can operate as *fleets*, where large collections of individual devices act together to solve a single problem [86]. We leverage this idea, and note that *different beyond edge devices will have different power budgets*. Hence, different classes of beyond edge devices can be specifically engineered for different tasks. In this work, we propose a beyond edge accelerator which can act as a local “mini-server”. It is designed to operate in an environment with relatively higher power, such as on the rooftop where sunlight can be collected (milliWatt power budget). Using this (relatively) large power budget, it can perform computational work for nearby beyond edge devices which are more power constrained. For example, a device located within the walls of a building and operating under an extremely low power budget will be able to offload its computation to the mini-server with low-distance, low-power communication. While the mini-server will have significantly reduced performance relative to a standard cloud-based server, it has extreme deployment capabilities, allowing it to exist close to other beyond edge devices.

While the proposed mini-server can be considered a trusted device, it does still introduce security concerns. Being deployed beyond edge, in potentially insecure environments, the device itself may be compromised or stolen. Hence, unencrypted data transmitted to or stored on the mini-server (such as machine learning models) is vulnerable. In order to maintain security, the mini-server must work with encrypted data. Hence, it must perform *homomorphic* computation, which allows data to remain completely encrypted. Unfortunately, this introduces a significant challenge, as homomorphic computation is orders of magnitude slower and more energy costly than standard computation. In this chapter we investigate how to avoid much of this overhead with two approaches.

First, we perform only linear operations on the mini-server, such as multiplication and addition, which have a much lower homomorphic overhead (leaving the few remaining non-linear operations to be performed on the original device). If most required operations are linear, such as in *support vector machine* (SVM) inference used in this work, this process remains efficient. Second, we are able to keep the computational depth low, which minimizes the amount of noise. This allows us to avoid the process of *bootstrapping*, which is the most costly component of homomorphic computation. Despite these advantages, homomorphic computation remains highly energy costly. Consequently, the mini-server must be extremely energy efficient in order to compensate.

An additional challenge of beyond edge deployment is that the mini-server must also be able to tolerate power outages. Checkpointing and guaranteeing correctness has significant overhead for energy efficiency and complexity [225]. We design the mini-server to make use of non-volatile processing-in-memory (PIM), which not only provides high energy efficiency, but inherent resilience to interruptions and low-cost checkpointing mechanisms [298].

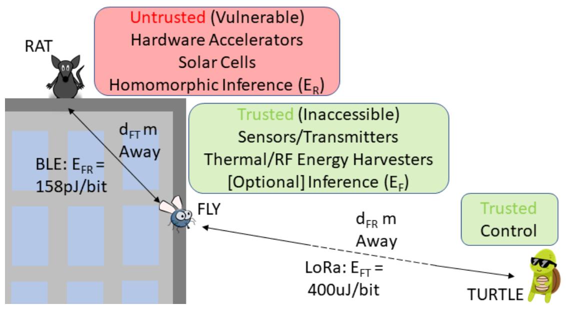


Figure 5.1: The FLY can transmit data to nearby RAT to offload computation. Final destination of results is the distant TURTLE.

5.2 Problem Statement and Definitions

In this section we cover the basic scenario covered and the assumptions made about the problem. We assume that there are three types of devices in play: the FLY *(in the wall)*, the RAT *(on the roof)*, and the remote TURTLE.

The TURTLE is a distant and secure server in a secure environment with no power restrictions. It is the control center and is the intended destination for all results of interest.

The FLY is a beyond edge device running on harvested energy. It acquires data with sensors, with the intention of reporting interesting results back to the TURTLE. The FLY is in an concealed environment, where the device itself is secure. For example, it may be embedded within the walls of a building at the time of construction, where it can monitor vibrations and act as a warning mechanism for structural integrity. Hence, it can operate on secure (un-encrypted) data. Due to its environment, it is limited to RF or thermal energy, which is in the microwatt range [180]. Hence, it must operate under a very low power budget.

The RAT ("mini-server") also operates beyond the edge and relies on harvested energy. However, it is deployed in a less concealed environment, such as on the roof of the same building. An advantage of this location is that is has access to a higher power budget. During daylight hours it can harvest sunlight, which can provide a power budget of tens of milliwatts [180]. However, due to being exposed, RAT cannot be fully trusted. Data stored on RAT, such as propriety machine learning models, is vulnerable. *Homomorphic* encryption and computing overcomes this obstacle by allowing data to remain encrypted at all times [43, 102], covered in Section 5.3.1.

The relative placements of the devices are shown in Figure 5.1. TURTLE is d_{FT} meters from FLY, and it takes E_{FT} μJ per bit to transmit from FLY to TURTLE. RAT is d_{FR} meters away from FLY and it takes E_{FR} μJ per bit to transmit from FLY to RAT. In this scenario, $d_{FT} >> d_{FR}$ because RAT can be deployed into similar environments as FLY. Hence, FLY can much more easily communicate with RAT than with TURTLE.

If the FLY wishes to report back to the TURTLE, it has 3 options.

(1) Send all sensor readings to the distant TURTLE. TURTLE will be in charge of all computation.

(2) Perform processing (inference) locally and only send TURTLE the meaningful results.

(3) Offload processing (inference) to the nearby RAT and then only send the meaningful results to TURTLE.

Option 1 is the simplest, however, it is also typically inefficient. Communication has a higher energy cost per bit than computation. Meaningful sensor data (capturing relevant information) is rare, hence transmitting all data is wasteful [120]. Assume that (non-homomorphic) inference on FLY takes E_F μJ and the probability that a sample contains results of interest is α. Then, performing inference locally is more efficient if

$$E_F + \alpha \times E_{FT} < E_{FT} \tag{5.1}$$

This equation almost always holds as long distance communication (beyond 3km) with low-power long-range (LoRa) hardware is relatively energy costly, consuming up to $400\,\mu$J per bit [40]. For example, sending data samples (such as an MNIST image sample[199]) would consume $E_{FT} = 2.5$ J per sample. Alternatively, the energy required for local inference (with standard beyond edge hardware) is in the range of $E_F = 27$ mJ [120]. In this case, local inference will be more efficient if $\alpha < 98.9\%$. Given results of interest are relatively rare, this is likely to be the case.

Option 3 replaces local inference (consuming E_F μJ) on FLY with

(1) (Optional) Encoding and encrypting sensor data (consuming $E_{encrypt}$)

(2) Transmitting data to RAT

(3) Homomorphic inference on RAT (consuming E_R)

(4) Transmission of results back to FLY (consuming E_{RF})

(5) Decrypting results on FLY (consuming $E_{decrypt}$)

For option 3 to be superior to option 2, the following equation must hold:

$$E_{encrypt} + E_{FR} + E_{RF} + E_{decrypt} < E_F \tag{5.2}$$

In words, it must be more efficient for FLY to transmit to RAT and receive the results than it is to process the data itself. If the input data collected by FLY is considered sensitive, FLY must encrypt the data before transmission, in order to maintain security in un-trusted environments. $E_{encrypt}$ and $E_{decrypt}$ can be done with 0.06 mJ (using the same parameters as this paper) with specialized hardware for homomorphic encryption on edge devices [368]. If the inputs are not sensitive, then they can be sent as raw data and $E_{encrypt}$ can be dropped from Equation 5.2. However, RAT will still return encrypted data in order to protect its ML model, hence $E_{decrypt}$ is still required. Due to the close proximity of FLY and RAT (in the order of meters), communication cost is significantly reduced. Technologies such as Bluetooth Low Energy (BLE) can be used, with some configurations offering as little as 158 pJ per bit [303]. At such efficiency, (un)encrypted data can be sent to RAT (E_{FR}) and encrypted results can be transmitted from RAT to FLY (E_{RF}) with less than ($1\,\mu$J) $80\,\mu$J. Hence, BLE can provide communication which will cost FLY less energy than local processing. FLY will require both BLE (for communication with RAT) and LoRa (for communication with the distant TURTLE).

The other condition required for option 3 to be viable is that the homomorphic computation energy on RAT (E_R) must be reasonable. While operating beyond edge, RAT has a higher energy budget than FLY. However, homomorphic computing has a very large overhead that can easily become excessive. RAT must be sufficiently efficient in order to return results to FLY promptly. For RAT to provide any benefit, the following equation must hold:

$$\frac{E_{FR}}{P_F} + \frac{E_R}{P_R} + \frac{E_{RF}}{P_R} < \frac{E_F}{P_F} \tag{5.3}$$

where P_F is the power available to FLY and P_R is the power available to RAT. In words, RAT must have sufficient efficiency to homomorphically compute and return data within its power budget faster than FLY can non-homomorphically compute within its power budget. In this paper, we design RAT to perform such computations within a reasonable energy budget.

5.3 Background

5.3.1 Homomorphic Computing

Homomorphic encryption allows computation to occur on encrypted data, typically with a significant time and energy overhead [43, 102]. However, some operations are more efficient than others. For example, linear operations tend to have the least overhead. The vast majority of the operations performed in SVMs are linear multiplications and additions, explained in Section 5.3.2. We choose to perform only these linear operations on RAT, leaving the final, non-linear operations to be performed after decryption on FLY.

We use the BFV scheme [43, 102], which provides integer arithmetic and is available on both Microsoft SEAL [321] and PALISADE [280]. First, a vector of data is *encoded* into a *plaintext*, which is a set of coefficients for a polynomial. Then, the plaintext is *encrypted* into a *ciphertext*. The length of the input vector is equal to the number of coefficients in both the plaintext and ciphertext (also called the degree of polynomial modulus). The length of the vector sets a noise budget, where larger vectors have larger noise budgets. All homomorphic computations consume some of the noise budget, and if the budget is exceeded, the data can no longer be successfully decrypted. Testing with Microsoft SEAL [321], we found 4,096 was the minimum vector size that could successfully complete the SVM computations. The *coefficient modulus* is the maximum value of the ciphertext coefficients, which is represented in a residue number system consisting of three 36-bit prime numbers. A ciphertext consists of two such polynomials. Hence, the total memory required by our ciphertexts is

$$2 \times 3 \times 36 \times 4096 = 110KB \tag{5.4}$$

Homomorphic addition is relatively straightforward, consisting of element-wise additions of the vectors involved. Modulus operations must be performed routinely to prevent the coefficients from growing excessively large. Given that the coefficient modulus is known ahead of time, modulus can be implemented with efficient (virtual) shift, add, and subtract operations in the memory [259].

Homomorphic multiplication is significantly more complicated and consists of multiple steps. The primary components are the number theoretic transform (NTT), which is an integer variant of the FFT, and scales and base conversions [359]. We follow the computation steps laid out by Özerk et. al. [273] for NTT and by Al Badawi et. al. [10] for all other steps.

5.3.2 Support Vector Machines

Support vector machines (SVM) are popular machine learning (ML) algorithms which perform well on small problems. The advantages of SVMs are a strong theoretical foundation and proven performance on a wide variety of problems [267]. SVMs are trained by inspecting input samples to a problem, where samples which are indicative of each classification are identified. After training is complete, these samples are referred to as *support vectors*, and are used to compare to new inputs. The new inputs are classified based on how similar they are to support vectors from each class. By construction, a standard SVM model produces a binary classification, producing a classification of ± 1. We use a simple extension to enable multi-class classification, where a separate SVM is trained to identify each class (the one-versus-all method [94]). For example, to classify MNIST, 10 SVMs are used where each identifies one digit 0-9.

A specific advantage for SVMs is that they consist mostly of linear operations, which can be implemented relatively efficiently in homomorphic computing. This provides an advantage over neural networks, for which sophisticated strategies must be invoked to reduce the overhead of frequent non-linear operations [292, 58]. SVM inference involves finding the dot product of the input sample with each of the support vectors. This involves purely multiplications and additions. At the very end, the dot products are squared (multiplication by self), summed, and finally compared, which represents a non-linear operation. While complete SVM training and inference has been successfully demonstrated using homomorphic operations [275], we opt to perform only the initial multiplication and summations on RAT. We leave the final squaring and sum to be performed on FLY. This prevents us from exceeding the computational depth allowed by our homomorphic encryption (without performing bootstrapping) and avoids the high overhead incurred by the final sum (which requires highly expensive rotation operations to sum elements in the same ciphertext [292]).

5.3.3 (Digital) Computing with Magnetic Tunnel Junctions

Magnetic Tunnel Junctions (MTJ) are resisistive memory devices consisting of two magnetic layers, a *free layer* and a *fixed layer*. The polarity of the free layer can change but the fixed cannot. When the magnetic layers are (not) aligned, the MTJ is in the parallel R_P (anti-parallel R_{AP}) state and has a low (high) resistance. Passing a sufficient amount of current (I_{switch}) through the MTJ from the free (fixed) layer to the fixed (free) layer will set the MTJ into the R_P (R_{AP}) state.

Logic can be performed with MTJs via thresholding [62, 222], where input MTJs (in parallel) are in series with an output MTJ, as shown in Figure 6.1. For example, a NAND can be peformed by presetting the output MTJ to logic 0 (R_P). A (gate-specific) voltage is applied such that electrons flow from the inputs to the output. If both input MTJs are logic 1 (R_{AP}, high resistance) the current through the output will be *less* than I_{switch}, and it will remain 0. However, if *either* input MTJ is logic 0 (R_P, low resistance) the current will be *greater* than I_{switch} and the output will switch to 1. In a nutshell, under the fixed gate-specific voltage the current through the output changes as a function of the input states, and only incurs switching (a change in the output state from the preset) according to the truth table of the corresponding gate. Different logic operations, including NOT, AND, and (N)OR can be performed in identical fashion, with different output preset values and gate-specific voltages, respectively.

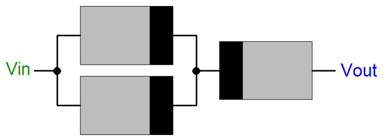


Figure 5.2: Circuit for logic with MTJs. Input MTJs (in parallel) are connected in series with an output MTJ. A gate-specific voltage applied across V_{in} and V_{out} drives a current which will switch the output MTJ depending on the state of the input MTJs, following the truth table of the corresponding gate. Fixed (free) layer of MTJ is shown in light (dark).

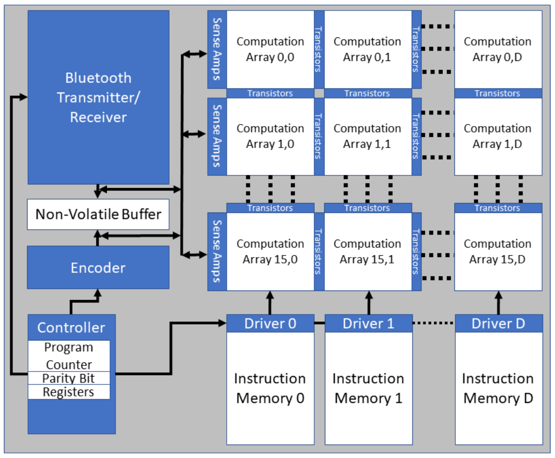


Figure 5.3: RAT architecture containing non-volatile computation arrays and instruction memory along with volatile circuitry for driving operations. Only the first column of computation arrays has sense amplifiers, allowing the peripheral circuitry to both read and write from them. Data transfer for all other computation arrays occurs with logic operations. As proposed by Gupta et. al.[140] transistors between neighboring arrays enables them to participate in the same logic operation.

5.4 Architecture Design

RAT must perform homomorphic computation while both remaining within a low power budget and being resilient to intermittent operation. We design an accelerator using a non-volatile processing-in-memory (PIM) substrate which is shown to be both highly energy efficient and inherently intermittent resistant [298]. We augment the PIM capability with simple, intermittent safe protocols and hardware for reception, transmission, and encoding. The architecture is shown in Figure 5.3. First, in Section 5.4.1 we describe the hardware contained within RAT, then in Section 5.4.2 we cover how different system components interact with each other, and tolerate intermittent operation.

5.4.1 Hardware

Computation Arrays

The primary component of RAT are arrays of MTJ devices used for computation. We call these *computation arrays*. These arrays both hold data and perform all homomorphic computation on it. The array architecture and cell design is shown in Figure 5.4, which uses two transistors. This architecture allows for the logic operations discussed in Section 5.3.3 to occur in either the rows or

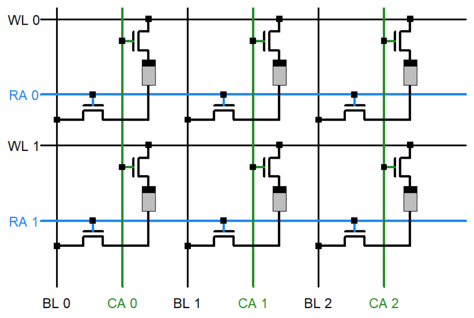


Figure 5.4: The 2T-1M cross-bar array architecture. Logic operations can be driven in memory by applying voltages either along bitlines (BL) or wordlines (WL). The access transistors, activated by the signal column activate (CA) and row activate (RA), remove sneak paths.

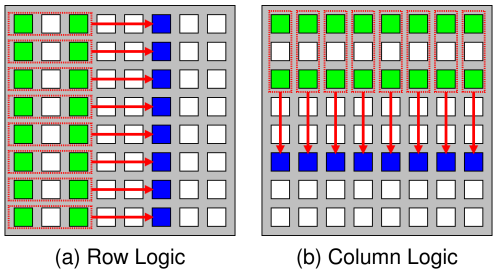


Figure 5.5: Computation arrays can perform row logic or column logic. Inputs are in green/light and outputs are in blue/dark.

columns of the memory array by applying voltages along the bitlines (BL) or the wordlines (WL). The access transistors, controlled by row activate (RA) and column activate (CA), remove potential sneak paths from the array. An example of row-logic is shown in Figure 5.6, where voltages V_{in} and V_{out} are applied to the bitlines, implementing the logic circuit shown in Figure 6.1. Many logic operations can be performed in parallel in each row (column) simultaneously, as long as the inputs and output reside in the same columns (rows) as shown in Figure 6.3. Having both row- and column-wise logic allows data to be moved efficiently within each array, removing the need for energy costly read and write operations. This computational capability is equivalent to digital cross-bar arrays [348, 32], but without the sneak paths which waste energy and introduce correctness concerns [206]. In row- (column-) logic, for each operation the column (row) addresses of the input(s) and output must be specified, along with which rows (columns) are participating in the operation. When rows or columns are performing computation we refer to them as *active*.

Using two transistors per cell significantly increases the area per cell from $0.0012\,\mu\text{m}^2$ [254] to $0.038\,15\,\mu\text{m}^2$ [400]. However, two transistors are essential for removing cross-bar sneak paths [207], which would significantly increase energy consumption. As energy efficiency is paramount for beyond edge devices [121], this cost in area is necessary. Due to the very large amount of data required by homomorphic computing, the area overhead for computation arrays can become substantial. For the smallest benchmark, ADULT, the computation arrays consume $6.72\,\text{mm}^2$. For the largest benchmark, MNIST, the area is $377\,\text{mm}^2$. For reference, the TI-MSP430FR5994, commonly used as a sub-component of beyond edge systems [120], consumes roughly $100\,\text{mm}^2$.

As described in Section 5.3.1, we use ciphertexts with 4,096 elements. Hence, it is ideal to

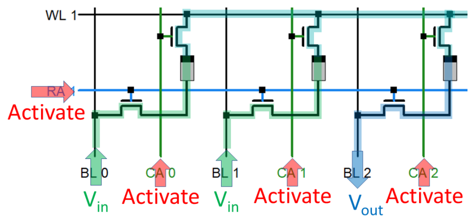


Figure 5.6: Voltages V_{in} and V_{out} are applied to the bitlines to drive logic operations. Row Activate (RA) and Column Activate (CA) set on which rows and columns the operations occur. Shown is a row-logic operation, which implements the same functionality as the circuit shown in Figure 6.1.

have 4,096 rows in order to store all elements and enable parallel element-wise homomorphic multiplications and additions. However, due to parasitic bitline/rowline resistance and capacitance, arrays are limited to 1024x1024 [400]. Hence, we use 16 rows of computation arrays, where each array is 512x512. Similar to Gupta et. al. [140], a row of transistors are used to conditionally connect to the bitlines of neighboring computation arrays. This allows logic operations to transfer bits between the neighboring arrays.

Each row of the computation arrays must store all required data for its given computation. The most space intensive subroutine of the homomorphic SVM is the initial multiplication, which can be between two ciphertexts or between a ciphertext and a plaintext. A single element of each polynomial is assigned to each row. Hence, each row requires $2 \times 2 \times 3 \times 36 = 432$ bits for the ciphertexts, 36 bits for *twiddle factors* for NTT operations [273], plus additional bits for temporary workspace. This easily fits within two columns of computation arrays (1024 bits). However, a third column of computation arrays is required to store additional twiddle factors required for different stages of the NTT algorithm. In total, $log_2(4096) = 12$ twiddle factors are required in each row.

Since logic operations can be used to transfer data within and between computation arrays, only the cells which store the final results (which are sent to the Bluetooth transmitter) need to be read. Hence, the vast majority of the computation arrays do not require sense amplifiers. As shown in Figure 5.3, only the first column of computation arrays contain sense amplifiers, which allows them to be used as a non-volatile buffer for the BLE transmitter/receiver and the encoder. Hence, input and output is passed through the first column on computation arrays.

In addition to the memory data, computation arrays must also maintain which rows or columns are currently active. For this purpose, we could use two dedicated non-volatile registers in each computation array - with as many bits in the registers as there are rows and columns in the computation array. Such registers can act as a bitmask for logic operations. However, rather than creating an additional hardware register, we can embed the registers into the memory itself, by dedicating a single row and column for each register. This allows registers to be written with standard logic and write operations. Dedicated instructions use the registers to activate the peripheral circuitry. For correctness guarantees which will be discussed in Section 5.4.2, two copies of the registers are required for both rows and columns, as shown in Figure 5.7.

Drivers and Instruction Memory

Signals must be sent into the computation arrays to perform logic gates. For every operation in each array, we need to specify which logic gate is being performed (what voltage to apply) and the

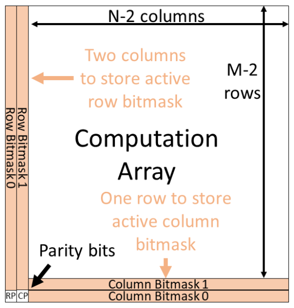


Figure 5.7: Two rows and columns of each computation array is dedicated to hold bitmasks for the active rows and columns. Standard logic and write operations can set the bitmask and dedicated instructions apply the bitmask to the peripheral circuitry. Two bitmasks are required for each in order to prevent corruption when modifying the bitmask. Parity bits, row parity (rp) and column parity (cp) indicate which bitmask is valid.

addresses of the input(s) and the output. Just as in prior work [298], the input and output addresses are specified with each instruction. Additionally, as noted in the previous paragraph, we must known which rows (or columns) are currently *active*. The currently active rows (or columns) are specified by the bitmask column and row within each computation array.

As each array could perform computation independently, many different gates, inputs, and outputs may need to be specified simultaneously. To efficiently distribute these signals, computation arrays are grouped into columns. Computation arrays in the same column act as a single unit and are driven by the same control signals. For each column of computation arrays, there is an associated *driver* and non-volatile *instruction memory* buffer.

The instruction memory arrays store the operations that each column of computation arrays is to perform (NOT, (N)AND, (N)OR) along with the row/column indices of the input(s) and the output of every operation. Each instruction requires the following information.

i. Opcode specifying the logic operation (3 bits)

ii. Up to three addresses for input(s) and output (12 bits each)

iii. Whether it is a row or column operation (1 bit)

To be specific, a driver is a CMOS circuitry that initiates the logic within the computation arrays. On command from the controller, each driver reads an instruction from the specified address and decodes it. Then it sends the input and output addresses to the row/column decoders and drives the appropriate voltage along the bit/row lines of the computation arrays. All drivers operate synchronously, executing the instructions at the same address (specified by the controller) in their corresponding instruction memories.

Encoder

RAT operates on encrypted data in order to keep the ML model secure (encrypted at all times). If the input is also considered sensitive, it should be encrypted on FLY (into a *ciphertext*) prior to transmission to RAT, in which case RAT can immediately begin processing and does not require an encoder. However, if the inputs are not considered sensitive, it will be more efficient to transmit the

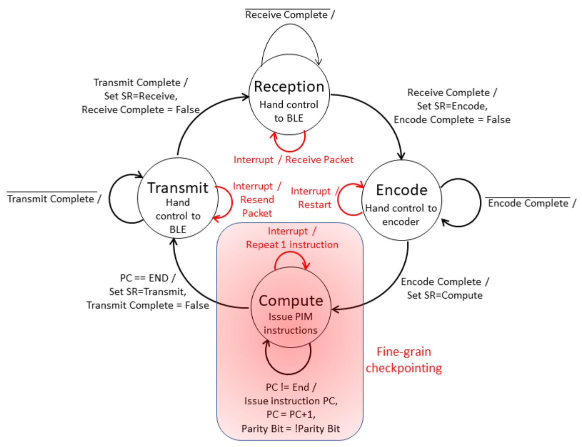

Receive Complete /
Reception
Hand control to BLE
Transmit Complete / Set SR=Receive, Receive Complete = False
Receive Complete / Set SR=Encode, Encode Complete = False
Interrupt / Receive Packet
Transmit Complete /
Transmit
Hand control to BLE
Interrupt / Resend Packet
Interrupt / Restart
Encode
Hand control to encoder
Encode Complete /
Interrupt / Repeat 1 instruction
PC == END / Set SR=Transmit, Transmit Complete = False
Encode Complete / Set SR=Compute
Compute
Issue PIM instructions
Fine-grain checkpointing
PC != End / Issue instruction PC, PC = PC+1, Parity Bit = !Parity Bit

Figure 5.8: RAT follows a simple state transition diagram. The controller is responsible for maintaining architectural state variables and ensuring correctness of state transitions. RAT has efficient checkpointing mechanisms in the compute phase, where it spends the vast majority of the time, due to ideal properties of MTJ devices.

raw data. In which case the input, once transmitted to RAT, must be encoded (into a *plaintext*) in order to properly interact with the encrypted ML model. Homomorphic encoders for edge devices have previously been developed [368]. We assume RAT contains such a hardware chip for this purpose.

Receiver and Transmitter

RAT uses Bluetooth Low Power (BLE) to communicate with the nearby FLY. BLE devices can offer extremely energy efficient communication at short distances [332, 124, 175, 341, 215]. We assume a configuration that enables particularly low power, down to $158\,\mathrm{pJ}$ per bit [303]. BLE will allow communication within a few tens of meters.

Controller

The controller comprises CMOS circuitry which orchestrates the operation of RAT. It does not directly orchestrate the operation of the other components, rather it simply turns them off or on, and triggers their operation. It maintains a status register (SR), program counter (PC), activates the BLE transmitter/receiver and encoder, and sends trigger signals to the drivers to perform logic in the computation arrays.

5.4.2 Operating Semantics

Now that we have described the hardware components of RAT, we describe how they work together and how they tolerate intermittency. The state transition diagram is shown in Figure 5.8. RAT has efficient and fine-grain checkpointing mechanisms in the computation phase, due to the inherent resilience of non-volatile MTJ based memory to power interruptions. RAT has less efficient checkpointing mechanism in the other phases, where more progress will be lost in the event of a restart.

Data Reception and Transmission (I/O)

Due to intermittent power, RAT cannot rely on continuous communication. To avoid complexity, we chose for RAT to operate under a simple protocol. When RAT is not busy (there is no input being processed), it is open to new input. RAT will wait for new input from FLY. After fully receiving input data, RAT enters the encoding phase and will no longer check for or acknowledge new inputs. Only once computation is complete will RAT re-activate BLE in order to transmit outputs.

When receiving inputs, the data is stored into a dedicated non-volatile buffer, as shown in Figure 5.3. The size of the input data is set, so RAT will know the number of packets it needs to receive. There is a dedicated region of memory for each packet, and there is a *valid bit* for each packet. RAT will set all valid bits to 0 prior to reception. The valid bit for each packet is set strictly after the packet has been written into memory. If RAT looses power after the packet is written but prior to setting of the valid bit, the packet is considered invalid and will need to be re-transmitted . Once all valid bits are set (all packets have been received), RAT will set a *completed* bit, which acts as a signal to RAT's controller to move to the next stage.

During reception and transmission, no other components of RAT are in use. The controller will stall all operation until data transfer is complete.

Encoding

RAT uses specialized hardware accelerators for encoding [368]. To make the encoding process intermittent safe, we require it to be *atomic*. It reads input from the dedicated non-volatile buffer, performs encoding in full, and then writes output into the first column of computation arrays, shown in Figure 5.3. Due to being atomic, if it is interrupted at any stage it restarts from the beginning. This guarantees correctness on restart, as none of the inputs have been overwritten, however it can waste energy. There is significant room for optimization (i.e., adding efficient checkpointing mechanisms for the encoder), however, we do not focus on the encoder in this work. The vast majority of time and energy is spent on homomorphic computation, where we optimize the checkpointing process.

Since the data is written directly into the computation arrays, it will be ready immediately for processing once encoding has completed. Computation arrays can transfer the data where it is required via logic operations during the computation stage.

Computation

Computation takes the most time and energy, therefore it is critical to make it resilient to intermittent operation and energy efficient. Prior work has demonstrated that in-memory logic performed with MTJs is inherently intermittent safe [298]. We exploit this same property to enable high frequency and light-weight checkpointing during computation. However, unlike prior work, we do not need to maintain standard memory functionality. This enables us to remove much energy-inefficient hardware, including most of the sense amplifiers.

During computation, the controller will broadcast the PC value to all drivers. Each driver reads the corresponding instruction in its instruction memory, decodes it, then drives the appropriate signals to the computation arrays in the same column. A sufficient amount of time is allotted between broadcasts of the PC value such that every instruction is guaranteed to have completed. This process repeats until the program has finished or RAT runs out of power.

Write and logic operations with MTJs [222, 62, 399] remain correct when interrupted or performed multiple times [298]. This means that if the driver triggers a logic operation in the computation arrays, it can be interrupted

i. Before the operation begins

ii. In the middle of the operation

iii. After the operation has finished

and correctness will be maintained. The same logic operation (instruction) can be triggered again, and the output will be as if no interruption had occurred [298]. Hence, a single instruction represents an *idempotent* operation. This means that, after RAT has restarted, the controller can safely send the same PC value to the drivers and the instruction will be performed (or re-performed) and produce the correct output.

Due to this inherent resilience, correctness of data in the memory is easy to guarantee as long as we progress in program order by only a single instruction at a time. This means we need to checkpoint after every instruction, and not start the next instruction until the previous has been completed. Such frequent checkpointing may be inefficient for more traditional architectures [180, 219], however RAT can do this with ease. As the computation is occurring in (non-volatile) memory, all data backup happens automatically. All results are permanently stored after every instruction, regardless of the checkpointing strategy. Hence, RAT can perform frequent checkpointing with low overhead by simply tracking architectural state variables in the controller. For computation, the controller only needs to keep track of the valid PC value.

As noted in Section 5.4.1, which rows or columns are active is part of the architectural state. A single row and column from each computation array is dedicated to hold the bitmask for active columns and rows. Hence, the active rows and columns are protected by the same mechanism as all other data in memory. Being non-volatile, the bitmasks persist through power interruptions. However, the peripheral circuitry will need to be re-activated on restart. Dedicated instructions, *activate rows* and *activate columns*, restore the peripheral circuitry based on the bitmask values. These instructions are issued whenever the bitmask changes, the instructions switch between row-wise or column-wise, and on re-start.

An advantage of storing the active row/column bitmasks in the computation arrays themselves is that they can be set with standard write and logic operations. A disadvantage is that it introduces potential incorrectness. If a row- (column-) parallel instruction modifies the row (column) bitmask, the instruction relies on the current value of the bitmask. If this operation gets interrupted prior to completion of the instruction, some of the values in the bitmask may have changed. When the instruction is re-performed on startup, the modified value of the bitmask will be used instead. Our solution is to duplicate the bitmasks and maintain parity bits, row parity (RP) and column parity (CP), which indicate which bitmask is valid. Hence, two columns and two rows are dedicated for bitmasks. Instructions are only allowed to modify the currently invalid bitmask, leaving the currently valid bitmask unperturbed. After the invalid bitmask has been successfully modified, the corresponding parity bit can be flipped and the rows or columns re-activated.

Controller

RAT's controller is responsible for maintaining the state transitions depicted in Figure 5.8 and it holds the architectural state variables required for driving computation. It maintains a *status register* (SR), which indicates whether RAT is in the reception, encoding, computing, or transmission state. Additionally, it maintains the *program counter* (PC) which is sent to the drivers to initiate instructions during the computation state.

Given the relatively little information the controller must maintain, simple checkpointing strategies are sufficient. Both the SR and the PC are duplicated and have an ancillary non-volatile *parity bit*. The parity bit indicates which copy is valid, and it is flipped strictly after each variable is updated. Hence, a currently valid copy of either the SR and PC is never written to, which prevents potential corruption due to an interrupted write operation. Setting the parity bit is an atomic operation, the non-volatile MTJ holding the bit will either be successfully flipped or will remain in its old value (if interrupted during the write operation). If RAT restarts without flipping the parity bit, the old value of the SR and PC will be used. As explained below, this will waste energy but not introduce incorrectness. If RAT restarts after flipping the parity bit, effectively all progress has been saved and RAT will start where it left off. To ensure correctness of these mechanisms, RAT performs the states and instructions sequentially and strictly *in order*. There is no overlap of instructions, states of the controller, or updates of the architectural state variables.

During the transmission and reception phases, the controller hands control over to the BLE transmitter/receiver. An efficient protocol for intermittent safe transmission/reception is beyond the scope of this work – see Section 5.6 for a discussion. However, we note that strategies for noisy transmission with large amounts of packet loss [16] could likely be adapted for handling intermittency. RAT's controller will hand over control and the simply wait for the *completed* signal, in which case it will transition to the next phase. If interrupted prior to the arrival of the complete signal, RAT will again hand control over to the transmitter/receiver on restart. In the worst case, an entire data packet will need to be re-transmitted or received. Additionally, if transmission/reception have completed, but the completion signal was not sent prior to shut down, the controller will have to re-check the signal before progressing to the next stage.

The encoding process follows a similar strategy. During the encoding processes there are no checkpoints. Introducing checkpoints can guard against progress loss on restart, however, for this work we assume the encoding process must not be interrupted. Hence, if there is a restart in the encoding state, the controller will instruct the encoder to start from the beginning.

RAT has efficient and finely-grained checkpointing mechanisms in the compute phase. Due to the ideal properties of MTJ based memory arrays discussed in Section 5.4.1, RAT can easily checkpoint after every instruction [298]. The controller sends the instruction address to the drivers, which load the instruction and trigger it in the computation arrays. The controller waits a sufficiently long time to guarantee the completion of the instruction, after which it updates the PC and commits the instruction by flipping the non-volatile parity bit. As this checkpoint occurs after each instruction, at most one instruction needs to be re-performed in the case of a power outage. The logic operations performed in the memory are inherently idempotent [298], (meaning that they can be performed multiple times) and will produce the same result. Hence, the instruction can be interrupted at any stage in its progressing and we can safely restart it when the power is restored.

Table 5.1: SVM benchmarks used in this work and parameters used in RAT. The number of support vectors does not impact latency and energy as they are padded to 4096. For comparison, the accuracy of full-precision SVMs from libSVM [57] with unlimited support vectors is reported.

Benchmark	Input Bit Precision	Dimension (D)	# Support Vectors	RAT Accuracy	LIBsvm Accuracy [57]
MNIST	3	784	3841	93.85%	98.05%
HAR	3	561	2466	94.64%	94.1%
ADULT	3	14	596	76.00%	78.62%

5.5 Evaluation

We evaluate RAT performing one homomorphic SVM inference to assess the feasiblity. We can then compare the net impact on FLY, to see if the presence of RAT provides an improvement in performance. We analyze the scenario where inputs are not sensitive, however, the machine learning models that process them are proprietary, and are therefore sensitive. This means that FLY can transmit non-encrypted sensor data to RAT, however RAT must perform homomorphic inference and return encrypted results to keep the model secure.

For benchmarks, we use MNIST [199], Human Activity Recognition [14, 360], and ADULT [188]. These datasets are representative of the input sizes that will be expected for beyond edge devices. When performing homomorphic inference, the data size is determined by D, the dimension of the input sample sizes, and N, the length of our homomorphic ciphertexts. The D individual elements from each sample must be in separate ciphertexts, as each element must be summed together (elements within the same ciphertext cannot be added without expensive rotation operations [292]). As described in Section 5.3.1, our ciphertexts are fixed at $N = 4096$. The number of support vectors required for each benchmark fill the 4096 element ciphertexts, with the remaining elements padded with dummy data. Hence, regardless of the number of support vectors, each benchmark uses 4096 element ciphertexts. MNIST contains 784 elements (representing a 28×28 image), HAR contains 561 elements, and ADULT contains 14 elements. For all benchmarks, we normalized the input to fit within 3-bit integers (values 0-7). This lowers transmission cost and prevents arithmetic overflow during computation. We use customized SVM models which use only integer arithmetic and limited the number of support vectors to 4096. Despite these limitations, we were still able to achieve reasonable accuracy relative to full-precision SVMs provided by libSVM [57], which we accessed through R [288] with the ‘e1071’ package [249]. The SVM parameters and accuracies are listed in Table 5.1. Using integers allows us to use the BFV homomorphic scheme for integer arithmetic [43, 102], which has a lower overhead. We validated that homomorphic operation produces identical output by using Microsoft SEAL [321] to perform the multiplications and additions within the SVM.

MTJ based memory technology is already commercially available [1, 2], however, the devices are expected to significantly improve over the next few years. Hence, we evaluate RAT with two different MTJs models. A *modern* model, which uses parameters of MTJs which have already been demonstrated, and a *projected* model, which estimates MTJ performance within 3-5 years. These parameters are listed in Table 7.1.

We model the energy harvester as a constant power source, which fills a $1\,\mathrm{mF}$ capacitor (energy buffer) on the chip. In practice, tunable energy buffering systems such as Capybara [72] can be

Table 5.2: Parameters for MTJ devices.

Parameter	Modern	Projected
R_P	$3.15\,\mathrm{k\Omega}$	$7.34\,\mathrm{k\Omega}$
R_{AP}	$7.34\,\mathrm{k\Omega}$	$76.39\,\mathrm{k\Omega}$
Switching Time	$3\,\mathrm{ns}$ [307, 268]	1 ns [399, 167]
Switching Current	$40\,\mu\mathrm{A}$ [307]	$3\,\mu\mathrm{A}$ [399]

used. The power source increases the voltage on the capacitor up to a specified value, at which point RAT turns on and consumes energy until the minimum voltage on the capacitor is reached. RAT then shuts down and waits for the capacitor to recharge. We assume that RAT has access to sunlight, which can provide a relatively large amount of power for beyond edge devices. We test from $2\,\mathrm{mW}$ (0.02 cm^2 solar panel [180]) up to $100\,\mathrm{mW}$ (1 cm^2 solar panel [180]). It is desirable to match the voltage level on the capacitor to the MTJ technology used. Projected MTJs have a lower operating voltage and power draw than modern MTJs. For modern MTJs, the voltage fluctuates between $400\,\mathrm{mV}$ and $700\,\mathrm{mV}$ and for projected MTJs the voltage fluctuates between $100\,\mathrm{mV}$ and $575\,\mathrm{mV}$. As noted in Section 5.3.3, different logic operations require different voltages. We use switched-capacitor converters for upconversion and downconversion to provide all required voltages [173, 289, 148].

We generate latency and energy estimates of RAT with an in house simulator which accounts for the overhead due to MTJs (as listed in Tabe 7.1) and peripheral circuitry, which we extrapolate from NVSIM [90]. NVSIM gives us the relative share of latency and energy that peripheral circuitry will consume for non-volatile memories with the same size as our computation arrays. We clock RAT at $30.3\,\mathrm{MHz}$ with modern MTJs and at $90.9\,\mathrm{MHz}$ with projected MTJs. This gives more than sufficient time for the controller to broadcast the PC and for the drivers to finish logic operations in the memory. This clock rate is conservative, which leaves potential performance improvements on the table. However, since the performance of beyond edge devices will be limited by energy efficiency rather than performance [120, 121] ensuring completion of all instructions far outweighs any potential performance gains.

As discussed in Section 5.4.2, the logic operations in the memory are inherently resilient to interruption and we can checkpoint after every instruction with low overhead [298]. However, the reception, transmission, and encoding process do not have this benefit. For transmission and reception, we assume that a power interruption results in the re-transmission or reception of a single element of data. Elements transmitted prior would have been saved in non-volatile memory. For encoding, we assume that the *entire* encoding process must be restarted if interrupted. If interrupted once, RAT will re-attempt on restart. At this point, the capacitor will be fully charged and RAT will have sufficient energy to complete the process. For transmission cost, we assume $158\,\mathrm{pJ}$/bit [303] and for encoding we take latency and energy directly from Van der Hagen et. al. [368] who developed an encoder/encryptor which operates on ciphertexts of the same size as in this work – $0.3\,\mathrm{ms}$ and $60\,\mu\mathrm{J}$.

The latency of one inference (including transmission and encoding) is shown in Figure 5.9 for modern MTJs and in Figure 5.10 for projected MTJs. Due to high energy cost of homomorphic computation, the latency is quite high and increases dramatically as the power source reduces. For RAT to still demonstrate improvement, the latency on RAT must be less than the latency of local

Table 5.3: As described in Section 5.2, a beyond edge device can (1) Report all results to a remote server, (2) Perform processing locally, or (3) Offload computation to the nearby RAT. If FLY has the energy efficiency of the beyond-edge device SONIC [120] and is operating on a reasonable $60\,\mu$W (which can be harvested with thermal energy [180, 239]), options (1) and (2) result in the reported latencies. The energy per bit for transmission to the distant server is assumed to be $400\,\mu$J [40]. Also shown is the power required by RAT to achieve the fastest solution with option (3). Due to being more exposed, RAT should have access to tens of milliWatts of power [180]. *Results extrapolated.

	Option 1: Remote Server	Option 2: Local Processing	Option 3: RAT
Benchmark	Latency (s) $= E_{FT}/P_F$	Latency (s) $= E_F/P_F$	Minimum power for RAT to provide fastest solution (mW)
MNIST	15,680	450	3.36
HAR	11,220	208	4.28
ADULT	280	8.03*	11.29

processing on FLY and transmitting to a high-power distance server. For local processing, FLY can perform non-homomorphic inference, since FLY is in a secure location and can use non-encrypted ML models. SONIC [120], a beyond edge ML accelerator, requires $27\,$mJ to perform MNIST and $12.5\,$mJ to perform HAR. If FLY is operating on thermal energy (due to its deployment within the walls of a building, e.g.), we can assume that its power budget is roughly $60\,\mu$W [180, 239]. If FLY is as efficient as SONIC, it will take $450\,$s to complete MNIST, $208\,$s to complete HAR, and $8.03\,$s to complete ADULT. Hence, we can see that RAT can provide a faster solution than local processing on MNIST/HAR/ADULT if its power budget is $3.36\,$mW/$4.28\,$mW/$11.29\,$mW, even with modern MTJs. For sending data to a distant server, the cost of transmission is approximately $400\,\mu$J per bit. To acquire enough energy to transmit inputs to the server, FLY will take 15,680 seconds for MNIST, 11,220 seconds for HAR, and 280 seconds for ADULT – much higher than the alternatives. These results are summarized in Table 5.3.

Consistent with prior work [120, 298], latency is mostly determined by energy efficiency as the device is energy constrained. Consequently, projected MTJs, which are much more energy efficient, enable a significantly reduced latency. Due to the significant overhead of homomorphic computation, despite our lack of bootstrapping, RAT requires a substantial power budget to remain within reasonable latency constraints. Due to the efficiency of our checkpointing mechanisms, energy consumption is determined mostly by the length of the program, rather than the number of interruptions. The absolute energy consumption of SVM inference on RAT (evaluated at $2\,$mW) is listed in Table 5.4. To evaluate the efficiency of RAT relative to prior work, we compare polynomial multiplication (the core of homomorphic multiplication and the most energy intensive component of our benchmarks) with an X86 CPU, FPGA implementation [260], and a processing-in-memory solution, CryptoPim [259]. The comparison is listed in Table 5.5. As expected, RAT provides a significant advantage over the CPU and provides a better (yet comparable) efficiency to the FPGA and CryptoPim. It should be noted that both the FPGA and CryptoPim are optimized for *performance*, rather than energy efficiency. However, energy efficiency is the most important metric in the beyond edge domain [121], largely because performance is limited by energy efficiency [120]. Additionally, neither the FPGA or CryptoPim have been designed to guarantee correctness during intermittency. Adding tolerance to intermittent operation will come with a performance and efficiency overhead.

The impact of intermittent operation can be evaluated by examining the *dead*, *restore*, and *backup* overheads, for both latency and energy [312]. Dead refers to latency and energy dedicated

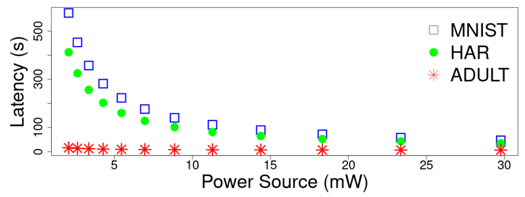


Figure 5.9: Latency of homomorphic SVM inference with modern MTJs using different power sources.

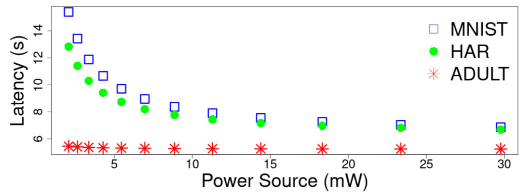


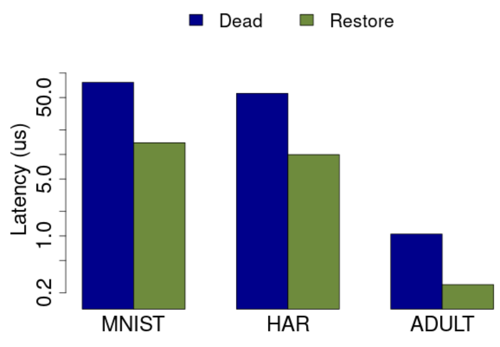
Figure 5.10: Latency of homomorphic SVM inference with projected MTJs using different power sources.

Table 5.4: Energy consumption of homomorphic SVM inference on RAT for different benchmarks.

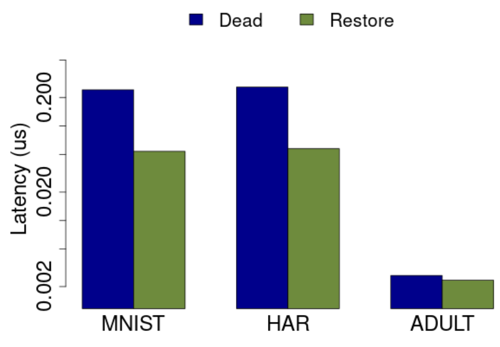
Benchmark	Energy (μJ)
MNIST	1,188,716
HAR	851,282
ADULT	31,736

Table 5.5: Energy consumption of polynomial multiplication (developed by Nejatollahi et. al [260]) on RAT and related work. N is the polynomial size and b is the bitwidth.

Architecture (N,b)	Energy (μJ)
X86 (1K,16) [259]	2483.77
X86 (4K,32) [259]	10864.64
FPGA (1K,16) [260]	12.52
CryptoPIM (1K,16) [259]	11.04
CryptoPIM (4K,32) [259]	178.62
RAT (1K,16)	9.68
RAT (4K,32)	54.65



(a) Modern MTJs

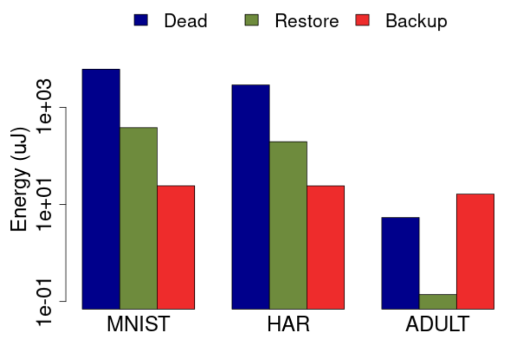


(b) Projected MTJs

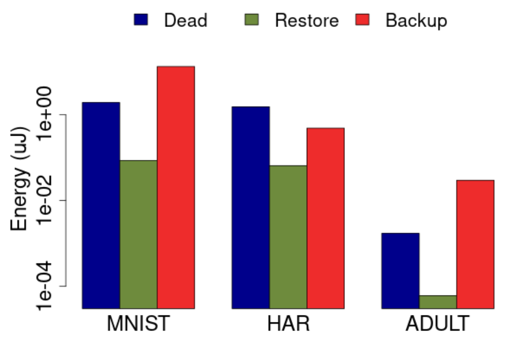
Figure 5.11: Latency overhead for correctness during intermittent operation.

to re-performing operations after restart which were already performed before shutdown. This accounts for wasted operations – which completed but the results of which cannot be used. For RAT, this comes from re-performing that last in-memory instruction, re-transmitting or receiving a packet of data, and re-encoding input. Restore refers to overhead associated with restarting the device, getting it back into working order after a power outage. For RAT, this is the re-activation of rows and columns of the memory arrays. Backup is any operation performed in order to save state prior to shutdown. This typically involves saving the architectural state and storing volatile data to non-volatile memory. As RAT performs all computation in the memory, data backup occurs automatically. Hence, the only backup cost for RAT is saving the architectural state, setting the SR, PC, and parity bit. The latency overhead for each of these is shown in Figure 5.11 and the energy overhead is shown in Figure 5.12. The shown overheads are evaluated at $20\,\mathrm{mW}$, the lowest power considered, which results in the maximum number of restarts, and therefore, the maximum overhead.

Overheads for projected MTJs are less because RAT will have to restart less often due to their greater energy efficiency. Dead energy involves re-performing instructions (which typically involve performing many parallel logic operations). Overall, projected MTJs are much more efficient than modern MTJs. Dead energy is 0.2889% of the total for modern MTJs and 0.0040% for projected MTJs. The restore energy is also much lower for projected MTJs, as it is also incurred by only restarts where the peripheral circuitry needs to be reset. Restore energy is 0.0185% of the total for modern MTJs and 0.0002% of the total for projected MTJs. As it involves only updates to architectural state variables, the backup energy tends to be much lower. The backup energy is



(a) Modern MTJs



(b) Projected MTJs

Figure 5.12: Energy overhead for correctness during intermittent operation.

0.0187% of the total for modern MTJs and 0.0147% of the total for projected MTJs. Backup energy is incurred on every instruction, so it is not largely impacted by the number of restarts.

Backup has no associated additional latency as it occurs during the execution of each instruction. The dead and restore latency are functions of the number of restarts. Dead (restore) latency is 0.0006716% (0.0001209%) of the total compute latency for modern MTJs. For projected MTJs the dead (restore) latency is a negligible 0.0000028% (0.0000006%).

5.6 Limitations and Potential Improvements

In this section we discuss the limitations of RAT and potential improvements. While relevant, these extensions are beyond the scope of this paper or are left for future work.

5.6.1 Communication Protocol

In this work we use a simplified communication scheme, where FLY and RAT transfer data back and forth. There are a number of complicating factors which can disrupt proper operation. FLY may begin transmission but then loose power for an extended period of time. In this case, RAT would remain in reception mode, spending its entire time waiting for input that does not come. RAT should contain some method of cancelling a reception and revert to an idle and available mode. While a variant of watchdog timer can be deployed to this end, data transfers during intermittent operation pose a considerable challenge. Additionally, if RAT goes without power for an extended period of time, it may work on input which is old and no longer relevant. Keeping beyond edge devices working on relevant tasks across time is an important problem [344].

5.6.2 Communication Energy and Deployment

Communication between FLY and RAT will only be efficient if they can be deployed in close proximity. Bluetooth Low Energy (BLE) only allows for communication within tens of meters. At further distances, they must resort to long-range (LoRa) technology [40], which can undo the benefit RAT provides.

5.6.3 Power Limitations

Homomorphic computation, even without bootstrapping, remains an energy costly process. Even with the efficient computation enabled by RAT, collecting enough energy beyond edge remains a challenge. Even simple computations will take a long time, which may be insufficient for a number of applications. Additionally, as solar power is the only power source which reaches the required levels, devices such as RAT would struggle to function at night. Further methods to increase energy efficiency may be necessary for practical implementation. A possible solution is implementing *fleets* of beyond edge devices [85, 86]. As such, each individual device could compute slowly, but a high throughput could be achieved with their combined effort.

5.7 Related Work

Power delivery is particularly challenging for beyond edge devices. Tunable systems like Capybara [72] have been designed to optimally store and deliver energy. We assume RAT relies on such a system.

Many low power processors and ML accelerators exist [324, 216, 75, 379, 172, 403, 366], however, these architectures do not guarantee correctness during intermittent operation. Adding such guarantees adds a large performance and efficiency overhead. Numerous beyond edge devices have been designed to function correctly in intermittent contexts [219, 231], including ML accelerators [120]. These devices have more traditional architectures, which have been augmented with nearby non-volatile memory for fast backup operations. Much research has been dedicated to tolerating intermittency for more general purpose architectures, including lowering checkpointing overheads and clever methods of detecting power outages [68, 305, 113, 290, 67, 232, 156, 369, 157, 22, 170, 15, 21, 31, 218, 228] along with modifications of software [233, 227]. RAT's advantage over these designs is significantly more energy efficient in-memory logic and simplified checkpointing mechanisms.

Resch et. al. [298] have proposed an in-memory intermittent safe accelerator. However, the architecture maintains standard memory format, which adds unnecessary space and energy overhead, and is capable only of 1D computations (only along rows or only along columns). In contrast, RAT's architecture is capable of 2D computations (similar to a crossbar, along rows and columns, but digital) and is tightly tailored to homomorphic SVM inference. Mapping homomorphic inference onto [298] would incur a large communication (read/write) overhead due to the required intra-array data movement. ResiRCA [287] uses in-memory computation to accelerate parts of ML inference and adapts the amount of parallelism to match the amount of harvested energy. However, they rely on a battery to maintain a controlling CPU.

Significantly, none of the prior work mentioned thus far has accelerated homomorphic inference, which is necessary to maintain security in un-trusted environments. To the best of the author's knowledge, no beyond edge homomorphic accelerator exists. However, much prior work has been done on accelerating homomorphic computation with continuous power. This includes software optimization [292], FPGA implementations [260, 359] and processing-in-memory implementations like CryptoPim [259]. CryptoPim developed strategies to minimize latency and energy overhead for NTT operations, the core of homomorphic multiplication. While CryptoPim is not designed to handle intermittent operation, we do use strategies they developed, such as efficient modulus operations, in our work.

Efficient encoding and encryption is required to make homomorphic computation feasible in edge or beyond edge domains. Van Der Hagen et. al. [368] developed energy efficient accelerators to this end.

5.8 Conclusion

Beyond edge devices operate on harvested energy, significantly increasing deployment capabilities and widening the potential applications. However, these devices are by construction extremely limited by the power available. In this paper we propose a beyond edge, intermittent safe accelerator which can compute homomorphically, and which can be used as a local server to offload computation. The deployment capability of this accelerator can significantly reduce the communication cost for other beyond edge devices to offload computation. The accelerator is capable of maintaining security by performing homomorphic computation and its unique architecture enables this to be done within a reasonable power budget.

Chapter 6

On Endurance of Processing in (Nonvolatile) Memory

6.1 Introduction

The performance of modern computing systems is limited by the performance of the memory. This is because CPU performance has increased more rapidly than the performance of memory for the past few decades [152]. This limitation is referred to as the *memory wall*, and it has created the search for new architectures which do not suffer from this problem. A promising candidate is processing-in-memory (PIM) which can perform computation directly in memory. PIM architectures can enhance both performance and energy efficiency of numerous emerging applications significantly [325]. Non-volatile PIM architectures are of particular interest due to their extreme energy efficiency [210, 195, 62] and high density [301, 399].

Unfortunately, nonvolatile memory (NVM) devices suffer from a low endurance. The devices can only be written a certain number of times before failing. The endurance varies significantly between technologies, but all are at risk of pre-mature failure which would render them impractical. Hence, there has been much research into performing load-balancing (i.e., evenly distributing write operations across NV memory cells), when being used for standard memory purposes [146]. However, PIM significantly changes the access patterns and drastically increases the number of write operations. For example, an in-memory multiplication can result in *over 150 $\times$ more write operations* than it would in a standard architecture. Hence, strategies which are sufficient for NVM have to be revisited for nonvolatile PIM (NVPIM).

In this work, we investigate the write operation cost of PIM applications and estimate the corresponding expected lifetime of PIM arrays. We test basic strategies to mitigate this problem and show their effectiveness. Our findings suggest that NVPIM will still be significantly limited by endurance, emphasizing the need for progress at the technology level.

6.2 Background

In this section we cover NV devices used for PIM, how logic operations are performed in memory, and how we can synthesize complex computations using these logic operations.

6.2.1 Nonvolatile Devices

In this paper, we consider nonvolatile memories which hold their state in their resistance. The state of a device can be determined by applying a voltage and sensing the current that flows through it, which constitutes a read. The specifics for changing the state, i.e., the write operation, varies between technologies. We will next briefly cover most promising representatives.

Magnetic RAM (MRAM) is based on Magnetic Tunnel Junctions (MTJs) which have two magnetic layers, a fixed layer and a free layer. If the layers are aligned, the MTJ has low resistance (parallel state); if not aligned; high resistance (anti-parallel state). The state can be changed by driving a current of sufficiently high magnitude through the MTJ, where the direction of the current determines the state. When electrons flow from the free (fixed) layer to the fixed (free) layer, the MTJ is put into the anti-parallel (parallel) state. The main advantages of MTJs are relatively high density and high endurance with respect to other nonvolatile technologies. Specifically, when it comes to endurance, MTJs can switch as many as 10^{12} times [251, 330] until permanent failure. A disadvantage of MTJs is that the resistance difference in the anti-parallel and parallel states is relatively low, which makes them more sensitive to noise such as voltage fluctuations.

Resistive RAM (RRAM) consists of a metal-insulator-metal stack [136]. Applying a voltage differential causes the formation of a conductive filament, creating a low resistance state. Apply a voltage differential in the opposite direction removes this filament and creates a high resistance state [12]. RRAM has more than 2 possible states, as it can take on a range of resistance values between the two extremes. However, it is common in practice to use only the highest and lowest resistance states to reduce noise. An advantage of RRAM is the high ratio between the high and low resistance states, reducing sensitivity to noise. However, a major drawback is limited endurance, with approximately 10^8 writes possible before failure [347].

Phase-Change Memory (PCM) has a state based on the structure of the atoms within the channel for electric current [387]. The write operation involves heating up the device by passing an electrical current through it. Quickly reducing the current causes the device to cool rapidly into an amorphous state, which has high resistance. Cooling the device slowly allows it to form a more uniform structure, which has low resistance. PCM also suffers from a very limited endurance, currently ranging from 10^6 - 10^9 writes before failure [179].

6.2.2 PIM Architectures

PIM architectures modify traditional memory hardware to enable logic operations to occur either within or very near the memory. In this work, we consider architectures which perform Boolean (digital) logic directly in memory[1], where the input and output bits of every operation are memory cells in

[1]In contrast to analog architectures which are specialized accelerators, and typically do not maintain standard memory operation.

the array. Such architectures maintain the basic memory hardware and operating semantics, allowing them to replace standard memory structures with little modification to the overall architecture. Additionally, hardware modification to the memory array itself is relatively modest. Typical solutions involve adding additional bitlines [301, 298], row-decoders which support multi-row activation [210, 325], or extra transistors in each cell [62, 399]. These architectures have demonstrated high performance and energy efficiency both for use in traditional computing systems [210, 325, 301] and in embedded systems [298, 300] for commercially relevant applications, including machine learning inference.

Regardless of the specific approach, the operations on the memory devices themselves are nearly identical. Current is passed through one or two input memory devices, and a single output memory device is written to. As this paper discusses the endurance of PIM architectures, we can abstract out the specific cell design. In the remainder of this section, we discuss PIM operating semantics accordingly.

Basic Logic Operations (Gates)

PIM architectures enable logic at the bit level in the memory. Basic logic operations –such as NOT, (N)AND, or (N)OR)– take one or two memory bit cells as input, compute the output, and store the result in another bit cell, which is typically in the same row or column as the input bit(s). Depending on the architecture, this can be done with or without the involvement of sense amplifiers. For architectures using sense amplifiers [210], the procedure is:

1. Read multiple input cells simultaneously.
2. To calculate the output according to the underlying truth table, perform thresholding using the sense amplifier.
3. Write back the result to the designated output cell.

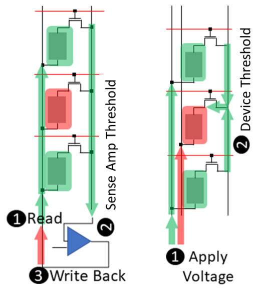
For architectures which do not use sense amplifiers [301], the procedure is:

1. Apply a voltage differential on the bitlines connecting the inputs and outputs.
2. Current travels through the inputs to outputs, conditionally switching the output according to the truth table of the operation being performed.

Both approaches are shown in Fig. 6.1, for column based computations without loss of generality. Regardless of the approach, the input cells effectively go through *read* operations and the output cell goes through a *write* operation. If the possible logic operations form a universal set, any computation can be carried out in the respective row or column, limited by the number of memory cells available.

More Complex Logic Operations

In traditional architectures, an arithmetic logic unit (ALU) can be used to perform complex logic operations relatively quickly. For example, addition and multiplication can be performed within a few cycles of the system clock. In contrast, PIM architectures require a series of logic gates to perform such operations.



(a) With SA (b) W/o SA

Figure 6.1: Column based PIM logic approaches. Input are shown in green; the output, in red respectively.

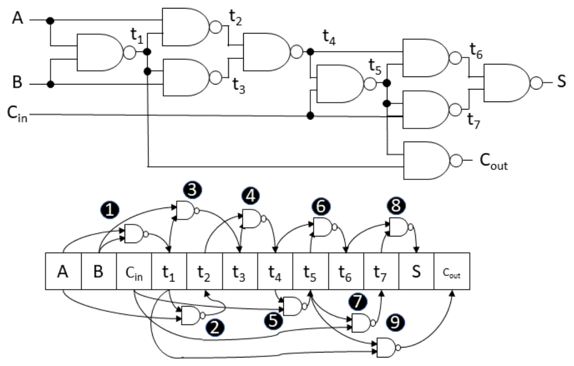


Figure 6.2: Full-Adder circuit and equivalent in-memory implementation.

The operation must be decomposed into a set of gates the architecture is capable of (e.g., NOT, AND, NAND). Each of these gates must then be scheduled within the array. When processing within a single column (or row), only a single logic gate can be performed at a time due to *structural hazards* – the hardware used to perform logic is shared by all cells in the column (or row). Hence, even if gates are logically independent (i.e., no *data hazard* applies) they must still be performed sequentially[2]. For example, a full-adder can be implemented with 9 NAND gates, taking 9 time steps, as shown in Fig. 6.2. Hence, optimizing both the latency and energy of a PIM computation (within a single row or column) involves finding the decomposition which requires the fewest logic gates in time and space.

b-bit addition can be done with a ripple-carry adder with $b-1$ full-adds and 1 half-add. Note that, while it is slow in traditional digital circuitry, a ripple-carry adder is optimal for PIM as it uses the fewest gates (which must be performed sequentially). A DADDA multiplier [358], on the other hand, can perform b bit multiplication with b^2-2b full-adds, b half-adds, and b^2 AND gates.

While specifics vary across architectures, our discussion so far covers much prior and state of the art work, as listed in Table 6.1. Here we list each design with the original memory technology used, but for most of these architectures, different memory technologies (MRAM, RRAM, PCM) can be exchanged for one another and the basic operating principles would remain the same.

[2]There are PIM architectures that are exceptions to this [140], however, they require additional transistors which significantly increase complexity.

Name	Parallelism	Memory Technology
Pinatubo [210]	Column	PCM
MAGIC [195]	Row and Column	RRAM
MAGIC [223]	Column	MTJ
Felix [140]	Row and Column	RRAM
CRAM 2T [62, 399]	Row	MTJ
CRAM 1T [301, 298, 65, 300]	Column	MTJ
CRAM [63, 401, 150]	Row	SOT-MTJ

Table 6.1: Architectures which perform logic gates in memory and follow the operation principle considered in this paper.

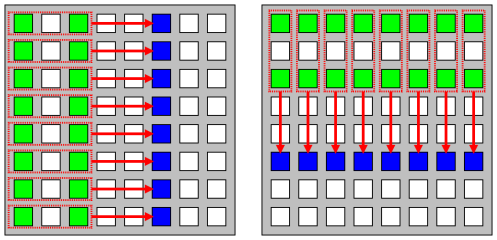

(a) Row-Parallel (b) Column-Parallel

Figure 6.3: Parallel two-input logic gates in row- and column-parallel architectures. Inputs shown in green (light) and the output in blue (dark).

Parallelism

The sequential nature of logic operations described in Section 6.2.2 leads to a high latency for any single operation. However, it is compensated for by high degrees of parallelism. PIM architectures are capable of much higher degrees of parallelism than other architectures, including GPUs. Potentially as many operations as the number of rows (or columns) within an array can be performed at the same time. This is because while a row- (column-) parallel PIM architecture can only perform one operation in each row (column) at a time, the same operation can be performed in many rows (columns) simultaneously. Whether parallelism comes from the rows or the columns depends on the architecture, both kinds are shown in Fig. 6.3. Within a single array of a row- (column-) parallel architecture, operations can be performed at the same time if:

1. They are the very same logic operation.
2. The input(s) and the output are in the same columns (rows).

For example, a common memory array dimension is 512×512, which would allow for 512 parallel operations. Additionally, PIM architectures allow for array level parallelism, as independent logic operations can be performed in different arrays. Hence, the limiting factor for PIM performance at scale is the number of arrays, the energy efficiency of the operations, and the overhead for any communication between arrays.

Row-parallel and column-parallel architectures are logically equivalent, except that in row- (column-) parallel architectures the logic operations run in parallel with (perpendicular to) the read and write operations. While these differences can largely be accounted for with different data layout optimizations, row- and column-parallel architectures place different constraints on the possible optimizations.

In the following, we will use the word ***lane*** to refer to the collection of cells (either in a row or a

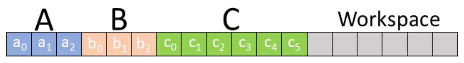


Figure 6.4: Cells within a lane of a PIM array are dedicated to inputs (A, B), outputs (C), and temporary workspace for the multiplication of two 3-bit integers.

column) which can work together to perform computation. For column-parallel architectures, a lane is a single column; and for row-parallel architectures, a single row.

6.3 PIM Data Layout and Operation

In this section we provide an overview of PIM operation, the additional challenges it introduces for performance in the face of endurance, and potential mitigation strategies.

6.3.1 Data Layout for Computation

Data layout design is a critical task for PIM architectures. It has a significant impact on the latency and energy of the application and the endurance requirements of the memory devices. Lanes within a PIM architecture can only compute on values contained within them, thus all data needed for each computation must be written first. Any data which cannot fit into the lane contributes to additional communication (read and write) cost.

Typically, a single, primitive operation is mapped onto a single lane. Data is aligned in many lanes, allowing many independent operations to proceed in parallel. Each lane must then contain cells dedicated to the following:

1. Input Data
2. Output Data
3. Temporary Workspace

For example, a single integer multiplication, $A \times B = C$ can be mapped to each lane as follows. Initially, space for the bits for A and B are allocated and the corresponding values written into the lanes. Commonly, one of the operands is static (e.g., the weights of a neural network) and the other is not (e.g., new inputs to a neural network layer). A number of logic gates are then performed in order to produce the product (output). These intermediate logic operations require storage for their outputs and temporary scratch bits. We call this storage the *workspace*. The minimum size for the workspace depends on the algorithm. However, when fully consumed, the workspace needx to be reset (overwritten or re-used) to enable further computation. Once computation is complete, the product C becomes ready in some set of cells, where they are available to be read out or used in further computation. This layout is shown in Fig. 6.4.

Effectively, all PIM architectures with the compute capability discussed in Section 6.2.2 follow this format. Large scale applications, such as convolutional or fully-connected neural networks, are decomposed into multiplications, additions, and subtractions which are performed within the lanes of the array. Different applications will result only in different data layouts and transfers between the computations.

6.3.2 Large Application Mapping

Embarrassingly parallel operations can easily be mapped onto lanes. A PIM architecture can map independent operations to each lane, and use all lanes performing the same logic gates on different data. For example, element-wise multiplication of vectors can be mapped in this way: N-element vectors can be multiplied with N multiplications performed in N lanes[3].

However, many applications are not so easily mapped. For example, an N-element dot-product initially requires the same N parallel multiplications. But then all products must be added together to produce the final sum. This requires read and write operations to move bits scattered across parallel lanes into the very same lanes. This mapping process is the subject of much prior work [301, 210], and it typically involves complex optimization. A wide range of applications can be effectively mapped to PIM , and only those which can exploit a high degree of parallelism will be performant [301].

6.4 Endurance Limitations

While nonvolatile PIM architectures provide high performance and extreme energy efficiency, a major problem they face is limited endurance. The memory devices used will break down after a certain number of write operations. While this is a well studied problem for nonvolatile memory, it has not been addressed for nonvolatile PIM.

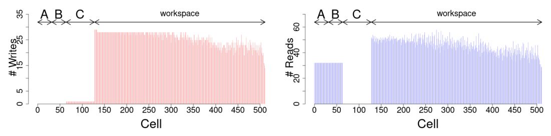
PIM architectures have a significantly different access pattern on the memory cells than standard memory architectures. Hence, they come with different demands on the memory technology. First, PIM uses many more reads and writes to perform the same computation than a traditional architecture (featuring separate memory and logic blocks), which taxes the devices much more. For example, a 32-bit multiplication on a standard architecture would involve reading two 32-bit numbers, performing the multiplication using an ALU, and then writing a 64-bit (to maintain the full result) number back to memory. In total, there are 64 cell reads and 64 cell writes. Assuming 1024 NVM cells are available to facilitate this computation, this is an average of 0.0625 reads and writes per cell. In a PIM architecture, using an in-memory DADDA multiplier [358] as a representative example, the same multiplication requires 9,824 in-memory gates, which incurs 9,824 cell writes and 19,616 cell reads. This produces an average of 19.16 reads/cell and 9.59 writes/cell. Hence, PIM architectures tend to burn through the endurance of NVM *much* quicker.

An upper limit on the lifetime of a memory array can be quickly calculated. Let us assume that each memory cell is capable of 10^{12} writes before failure [251, 330], which is optimistic for MTJs and well beyond what RRAM can support. A 1024×1024 array is capable of a total of $1024^2 \times 10^{12}$ writes before failure. Using the previous multiplication example, under perfect load balancing the array can perform a total of:

$$\frac{1024^2 \times 10^{12}}{9824} = 1.07 \times 10^{14} \tag{6.1}$$

32-bit multiplications before total failure of all cells. At full utilization, all 1024 columns computing in parallel, and operating at a reasonable switching time per gate of 3ns [301, 309], total failure (until

[3]Assuming there is a sufficient number of bits in each lane to complete computation. Practical array sizes (256×256, 512×512, 1024×1024) can easily accommodate multiplication of 64-bit operands.



(a) Write Count (b) Read Count

Figure 6.5: The read and write operations per cell in a row is heavily imbalanced. Workspace cells are used many times in order to produce a single result.

every device breaks down) takes:

$$\frac{1024^2 \times 10^{12}}{1024 \times \frac{1}{3\times10^{-9}}} = 3,072,000\ seconds = 35.56\ days \tag{6.2}$$

Under practical conditions, a small percentage of failed devices will cause incorrect operation, so effective failure will occur much sooner. Using RRAM endurance of approximately 10^8 writes, failure occurs in *just over 5 minutes*. Hence, physical properties of devices are currently limiting practical usage. Device properties will undoubtedly improve, and, clearly load balancing strategies are going to be critical in order to extend lifetime as much as possible.

Standard memory architectures may have an imbalance in reads and writes by writing or reading some rows more than others. PIM architectures introduce another opportunity for such imbalance by performing logic gates in some columns more than others. Returning to the 32-bit multiplication example, we can stick to a constant data layout similar to that shown in Fig. 6.4. This approach particularly leads to a large imbalance in the usage of each cell within the lane, as shown in Fig. 6.5. Specifically, the cells dedicated to workspace are used much more frequently, resulting in a significant load imbalance. This imbalance can result in some cells failing significantly sooner than others. Hence, load balancing is even more critical in maximizing the lifetime in this case.

6.5 Load Balancing

Given that PIM puts higher demand on the endurance of nonvolatile memory, it is essential to mitigate this problem. In this section, we cover basic load balancing strategies to increase the lifetime of PIM arrays.

The imbalance of cell usage noted in Section 6.4 will cause some memory cells to fail much more quickly than others. *Load balancing* is a well-known strategy to extend the lifetime of non-volatile memories [146]. Load balancing, which can use either hardware or software, involves distributing the write operations as evenly as possible to all memory cells. This attempts to prevent some cells from failing prematurely due to excessive use.

Load balancing strategies can also be applied to PIM. However, balancing the logic operations is much more critical than balancing the standard memory write operations. Hence, designing effective load balancing for PIM architectures is inherently a different challenge, and prior strategies (targeting memory functionality only) will not be effective.

6.5.1 Software Load Balancing

Software load balancing refers to changes made to the program in order to evenly distribute the write operations. The benefit of software approaches is that they do not require any hardware mod-

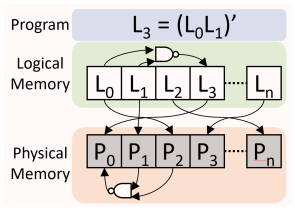


Figure 6.6: Logical to physical mapping of bits via software can arbitrarily remap logic operations.

ifications to the architecture, which can cost extra energy and increase latency for every operation. Significantly, software strategies have a greater ability to re-distribute write operations by arbitrarily modifying the program. However, a significant drawback of software approaches is that they require either knowledge from the programmer or compiler support. Additionally, software strategies may require periodic re-mapping (re-compilation) in order to be effective, which will consume energy and latency.

Balancing Logic within Lanes

Fig. 6.5 shows that some cells within a lane are used more often than others in a single computation. Notably, cells which hold the input and output are used much more frequently than cells that are used for temporary workspace. If this imbalance of usage persists for long periods of time, the workspace cells will fail much sooner. Hence, it is desirable to allow cells holding inputs and outputs to also be used as workspace.

An optimized operation (e.g., multiplication or addition) uses the same number of gates with the same inputs and outputs each time; it is constant. However, the individual logic gates that the operation is composed of can have the inputs and the output anywhere within a lane, i.e., the operations can be *re-mapped* by modifying their (e.g., row) *addresses*, while leaving the logical result unmodified. Hence, the gates can be arbitrarily rearranged to balance writes within the lanes (e.g., columns). It is conceptually easy to implement fine-grained re-mapping in software by maintaining a *logical* to *physical* mapping for each bit. A program operates on logical bits and remains constant. However, the logical to physical mapping can be changed periodically, arbitrarily re-mapping the operations. This process is shown in Fig. 6.6. The logical to physical mapping can be changed periodically, which can level the write distribution.

Changing the logical to physical mapping at **random** is highly effective at load balancing. Unfortunately, this solution has a significant drawback for row-parallel architectures, where the lane is orthogonal to the read/write operations. For example, originally, a 32-bit variable may reside in consecutive bits in the first 4 bytes of a row (lane). A row parallel architecture can access this variable in a single cycle with a standard read or write operation. However, re-arranging the logic operations causes the bits of the variable to be spread out in different bytes across the lane. Hence, many more bytes may need to be accessed in order to read or write the variable. Additionally, when the bits are read out of the array, they can be in any permutation. Hence, external hardware must re-order the bits before being used. Hence, re-ordering bits *within* the memory array for load balancing can add complexity to the hardware *external* to the array. This limitation is not as much of an issue for column-parallel architectures, where the lane is in parallel with the memory access operations. Hence, the bits of all variables must be read and written sequentially, regardless of their location.

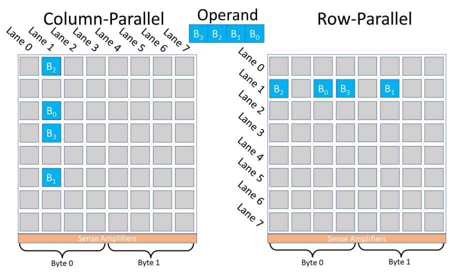


Figure 6.7: Re-arranging the bits of an operand within a PIM lane can disrupt memory operations. For row-parallel architectures, bits of the operand can be read in parallel, but they will be out of order and potentially in different bytes. Column-parallel architectures must read bits out of the lane sequentially, hence they are less impacted by such re-arrangement.

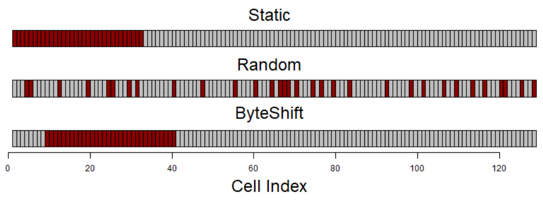


Figure 6.8: Representative placement of a 32-bit operand within a lane for different re-mapping strategies. Mapping takes place at compile time.

Hence, re-ordering does not significantly impact the read/write efficiency of column-parallel architectures. This property is shown in Fig. 6.7.

A strategy to utilize logical to physical re-mapping, while also maintaining ideal memory access patterns, is to simply periodically *shift* the address. To maintain proper read and write operations, the shift should be by an integer number of bytes (for byte-addressable memory), hence we refer to this strategy as **byte shift**.

Re-mapping logical to physical addresses, whether randomized or shifted, has the advantage of no overhead during execution of the program. However, both require periodic re-compilation in order to balance load. ByteShift is more friendly towards memory operations, as it maintains the order and coherency of bits in the memory. The different re-mapping strategies are depicted in Fig. 6.8.

Balancing Logic Between Lanes

Just as with in-lane usage, usage across lanes can vary as well. However, the causes of imbalance are different. Imbalance in lanes comes from the mapping of individual logic operations, which can be re-mapped at a fine granularity. In contrast, imbalance between lanes is more a function of the application, and usage in different lanes cannot be re-mapped at a fine granularity.

Imbalance between lanes typically appears when results from many lanes need to be combined. For example, an N-element dot-product initially requires N parallel multiplications, which can be balanced on N lanes. But then all products must be added together to produce the final sum. A series of memory operations must be performed to move data onto the same lanes, where they can be combined with addition. Hence, parallelism decreases as the application continues. By construction, such an application tends to use some lanes more than others. Lanes which perform the extra additions will wear out sooner.

Despite differences in the character of the imbalance, it can still be mitigated in software in the same way the imbalance within the lanes. Bit addresses can be periodically changed over time. A complete randomization of addresses is likely to produce the most optimal load balancing. However, a byte shift may be more desirable due to keeping addresses aligned for memory accesses.

6.5.2 Hardware Load Balancing

Hardware based balancing is also possible. In standard nonvolatile memory architectures, hardware based load balancing strategies typically involve estimating write counts and re-directing memory accesses accordingly [146]. However, such complexity is not feasible to implement at the level of a single PIM array. The main benefit of nonvolatile PIM is extreme energy efficiency. Hardware dedicated to balancing may easily become the bottleneck if it is not exceedingly lightweight. Maintaining counters to track writes at the bit-level within the array is unreasonable. Luckily, simple, albeit less effective, strategies do exist.

Hardware Balancing Within Lanes

We propose that the well-known practice of *register renaming* in traditional CPU architectures can be used to perform lightweight hardware load balancing. This process is similar to the software based logical to physical re-mapping, however it requires logical to physical translation at execution time. Additionally, it cannot arbitrarily re-map bits.

Hardware re-mapping requires a spare bit which can be used to swap logical addresses. For a lane with N physical bits, there are $N-1$ logical bit addresses and 1 free bit address. re-mapping can be applied whenever there is a write or a logic operation in all lanes. When a write operation is performed to bit address A in all lanes, the hardware re-directs the write to the free address, overwriting its contents. It then marks the free address as A, and sets the previous physical address of A as the free address. It is necessary that the operation writes to all lanes to fully transfer the intended contents of A to the new physical address. If not all lanes are in operation, the unmodified contents of A will be left at the previous address.

For architectures like Pinatubo [210] which perform computation with sense amplifiers, the output memory cell does not need to be preset. Re-mapping can occur entirely in-place. Bit re-mapping does not require any additional data transfers as the output of an operation is simply re-directed. Both writes and logic operations can be renamed without complication. However, for architectures like CRAM [62] which use device thresholding for logic, the output cell needs to be preset. For this architecture, an additional write operation will be required.

Hardware Balancing Between Lanes

In theory, the same re-mapping scheme could be applied between lanes, as well. However, this may not be possible depending on the architecture. For row-parallel architectures, memory operations can access the entire lane (row) at once. This makes lane re-mapping feasible by swapping row addresses on write operations. However, for column-parallel architectures, memory accesses can only access 1 bit from each lane at a time. Reading and writing an entire lane requires accessing each bit sequentially. Since a write operation only overwrites 1 bit in a lane at a time, a lane cannot

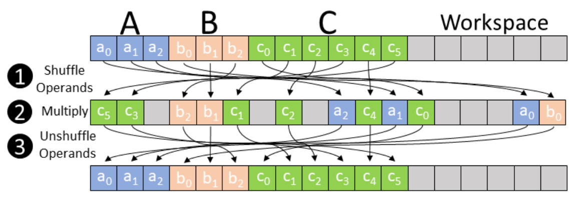


Figure 6.9: Additional COPY gates can be used to arbitrarily shuffle operands and output during execution. Initially, only A and B contain real data and need to be shuffled. After computation, only C contains necessary data and needs to be unshuffled. can make A, B, C... smaller to save space

be renamed with a single operation. A brute force approach could be applied, where a logical lane (column) is copied from one physical column to another over many sequential accesses. However, this is a complex and time consuming operation, and it not feasible to be implemented in hardware.

6.5.3 Memory Optimized Re-mapping

The load balancing strategies discussed so far will impact memory operations (read and write). It is possible to re-map computations but return data to the original addresses afterwards. This prevents any required changes to the read and write patterns. Prior to a computation, the input operands can be shuffled arbitrarily by performing COPY (or two sequential NOT[4]) gates. Then, the computation can proceed as normal, only with different column addresses for each gate. After the computation is finished, COPY gates can again be used to re-order the output to the expected location. This process is depicted for a multiplication operation in Fig. 6.9.

The significant drawback of this approach, contrary to previously discussed strategies, is that is requires additional logic gates to implement the shuffling. The relative overhead for this shuffling process depends on the operations being performed in memory. For a bit precision of b bits, shuffling will require $2 \times b$ COPY gates (or $4 \times b$ NOT gates) to move the two operands to their new locations. Note that the output and workspace do not need to be physically shuffled as they do not yet contain data, however, write operations may be required to pre-set them. The number of gates required to unshuffle the output depends on the output size. For multiplication, if roll-over is ignored, the output will have the same number of bits as the inputs. However, in some applications, it is useful to allow the output to expand to a higher bit precision. The is easily supported in PIM, so we consider that case here. For multiplication, the output has twice as many bits, so $2 \times b$ COPY (or $4 \times b$ NOT) gates will be required to move it back to the original location. In total, this additional overhead requires $4 \times b$ COPY (or $8 \times b$ NOT) gates. This overhead is small relative to the number of gates required for computation. A DADDA multiplier [358, 52, 291] requires $b^2 - 2b$ full-adds, b half-adds, and b^2 AND gates. Using 2-input logic gates, full-add requires a minimum of 5 gates and half-adds require 2 gates. Hence, a multiplication requires $6b^2 - 8b$ gates in total, and the relative overhead (additional number of gates) for shuffling becomes $1/(\frac{3}{2}b - 2)$. For 32-bit numbers, this equates to an extra 2.17%. The relative overhead for addition is much higher, due to the significantly lower complexity of the algorithm. Ripple-Carry addition (optimal for PIM) requires $b - 1$ full-adds and 1 half-add. The output is 1 bit longer, hence the shuffling cost is $3 * b + 1$. The relative overhead is $1/(\frac{5b-3}{3b+1})$. For 32-bit numbers, this equates to an extra overhead of 61.78%. Overheads for different levels of bit precision are listed in Table 6.2.

[4]Some PIM architectures do not natively support COPY [298] and will use NOT gates instead.

Table 6.2: Percentage of extra gates required by randomizing during execution. Overhead corresponds directly to extra latency and energy as all gates must be performed sequentially. Overhead doubles if using two NOTs instead of COPY.

Bit Precision	Dadda Multiplication Overhead (%)	Ripple-Carry Addition Overhead (%)
4	25	76.47
8	10	67.57
16	4.55	63.64
32	2.17	61.78
64	1.06	60.88

An additional drawback of this approach is that it is requires more complex software support. It still requires periodic changes to bit addresses. However, these changes cannot be performed by simply modifying a logical to physical mapping. Additional logic operations must be inserted into the program to perform the shuffling. This requires modification to the original program and full re-compilation. In summary, this approach can load balance without perturbing read and write operations, but it can add a significant logic gate count for some operations and requires more software support.

6.6 Using PIM Arrays with Failed Cells

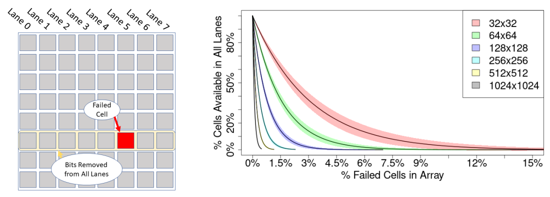
Given that nonvolatile cells are subject to failure, it is natural to ask if it is possible to effectively continue processing despite failed cells, aka graceful degradation. This is a possible solution, but due to unfortunate hardware and typical software characteristics of PIM computation, this is challenging to implement in practice.

PIM applications need to exploit parallelism to achieve high performance. This means that most of the lanes need to be in use simultaneously. PIM architectures require the input cells be in the same addresses within each lane. Hence, if a single cell fails in a single lane, all cells at the same address in other lanes cannot be used in parallel with this lane, as shown in Fig. 6.10a. In an $N \times N$ PIM array, there are N^2 cells which can fail, but only N cells in each lane. Hence, available space quickly reduces with failed cells, as shown in Figure 6.10b. The number of available cells quickly hits a point where multiplication is not possible at all due to insufficient space.

A workaround solution is to divide lanes into different sets, and to only use lanes in the same set in parallel. This does extend lifetime, by increasing the number of usable cells. However, it comes at a quickly increasing cost in latency, as different sets must run sequentially. Hence, even a few cell failures in a PIM array can significantly disrupt operation.

6.7 Evaluation Setup

PIM arrays are used to accelerate computational kernels where parallelism can be exploited. If located in an embedded device, the device will only function as long as the arrays persist. If located in a server, the accelerator needs to be replaced once a sufficient number of arrays have failed. Hence, we need to analyze how long a PIM array can function in order to estimate costs.



(a) A single cell failure removes bits from all lanes

(b) Percentage of bits in lane that can be used versus percentage of bits in array that have failed.

Figure 6.10: Failed cells quickly reduce the number of bits available in each lane. A single failed cell prevents the use of bits in all lanes with the same address.

6.7.1 Benchmarks

To be useful, PIM must be effective on application level computations. Large-scale applications must be broken down into computations that can be performed within the arrays of the memory. Memory arrays can process data independently, after which results are read out or combined with read and write operations.

Our analysis focuses on computations that can be performed within a single array, with the assumption that many such arrays perform similar work in parallel. Without loss of generality, we use three representative case studies which cover extreme ends of potential computations: *1) Embarrassingly parallel multiplications*, *2) Neural network (NN) inference*, and *3) Vector dot-products*. Embarrassingly parallel multiplication represents an ideal application for PIM. Each column of the PIM array can operate independently without any intermediate communication. Dot-products, on the other hand, represent the least ideal case. Results from all active columns must eventually be combined into a single output, which leads to maximum data communication. NN inference is a middle ground, where independent computations are too large to map to a single column, but do not require all columns to compute.

Clearly, we could choose more trivial or more complex computations. However, computations less complex than multiplication become trivial. Computations more complex than dot-product, on the other hand, require many more arrays than one, otherwise they are not suitable for PIM. Applications which do not have large degrees of parallelism or incur a high data communication traffic perform poorly on PIM architectures.

Embarrassingly Parallel Multiplication

The first benchmark is a simple parallel integer multiplication of 32-bit operands. A single multiplication is performed within each column (lane). There is no communication between lanes, and all lanes are utilized. Hence, there should be no imbalance between lanes. However, the multiplication algorithm (DADDA multiplier) may have imbalanced usage within each lane.

Dot-Product

A dot-product of two vectors A and B consists of an element-wise multiplication followed by a summation. If the vectors each have N elements, the final output can be written as:

$$C = \sum_{i=0}^{N-1} A_i \times B_i \tag{6.3}$$

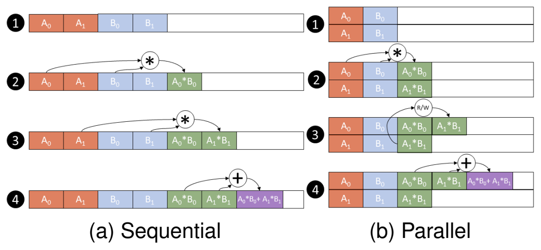


Figure 6.11: Vector dot-product mapped to a PIM array. a) Computation on elements stored in a single row can only proceed sequentially. b) Computation on elements stored in different rows can be done in parallel, but requires data transfer to extract the final result.

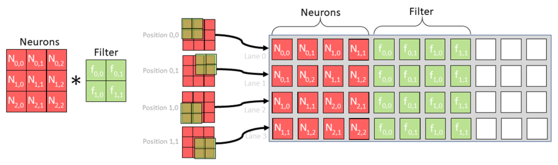


Figure 6.12: 2-dimensional convolution in a PIM array.

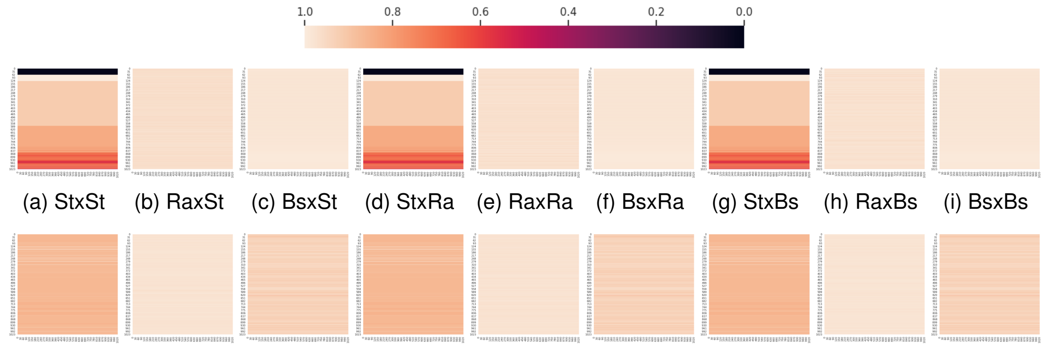
The dot-product can be performed using the multiplication and addition operations described in Section 6.3.1. Figure 6.11 shows two examples of a dot-product of two-element vectors performed in memory. Here, A_0 is multiplied with B_0 and A_1 with B_1, after which, the results are added. Fig. 6.11a shows a sequential computation within a single row; Fig. 6.11b, a parallel computation spread over two rows. In the parallel version the multiplications proceed simultaneously in separate rows. However, this requires an intermediate read and write data transfer to bring the results to the same row so that they can be added. In general, an N element dot-product having $2\times M$ elements in each row uses $ceiling(\frac{N}{M})$ rows. With large numbers of rows, a large number of reads and writes may be required to move intermediate results to the same rows. However, the logic operations dominate the latency, energy, and endurance cost. A single data transfer takes 2 sequential operations (read/write) and at most 1 read and 1 write per cell in a row. A multiplication takes over 20,000 sequential operations (logic gates) and requires roughly 40 reads and writes per cell (Section 6.4). Hence, regardless of the specific data layout chosen, the performance of dot-products within the memory is dominated by the latency and efficiency of the underlying technology's logic operations. For our benchmark we use 1024 element vectors with 32-bit operands.

Convolution

Inference applies a *filter* to a set of input *neurons*. The filter consists of a set of weights. The number of weights is typically small relative to the number of input neurons. The filter is "slid" over the neurons. At each location the weights of the filter are element-wise multiplied with the neurons they overlap with. The results of these multiplications are then summed together and finally become subject to some non-linear transformation (such as threshold, sigmoid, or htan). The neurons and filter can be 1-, 2-, or 3-dimensional. Typically they have the same depth (z dimension), and the filter is slid over the neurons in the x and y dimensions. Fig. 6.12 covers a 2-dimensional example, along with the corresponding data placement in a PIM array.

A filter with K rows and L columns requires $K \times L$ multiplications. Each multiplication requires the corresponding input neuron and filter weight. Fig. 6.12 shows all $K \times L$ multiplications occurring on the same lane, hence, each lane contains 4 neurons and 4 weights. However, they may also be distributed to multiple lanes. For example, a single multiplication can be performed in each lane. In which case each lane will have a single neuron and weight. Following the the layout in Fig. 6.12, the 4 multiplications in each lane proceed sequentially (using the techniques described in Section 6.3.1). All 4 lanes can perform this computation in parallel. After multiplication, the results all reside within the same lane. Then, the sum can be generated using addition. For typical NNs, the sum is read out from each row, and external circuitry performs the non-linear transformation. However, for binary NNs (BNNs) [76], a simple comparison operation (Section 6.3.1) can perform a logical threshold operation [301], producing the single bit output. In the case where neurons and weights for a single filter location are placed in separate rows, the results of each multiplication cannot be directly summed (as they reside in different rows). Read and write operations become inevitable after the multiplication to move results to a single row.

In summary, regardless of the specific data layout, convolution in a PIM array consists entirely of multiplications, additions, intermediate reads and writes, and (potentially) a non-linear operation. For our benchmark, we perform two-dimensional convolution with a 4×3 filter on a set of 16×16 neurons with 8-bit precision, using a comparison as the non-linear operation. Three multiplications are performed sequentially and the products are added into a partial sum within each lane. Then the partial sums from 4 lanes are moved to a single lane to compute the final sum and output.



(a) StxSt (b) RaxSt (c) BsxSt (d) StxRa (e) RaxRa (f) BsxRa (g) StxBs (h) RaxBs (i) BsxBs

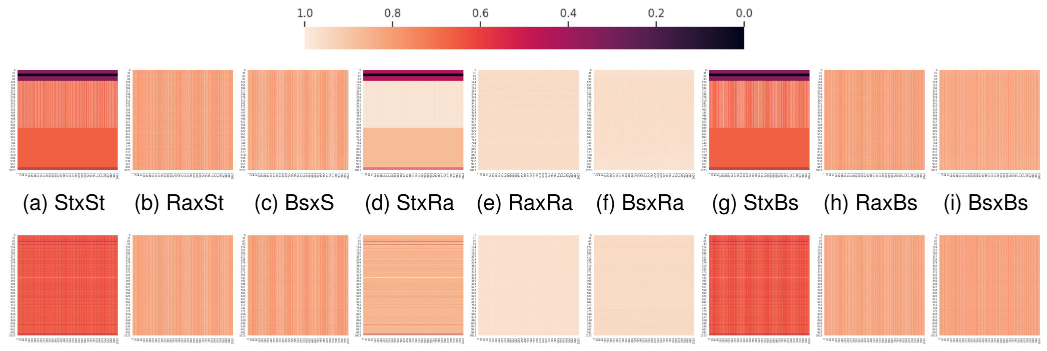
(j) StxSt+Hw (k) RaxSt+Hw (l) BsxSt+Hw (m) StxRa+Hw (n) RaxRa+Hw (o) BsxRa+Hw (p) StxBs+Hw (q) RaxBs+Hw (r) BsxBs+Hw

Figure 6.13: Normalized multiplication write distribution heatmaps (1 indicates maximum utilization) with re-compilation every 100 iterations. Each map captures a different row strategy$\times$column strategy

6.7.2 Parameters

We choose a PIM array size of 1024×1024, which is a typical subarray size used for NVM [90], large enough to perform non-trivial computations, yet small enough to maintain electrical properties to feasibly enable PIM [400]. We evaluate a column-parallel architecture as a more realistic hardware implementation, requiring few modifications to existing NVM designs [210]. We also account for the overhead for pre-setting the output memory cell of logic operations, as is required by CRAM architectures [301, 399, 62, 65, 298].

Due to temporally fine-grained hardware based re-mapping, each iteration of a benchmark can have a different write distribution. Hence, it is necessary to fully simulate a large number of iterations. We simulate each benchmark 100,000 times to obtain an estimate of the overall write



(j) StxSt+Hw (k) RaxSt+Hw (l) BsxSt+Hw (m) StxRa+Hw (n) RaxRa+Hw (o) BsxRa+Hw (p) StxBs+Hw (q) RaxBs+Hw (r) BsxBs+Hw

Figure 6.14: Normalized convolution write distribution heatmaps (1 indicates maximum utilization) with re-compilation every 100 iterations. Each map captures a different row strategy×column strategy.

distribution over time. We find write distributions for all combinations of load balancing strategies. Specifically, we experiment with two strategies in software, *random shuffling* of addresses and *byte-shifting* of addresses, respectively, which we refer to as *Ra* and *Bs*. We also include a *static* strategy, *St*, which excludes any re-mapping. Each of these strategies can be used for either rows or columns, giving rise to a total of 9 different load balancing configurations (3 row strategies $\times$ 3 column strategies). *Hardware re-mapping*, *Hw*, on the other hand, as detailed in Section 6.5, is applied only within the lane (within columns and across rows) and can be turned on or off. Hence, there is a total of 18 load balancing configurations per benchmark overall.

Software re-mapping can be invoked every time the program is recompiled. Recompiling does not come for free, hence cannot be performed very frequently. However, more frequent re-mapping (hence recompilation) is more effective at balancing load. Accordingly we sweep the re-mapping frequency (i.e., every 10, 100, 1000, and 10000 iterations of the application) to characterize this trade-off space. Hardware re-mapping, on the other hand, does not incur any recompilation overhead. In this case we experiment with the most extreme case of re-mapping on every gate that uses all lanes. For all types of re-mapping, we assume oracular operation, as our focus is finding the theoretical limit for the benefits of re-mapping[5]. Otherwise latency and energy efficiency overheads are architecture specific.

We use write distributions to estimate the lifetime of the PIM array by finding when the first memory cell fails. We consider this as the failure of the entire array, because at this point the array can produce incorrect results. Additionally, the analysis in Section 6.6 shows that even a few failed cells can significantly disrupt operation. The lifetime of the array hence corresponds to:

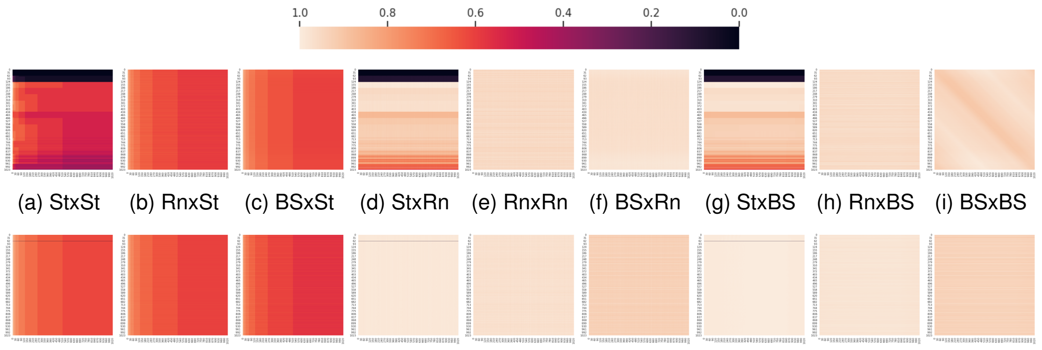
$$Lifetime = \frac{Cell\ Endurance}{max(WriteCount)} \times Application\ Latency \tag{6.4}$$

We assume the same endurance for each cell, which makes our analysis more pessimistic as the actual endurance is more likely to vary across cells (our approach can be thought of as using the average endurance for the expected lifetime). Specifically, we base our analysis on MTJs (ReRAM has a much worse endurance as detailed in Section 6.2) and assume an endurance of 10^{12} writes [251, 330]. Application latency is the time it takes to complete each benchmark. We compute

[5]As we are going to show in Section 7.5, even in their idealized form with no overhead, these techniques cannot be of much help due to fundamental physical limitations.

latency by summing the time it takes to do all operations (read, write, and logic). We assume 3ns per operation [309, 298].

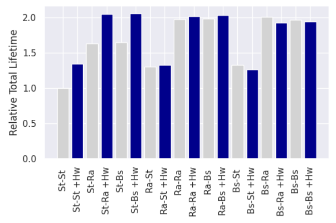
We assume that the PIM array performs each benchmark repeatedly, i.e., as soon as it computes the final results a new set of inputs is loaded and the process repeats. This is indicative of typical operation, as PIM arrays (whether used in embedded applications or high performance servers) serve as accelerators for computational kernels. For example, an embedded device which performs machine learning will likely only offload dot-products (used for matrix-vector multiplication) or convolution operations to the PIM array. Hence, the PIM array is likely to see many repetitions of the same computation on different data.



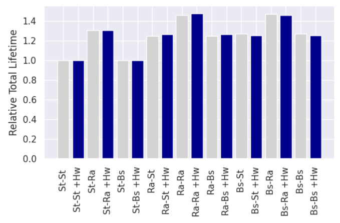
(a) StxSt (b) RnxSt (c) BSxSt (d) StxRn (e) RnxRn (f) BSxRn (g) StxBS (h) RnxBS (i) BSxBS

(j) StxSt+H (k) RnxSt+H (l) BSxSt+H (m) StxRn+H (n) RnxRn+H (o) BSxRn+H (p) StxBS+H (q) RnxBS+H (r) BSxBS+H

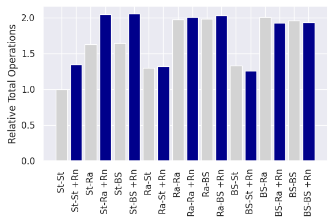
Figure 6.15: Dot-product write heatmaps (1 indicates maximum utilization) with re-compilation every 100 iterations. Each map captures a different row strategy×column strategy.



(a) Multiplication



(b) Convolution



(c) Dot-Product

Figure 6.16: Lifetime improvement with each strategy combination, in terms of operations possible before failure. Strategies include Static (St), Randomized (Ra), and Byte Shift (Bs) in software, along with remapping in hardware (Hw).

6.8 Evaluation

We start by inspecting the write distributions within the PIM array. The more uniform the write distribution, the better. Even distributions make better use of all cells, increasing the expected time to failure. We use heatmaps to visualize write density as shown in Fig. 6.13 for embarrassingly-parallel multiplication; Fig. 6.14, for convolution; and Fig. 6.15, for dot-product. Results are labeled by *row mapping strategy*×*column mapping strategy*, along with a *+Hw* if hardware re-mapping applies. As detailed in Section 6.7, the three options for *row mapping strategy* and *column mapping strategy* are: Static *St*, Randomized *Ra*, and ByteShift *Bs*.

Table 6.3: Lifetime in days of a 1024×1024 PIM array performing each benchmark continuously. Load balancing significantly improves lifetime, but not to practical levels.

Benchmark	Lifetime (Days)	Load Balanced Lifetime (Days)
Multiplication	31.57	39.32
Convolution	31.76	47.93
Dot-Product	30.43	62.63

For multiplication (Fig. 6.13), the inputs are only written once, where workspace cells are used many times. Hence, there is a large imbalance across rows when the row mapping is static. On the other hand, as this benchmark uses all columns for computation, there is no imbalance between columns. Row mapping strategies of randomization (Ra) and byte shift (Bs) are sufficient to significantly balance the writes even over rows. Hardware re-mapping (Hw) on top of this produces a nearly even write distribution.

The convolution benchmark performs 3×4 convolution, with each neuron-filter product mapped onto four columns. One of the four columns is used for the final sum. Hence, convolution (Fig. 6.14) over uses one-fourth of the columns, hence under-utilizes three-fourths of the columns. Convolution also uses an initial parallel multiplication, and hence also shows an imbalance across rows. Row re-mapping strategies are generally effective at balancing out the row usage. For columns, byte-shift (Bs) proves ineffective as highly used columns still overlap when shifted an integer number of bytes.

Dot-product (Fig. 6.15) heavily uses columns at low addresses, as partial sums are repeatedly moved to lower addresses to perform the reduction sum. Hence, there is a significant imbalance across columns. Both randomization (Ra) and byte shifting (Bs) manage to overcome this.

Considering the write distributions, we compute the lifetime of the PIM array with Equation 6.4. Assuming 3ns per operation [309, 301], and and endurance of 10^{12} writes [251, 330], the lifetime in days for a PIM array performing each benchmark non-stop is reported in Table 6.3. Also reported is the maximum lifetime achieved with load balancing strategies. It should be noted that imbalance impacts lifetime in multiple ways. For example, the large imbalance in dot-product causes some cells to fail sooner, which reduces lifetime. However, the imbalance also means that many columns are inactive for a large percentage of the time (dot-product exploits less parallelism then embarrassingly parallel multiplication, and therefore has fewer writes per unit time). If the premature failures resulting from imbalance can be removed, dot-product is less demanding on the PIM array, and hence will last longer.

Lifetime for PIM arrays for all load-balancing strategies relative to no re-mapping (i.e., St$\times$St) is shown in Fig. 6.16a for multiplication; Fig. 6.16b, for convolution; and Fig. 6.16c, for dot-product. Notably, dot-product, which has the most imbalance, benefits the most from load balancing. Multiplication with the least imbalance benefits the least. As multiplication only has imbalance in the lanes, only strategies which perform in-lane load balancing provide benefit. Specifically, Sta$\times$Ra and Sta$\times$Bs do not provide any benefit. For convolution, strategies which only re-map rows do not significantly improve lifetime. This is because the lifetime is limited by the imbalance across columns.

For software strategies, we found that the frequency of re-compiling does not need to be high. Over 100,000 total iterations of the benchmarks, we tested re-mapping every 10000, 1000, 500, 100, 50, and 10 iterations. We found that the expected lifetime saturates at approximately every 50 iterations. Over all the benchmarks and configurations that improved from 50 to 10 iterations, the improvement was on average only 0.93%. Hence, re-compiling every 50 iterations over 100,000 iterations (after every 0.05% of the total iterations) provides nearly optimal write distributions. As the PIM array is expected to perform many more than 100,000 iterations, the frequency of re-compilation can be significantly reduced. For benchmarks that were re-compiled every 0.05% of the iterations, the expected lifetime was on average 2.59×10^{11} iterations. Hence, re-compilation on average would only need to be performed approximately every 129,000,000 iterations. This infrequent re-compilation would incur relatively low overhead.

Unfortunately, despite significant improvements in lifetime due to re-mapping strategies shown in Fig. 6.16, the limitations imposed by endurance remain, as highlighted in Table 6.3. Even with aggressive load balancing strategies and optimistic endurance for modern devices, non-volatile PIM arrays can only be expected to function for approximately 1-2 months. This emphasizes the need for higher endurance in non-volatile memory devices and motivates device-level research.

6.9 Conclusion

In this work we showed that PIM operation requires many more writes to memory cells than standard memory operation. Considering realistic endurance of current non-volatile memory technologies, non-volatile PIM architectures would not function long with typical usage. Load balancing is the main method by which the lifetime of PIM architectures could be extended. We considered oracular versions of PIM specific load balancing techniques and found that, while effective, they are not sufficient to overcome the strict endurance limitations. This highlights the need for device level research to further increase endurance in order to make non-volatile PIM a reasonable solution.

Chapter 7

A Scalable Cross-Bar Architecture

7.1 Introduction

Due to the increasing performance of CPUs relative to memory over the past decades, the performance of modern architectures is limited by data movement. This is referred to as the *memory wall*. This means that, independent of the computing power available, an application's speed will be mostly determined by the amount of data it consumes and the bandwidth at which data be transferred between the CPU and the memory. This is a particularly unfortunate reality, as modern applications are using ever more data. Scientific applications and machine learning are notorious for their large memory budgets. Hence, overcoming the memory wall is one of the most critical challenges facing computing today. The response to the memory wall thus far has mostly come in the form of diversifying and specializing hardware. Much work has optimized GPUs, FPGAs, and ASICs for optimal data usage in order maximize performance.

Processing-in-Memory (PIM) architectures are most extreme response to the memory wall, and circumvent the bottleneck by performing the computation within the memory itself. This completely removes the need for expensive data transfers between memory and the centralized processing unit. Numerous PIM architectures exist which can perform basic logic either within the rows or within the columns of the memory arrays [210, 62]. Most significantly, PIM architectures which use non-volatile resistive memory elements, such as RRAM [163] or magnetic tunnel junctions (MTJs) [309], provide high density and energy efficiency. Extremely large numbers of logical operations (thousands to millions) can be performed in parallel inside the memory arrays, without any data transfer to compute units.

Unfortunately, and perhaps surprisingly, data movement still remains a significant limitation for these architectures. While data does not need to be transferred to a central processing unit, it does need to be transferred within and between the memory arrays. A PIM architecture which is capable of column (row) logic operations can use logic operations to move data between different rows (columns), but not between columns (rows). This necessitates intermediate read and write operations to move data, which introduce complexity and additional energy overhead. This puts a significant limitation on the scalability of PIM architectures.

Digital[1] cross-bar architectures [348, 32, 356] circumvent this limitation by enabling both column

[1] Digital PIM architectures perform Boolean logic within the array. This is in contrast to analog architectures, which perform

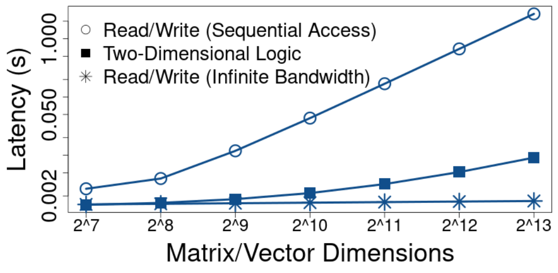


Figure 7.1: Latency of in-memory matrix-vector multiplication with different data transfer capabilities. Communication becomes the limiting factor for performance.

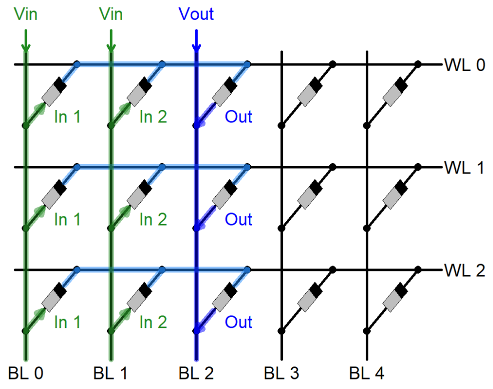


Figure 7.2: A two-input logic gate performed in an idealized cross-bar by applying voltage on the bitlines.

and row logic operations. This 2D compute capability allows data to be moved arbitrarily within the array via logic operations, remaining inside the array at all times. Basic logic gates can be performed by apply voltages along bitlines or wordlines. An example two-input logic operation in the rows from voltage applied on the bitlines is shown in Figure 7.2. This reduces communication overhead by eliminating the need for sense amplifiers and external circuitry to transfer data between rows or columns. Additionally, such functionality can be extended to enable logic operations between neighboring cross-bar arrays. This allows inter-array data transfer directly via logic operations, rather over an interconnection network. The logic operations can operate in a highly parallel fashion and there is less contention for shared hardware resources, resulting in high data transfer rates. Figure 7.1 shows how this enables scalable PIM implementations of matrix-vector multiplication, the cornerstone of machine learning inference and many other scientific applications.

However, a critical limitation of cross-bar architectures is the existence of sneak paths within the array [206]. The rows and columns of cross-bars are electrically connected together over the resistive memory devices in the array. Standard cross-bars have no additional hardware dedicated to isolating rows or columns from each other. The large number of connections results in many, unintended (sneak) paths for current. This inability to completely isolate the memory cells allows unwanted current to travel between different rows and columns during operations, which adds noise and consumes additional energy. If the noise is large enough, it can lead to incorrectness and make the device unusable. As a result, digital cross-bars have been limited to small dimensions and the

weighted-sum operations. The data movement advantage discussed here does not apply to analog cross-bars.

use of RRAM devices, which are resilient to noise due to their large resistance [348]. Still, such architectures have only limited practical use. RRAM devices have low endurance (10^{10} cycles [163]) and slow operations [222]. Being restricted to small cross-bars can make application mapping more challenging [32] and adds complexity to the peripheral circuitry.

Many strategies have been proposed to mitigate sneak paths [206], such as including a single transistor [217], diode [404], or selector [364] within each memory cell. However, none of these strategies can eliminate the sneak path issue for both dimensions of computation. In order to completely eliminate the sneak path issue, two transistors must be added to each cell. Unfortunately, this greatly increases the area overhead. For this reason, cross-bar architectures with more than one transistor in each cell have not been considered. However, we argue that a two transistor cell design, due to its ability to eliminate noise and energy from sneak paths, is the optimal architecture for the following:

1. Faster and more energy efficient memory devices (which are more susceptible to noise) can be used.
2. Arrays can be made larger, which enables lower complexity peripheral circuitry and application mapping.
3. Increasing energy efficiency is more important than decreasing area (PIM architectures already provide sufficient density).

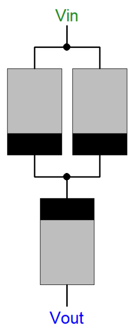
The price paid in area overhead is more than compensated for by boosts in performance, energy efficiency, and correctness. By eliminating sneak paths, RRAM can be replaced with Magnetic Tunnel Junctions (MTJs), which have much higher endurance, and faster, more energy efficient operations [309]. Cross-bars of MTJs can be built up to standard memory array sizes, decreasing complexity of the peripheral circuitry. The extra transistors consume additional energy, but less than what would be incurred by sneak paths.

7.2 Background

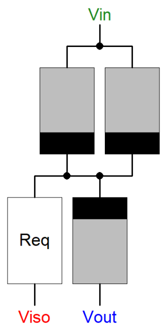
Many resistive memory technologies and PIM architectures have been developed in recent years, promising high performance, energy efficiency, and density. In this paper, we focus on *true* in-memory computing architectures, where computation occurs entirely inside the memory. In these architectures, voltage signals are applied to wordlines and bitlines and logic occurs via a thresholding mechanism inside the memory elements themselves [348, 62]. This is in contrast to digital PIM architectures which require sense amplifiers to perform logic.

7.2.1 Magnetic Tunnel Junctions

Memristors are non-volatile circuit elements which have a variable electrical resistance. The state of a memristor determines the level of resistance. Many memristor technologies exist, including RRAM, phase-change (PC) RAM, and magnetic tunnel junctions (MTJ). In this paper, we consider MTJs due to their superior endurance, performance, and efficiency [309]. A disadvantage of MTJs is a low ratio of resistance between their two states (high resistance and low resistance). This



(a) Ideal circuit for logic.



(b) With V_{iso} sneakpaths.

Figure 7.3: The presence of sneak paths allows V_{iso} to disrupt logic circuits within cross-bar arrays. R_{eq} is the resistance of all sneak paths to V_{iso}, and is determined by the resistances of the MTJs and the size of the cross-bar.

makes them more susceptible to voltage fluctuations, and are particularly challenging to use within cross-bars.

MTJs consist of two magnetic layers, the fixed layer and free layer. The polarity of the fixed (free) layer can (not) change. When the layers are aligned (anti-aligned), the MTJ is in the low-resistance parallel state with resistance R_P (high-resistance anti-parallel state with resistance R_{AP}). Passing a sufficient amount of current, I_{switch}, from the fixed (free) layer to the free (fixed) will set the MTJ to the parallel (anti-parallel) state.

7.2.2 Logic with MTJs

Logic can be performed with MTJs via *thresholding*. Using the circuit shown in Figure 7.3a, input MTJs are connected in parallel, which are then in series with an output MTJ. The output MTJ is preset to a known value. Voltages V_{in} and V_{out} are applied to drive current through the MTJs. The output MTJ will either change or not depending on the value of the input MTJs. This thresholding enables efficient logic within memory arrays, and is the key mechanism used in digital PIM architectures [348, 222, 62, 400].

The specific logic operation to be implemented determines the preset value and the magnitude and direction of the applied voltage. For example, the universal NAND gate requires the output to be preset to 0. Voltage is applied so that the output will switch to 1 if either of the inputs is 0. The appropriate level of voltage can be solved for, given the constraints imposed by the circuit. In this perfectly isolated circuit there are two contraints. First, the output MTJ should switch if at least one of the input MTJs is 0 (in the P state).[2]

$$\frac{V_{out} - V_{in}}{(R_{AP} \parallel R_P) + R_P} \geq I_{switch} \tag{7.1}$$

Second, the output MTJ should not switch if both input MTJs are 1 (in the AP state).

$$\frac{V_{out} - V_{in}}{(R_{AP} \parallel R_{AP}) + R_P} < I_{switch} \tag{7.2}$$

These two conditions provide a range voltages for $V_{in} - V_{out}$ for which the circuit will correctly

[2] $\parallel$ denotes “in parallel with”. $R_1 \parallel R_2 = 1/(1/R_1 + 1/R_2)$

perform a NAND gate. However, this range will change in the presence of sneak paths in the cross-bar. Note that V_{in} is negative because electrons flow in the opposite of the direction of the current.

7.3 Cross-Bar Constraints and Sneak Paths

Prior work has considered RRAM in cross-bars [348] and MTJs in one-dimensional PIM architectures [222, 62]. We now analyze the limitations of implementing a cross-bar with MTJs, which have a low ratio between their high and low resistance states - making them more susceptible to voltage fluctuations and sneak paths. There are two major challenges to consider, 1) enabling logic within a subset of the rows or columns while leaving other rows or columns unperturbed, and 2) preventing logic operations in multiple rows or columns from interfering with each other. An architecture will have to reliably perform both in order to be viable.

7.3.1 Performing Logic in a Subset of the Rows or Columns

A standard cross-bar does not contain transistors (or any selective device). Hence, a voltage applied on wordlines (bitlines) to drive a logic operation will cause the same operation to occur in all columns (rows). While performing computation in all rows is useful for some computations, many algorithms mapped onto cross-bars call for computation only within a subset of the rows [32]. Otherwise, data may be unintionally overwritten and corrupted. A method to enable computation on only a subset of the rows is to apply an *isolation voltage*, V_{iso} to the corresponding bitlines (wordlines) [348]. This strategy works, but it perturbs the desired logic operation, significantly reducing the voltage margins for correctness.

For concreteness, we show an explicit example of a row-wise NAND gate on a 3×5 cross-bar ($N = 3$ and $M = 5$). However, the analysis here extends to larger arrays, different gates, and column-wise operations as well. Figure 7.4 shows the implementation of a NAND gate in Row 0, with inputs in columns 0 and 1 and the output in column 2. Wordline 0 connects the inputs to the output. V_{in} (applied to bitline 0 and bitline 1) and V_{out} (applied to bitline 2) supply the voltage differential to drive the NAND gate. V_{iso} is applied to all wordlines (except wordline 0) to prevent the operation from being performed in any of the other rows. This leaves the data in all other rows intact.

V_{iso} has two potential impacts on correctess. First, through sneak paths in the cross-bar, it affects the voltage between the input and output MTJs. Second, it drives a voltage differential across MTJs in the other rows, risking an unwanted *write* operation. We must ensure that neither condition occurs, while still guaranteeing proper operation of the logic operations.

Sneak Paths

For ease of analysis, assume initially that all MTJs have the same resistance R, regardless of state. First consider the sneak paths from all rows which reach wordline 0 through bitline 3. The source of V_{iso} is connected to bitline 3 through $N-1$ parallel MTJs. Bitline 3 is then connected to wordline 0 through a single MTJ. Hence, the resistance between the source of V_{iso} on all wordlines (except

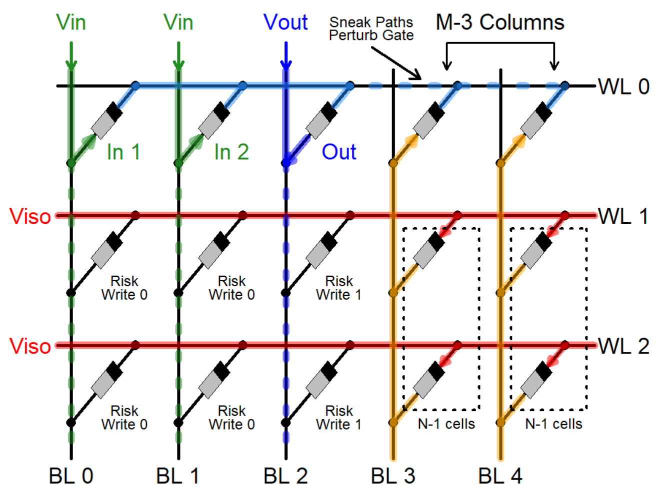


Figure 7.4: V_{iso} prevents logic on rows, but perturbs the gate operation via sneak paths and risks writing MTJs on other rows. Larger cross-bars have more sneak paths and, hence, lower voltage margins.

for wordline 0) through bitline 3 is

$$\frac{R}{N-1} + R \tag{7.3}$$

As there are $M-3$ bitlines for sneak paths (all except for bitlines 0, 1, and 2), there are $M-3$ such resistances in parallel between V_{iso} and wordline 0. Hence the total resistance between the source of V_{iso} and wordline 0 is

$$\frac{\frac{R}{N-1} + R}{M-3} = R_{eq} \tag{7.4}$$

Hence, the original threshold circuit in Figure 7.3a, becomes the circuit in Figure 7.3b, where V_{iso} is connected to the MTJs in the logic gate via R_{eq}. With large cross-bars R_{eq} is considerably smaller than R_P, and most of the current will travel through R_{eq}. In practice, the MTJs constituting R_{eq} will be in unknown states, but to guarantee correctness the worst case should be assumed. In this case $R = R_P$ is the worst case as $R_P < R_{AP}$, and a lower resistance will allow for a larger sneak current.

In order to maintain the conditions required for NAND in Equations 7.1 and 7.2, $|V_{in} - V_{out}|$ must be increased to compensate for the leakage through R_{eq}. The voltage on the node connecting the inputs to the outputs V_c can be solved for using standard nodal analysis.

$$\frac{\frac{V_{in}}{R_{IN}} + \frac{V_{iso}}{R_{eq}} + \frac{V_{out}}{R_P}}{\frac{1}{R_{IN}} + \frac{1}{R_{eq}} + \frac{1}{R_P}} = V_c \tag{7.5}$$

where R_{IN} is the parallel resistance of the two input MTJs. This provides a new set of constraints in place of Equations 7.1 and 7.2.

$$R_{IN} = R_{AP} \parallel R_P \longrightarrow \frac{-V_c}{R_P} \geq I_{switch} \tag{7.6}$$

$$R_{IN} = R_{AP} \parallel R_{AP} \longrightarrow \frac{-V_c}{R_P} < I_{switch} \tag{7.7}$$

V_c should have a large enough (negative) magnitude to induce switching on the output MTJ when at least one of the input MTJs is in the R_P state ($R_{IN} = R_{AP} \parallel R_P$), and it should be too small to induce switching if both input MTJs are in the R_{AP} state ($R_{IN} = R_{AP} \parallel R_{AP}$). The output MTJ is guaranteed to be in the R_P state.

Unintentional Writes

The second impact of V_{iso} is a potential *write* operation to MTJs not involved in the logic gate. MTJs in the same columns as the inputs are exposed to a voltage differential of $V_{in} - V_{iso}$. This can cause cause a write 0 (due to current direction). This provides an additional condition

$$\frac{V_{iso} - V_{in}}{R_{AP}} < I_{switch} \tag{7.8}$$

Note, again, that V_{in} is negative. MTJs in the same column as the output MTJ are exposed to a differential of $V_{iso} - V_{out}$. If V_{iso} is more negative than V_{out} this can induce a write 1 (due to current direction). This provides the additional condition:

$$\frac{V_{out} - V_{iso}}{R_P} < I_{switch} \tag{7.9}$$

V_{iso} can also, potentially, be more positive than V_{out}, in which case it can induce a write 0. This provides a condition:

$$\frac{V_{iso} - V_{in}}{R_{AP}} < I_{switch} \tag{7.10}$$

Values of V_{in}, V_{out}, and V_{iso} which satisfy the conditions in Equations 7.6, 7.7, 7.8, 7.9 and 7.10 will properly drive NAND gates in the cross-bar array. An example for a 4×4 cross-bar is shown in Figure 7.5, where V_{in},V_{iso} combinations within the polygon represent solutions. However, the margins for these values will be small for large arrays. Assuming V_{out} is held at a constant $0\,\mathrm{V}$, the maximum allowable fluctuation of voltage on V_{in} and V_{iso} is shown in Figure 7.6. Even at modest array sizes, the margin becomes too small to operate reliably.

7.3.2 Multiple Logic Operations

The second major consideration is preventing different, independent logic gates from interfering with each other. Even without the application of V_{iso}, sneak paths exist which can cause incorrectness. When logic is performed in rows (columns), bitlines (wordlines) which are not connected to inputs or the output carry current between wordlines (bitlines) of different logic operations. Due the large number of nodes in the circuit, an analytical description is untenable and we use HSPICE to solve for the voltage margins. The margin of V_{in} when performing NAND gates in all rows is shown in Figure 7.7.

7.3.3 Satisfying Both Constraints

Section 7.3.1 covered the challenges involved in performing computation in some rows (or columns) while leaving others unperturbed and Section 7.3.2 covered the challenges in performing computation in multiple rows (or columns). Generally, both challenges need to be addressed simultaneously.

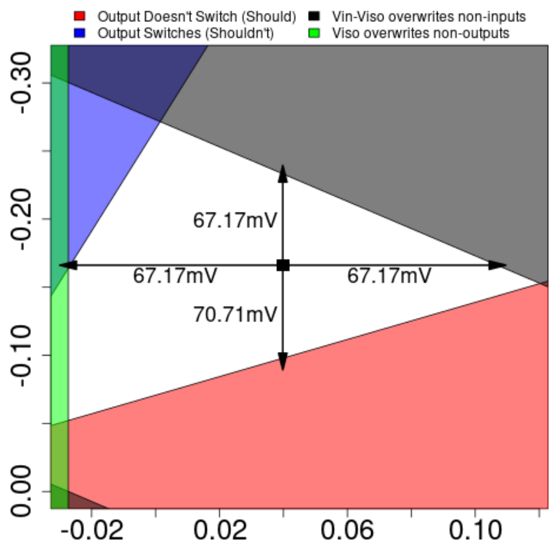


Figure 7.5: Only certain combinations of V_{in} and V_{iso} will work. The colored regions represent different constraints being broken for an 4×4 cross-bar. Assuming R_P = 7 kΩ and R_{AP} = 70 kΩ. V_{in} is negative as electrons must flow from the inputs to the output.

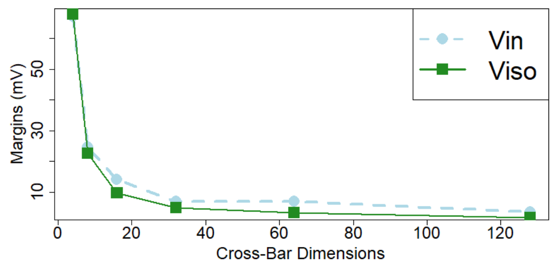


Figure 7.6: Voltage margin of V_{in} and V_{iso} when performing a NAND in one row for cross-bars of different sizes. Assuming R_P = 7 kΩ and R_{AP} = 70 kΩ.

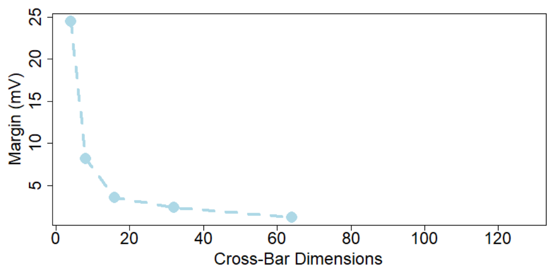
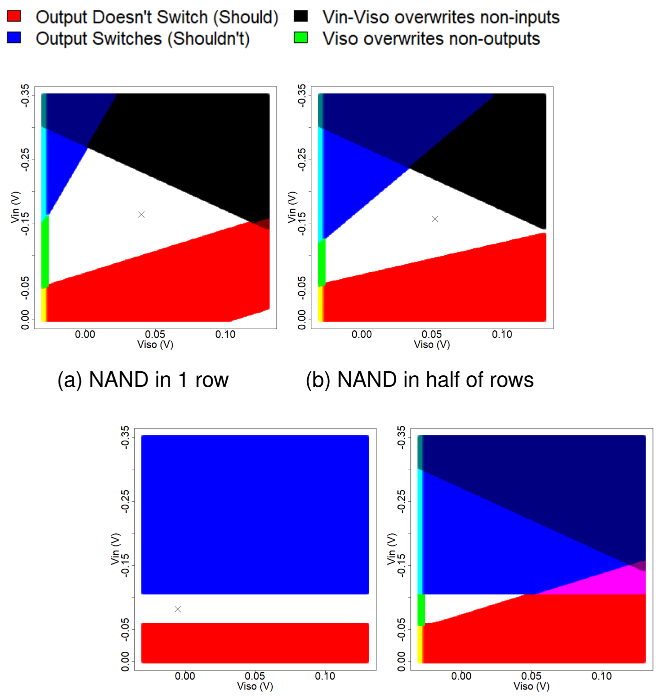


Figure 7.7: Voltage margin of V_{in} when performing a NAND in all rows for cross-bars of different sizes. When all rows participate, V_{iso} is not required. Assuming R_P = 7 kΩ and R_{AP} = 70 kΩ.

The optimal values of V_{in} and V_{iso} change depending on the number of rows (or columns) that are performing logic. figures// 7.8a-7.8c show how the constraints change as the number of rows performing computation increases. Figure 7.8d shows all constraints simultaneously. Either, constant values of V_{in} and V_{iso} must be chosen within the white polygon in Figure 7.8d, or the values of V_{in} and V_{iso} must be changed depending on the number of active rows.



(a) NAND in 1 row

(b) NAND in half of rows

(c) NAND in all rows

(d) NAND in any number of rows

Figure 7.8: The required voltage of V_{in} and V_{iso} change depending on the number of logic operations being performed. The white polygon represents acceptable solutions in each case. Even for an 4×4 cross-bar shown here, the margins are very small. The optimal operation point, which maximizes the margins, is shown with an x.

7.4 2T1-R Architecture and Advantages

Given the limitations of cross-bars discussed in Section 7.3, we argue that a two-transistor one-resistor (2T-1R) cell is an optimal cross-bar design. Two transistors are required to remove all sneak paths, while simultaneously enabling two-dimensional digital computation. Cross-bars with only one-transistor or selector per cell can only remove sneak paths for one dimension of computation.

PimCity is shown in Figure 7.9. The architecture is identical to a standard cross-bar, except a transistor can isolate each MTJ from the BL and WL. One of the transistors in each cell is activated row-wise, via the Row Activation (RA) control line. The other transistor is activated column-wise, with the Column Activation (CA) control line. When *both* transistors are enabled in a cell, current is allowed to pass through the MTJ. Due to being able to specify both row and column activation, the cells which have both transistors activated can be specified exactly, without any half-selects or sneak paths.

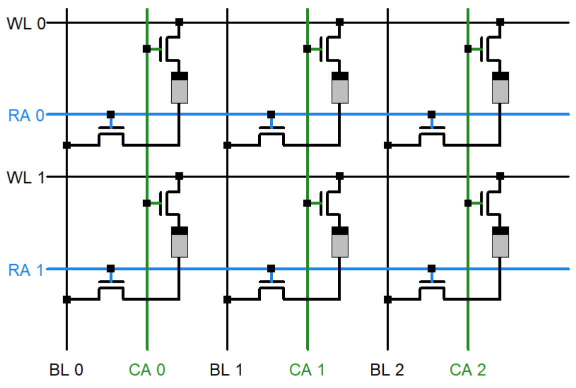
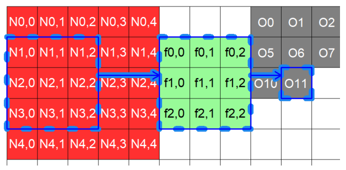
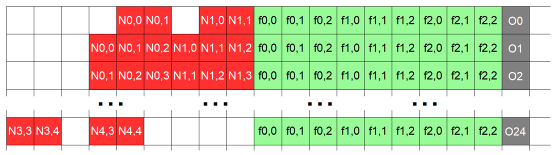


Figure 7.9: A 2T-1R crossbar enables both row-wise and column-wise selection of cells.



(a) 2D logic without duplication


N0,0 N0,1 N1,0 N1,1
N0,0 N0,1 N0,2 N1,0 N1,1 N1,2
N0,1 N0,2 N0,3 N1,1 N1,2 N1,3
N3,3 N3,4 N4,3 N4,4
f0,0 f0,1 f0,2 f1,0 f1,1 f1,2 f2,0 f2,1 f2,2
O0
O1
O2
O24

(b) Duplication for 1D row logic [301]

Figure 7.10: Two-dimensional computing reduces data duplication requirements for application mapping. Computation requires temporary values and multiple steps (not shown).

Given that scalable two-dimensional computation comes at a large area cost, it should also provide large benefits. Two-dimensional architectures provide two major benefits. The first is superior intra-array communication, which simplifies data transfer and reduces data duplication requirements. The second is superior inter-array communication, which enables scalable architectures.

Many 1D PIM architectures exist [301, 210, 62] which perform computation in either the rows or the columns of memory arrays. Architectures which perform computation in the columns (rows) cannot transfer data between different columns (rows) with logic operations. They require intermediate read and write operations to move data within the array, which has high latency, energy, and complexity. Data must frequently be permuted [301], requiring hardware external to the array to perform this processing.

Overhead for this data movement can be mitigated with data duplication techniques [301]. For example, binary neural network convolution requires the element-wise dot product of a *filter* with a collection of *neurons*. The filter must be slid over the neurons, with the output computed at every location. A one-dimensional PIM architecture can make this operation more efficient by duplicating neurons and filters, placing the required bits into each row as shown in Figure 7.10b [301]. However, this comes at a large data duplication cost. In 2D convolution, if the block of input neurons is $N \times M$, and the filter is $f \times g$, $2 \times N \times M \times f \times g$ cells (plus work space) are required.

Table 7.1: MTJ parameters

Parameter	Value
R_P	$7\,\text{k}\Omega$ [158]
R_{AP}	$70\,\text{k}\Omega$ [158]
I_{switch}	$3.9\,\mu\text{A}$ [399]

Two-dimensional architectures can perform logic in both rows and columns, where the locations of the inputs and output are arbitrary. This enables them to perform all data transfers directly, without read and write operations. Effectively, data movement and logic operations are indistinguishable and use the exact same mechanisms, enabling efficiency. Additionally, data duplication is not fundamentally required to achieve performance as the data can be operated on in-place. Figure 7.10a shows a data layout for two-dimensional computation. Using a combination of row and column logic, neurons and filters can be used without duplication, requiring $N \times M + f \times g$ cells (plus work space).

The second major advantage is the possibility of transferring data between neighboring cross-bar arrays. If the wordlines and bitlines of neighboring arrays can be conditionally connected (at the cost of 1 additional transistor per row and column), logic operations can transfer bits between them. These data transfers can be highly parallel, and obviate the need for an interconnection network. Such transfers can be highly parallel and scalable, such as indicated in Figure 7.1.

7.5 Evaluation

In this section we compare the area, latency, and energy overhead of PimCity (2T-1R cross-bar) with the standard cross-bar architecture. We use HSPICE simulation using 7-nm FinFETs from the ASAP PDK [66]. We model MTJs as resistors, using the parameters listed in Table 7.1. The advantage of PimCity over a standard cross-bar will decrease with more ideal MTJ performance. To be overly generous to the standard cross-bar, we assume MTJ parameters which will be possible within the next few years. Larger R_{AP}/R_P ratios will be possible, and I_{switch} will reduce due to a decrease of the damping constant of ferromagnetic materials [96] and using a dual-reference layer structure [88]. As PimCity is logically equivalent to a cross-bar architecture, application mapping remains identical. Automated compilers, such as Contra [32], can be used.

7.5.1 Area

The most significant limitation of a 2T-1R cell design is the area it consumes. Transistors are much larger than MTJs, hence their size determines the cell area [253]. To model the access transistors, we use FinFETs from the ASAP 7-nm Predictive PDK [66]. The size of the transistors depends on the number of fins required, which is determined by the amount of drive current required to write MTJ devices. Hence, the transistors need to be large enough to supply a few microamps of current. We estimate the area of PimCity based on previous cell designs for MTJ based PIM architectures [400], which use two transistors per cell. A suitable cell of PimCity will have dimensions of $109\,\text{nm}\times 350\,\text{nm}$ [400]. Standard cross-bars, without transistors, have a cell area of approximately $35\,\text{nm}\times 35\,\text{nm}$ [254].

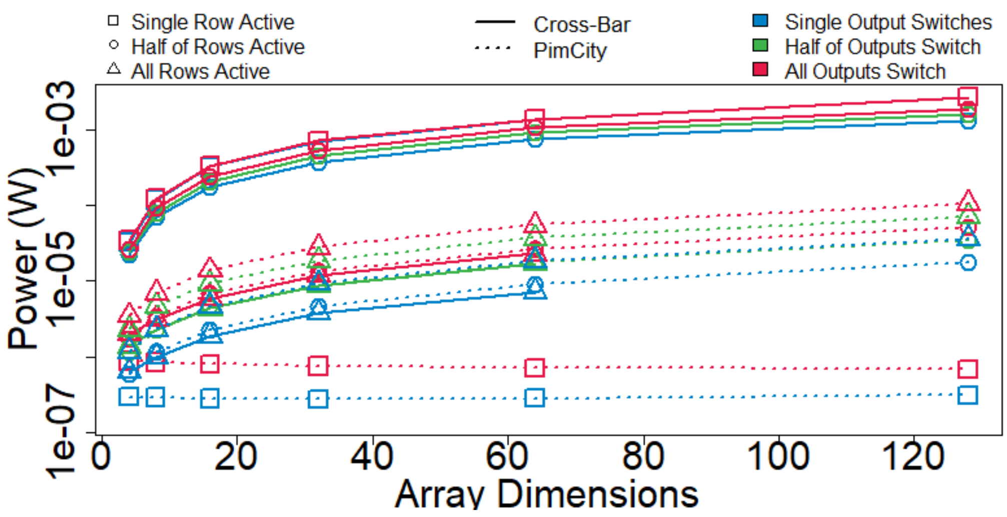


Figure 7.11: Power consumption of Cross-Bar and PimCity architectures for increasing array sizes. Power consumption varies by number of active rows and the number active rows which occur in switching of output MTJs. The cross-bar cannot successfully perform Based on HSPICE simulation using 7-nm FinFET devices from ASAP Predictive PDK [66].

The increased area, while significantly larger than standard cross-bars, remains well below conventional alternatives. For IoT applications, PIM implementations of neural network inference require less than 35MB of memory [298]. 35MB of PimCity, accounting for a 10% overhead for peripheral circuitry, would consume $11.7\,\mathrm{mm}^2$. In contrast, a conventional hardware solution for the same applications with the widely used MSP430FR5994 microcontroller consumes over $100\,\mathrm{mm}^2$ [120]. Hence, PimCity still offers a large density improvement over alternative architectures. Additionally standard cross-bars are restricted to small array sizes (due to voltage margin limitations) which will require a higher relative area overhead for peripheral circuitry.

7.5.2 Power Consumption

Using transistors to isolate memory cells also has energy efficiency benefits over standard cross-bar architectures. While the transistors themselves consume energy, they eliminate energy costly sneak paths. The power consumed by a standard cross-bar and 2T-1R PimCity when performing NAND gates is shown in Figure 7.11. There are three variables which determine the energy consumption, the size of the array, the number of active rows (which perform logic), and the number of logic operations which result in switching of the output. Power consumption for PimCity only increases with increasing numbers of active rows or switching outputs. Power consumption of the standard cross-bar increases with array size, regardless of the number of active rows. The presence of sneak paths and the application of V_{iso} to non-active rows results in large amounts of wasted energy.

7.5.3 Voltage Margins

Sneak paths cause voltage margins of cross-bars to quickly decay with increasing array size, as shown previously in Figures 7.6 and 7.7. As PimCity uses transistors to completely isolate the transistors, voltage margins are not affected by sneak paths. The voltage margin is determined entirely by equations 7.1 and 7.2, with the addition of transistor resistances in series with each MTJ. For the given parameters, the voltage margin for V_{in} is a constant $107\,\mathrm{mV}$. However, it is possible to lower V_{in} (reducing the margin) to increase energy efficiency. For the power consumption reported in Figure 7.11, we maintained a minimum margin of $30\,\mathrm{mV}$. While PimCity is immune to sneak paths, larger arrays will still have have larger bitline and wordline resistances, limiting array size to 1024×1024 [400].

7.6 Related Work

Analog cross-bar architectures [261, 351] can provide high density and highly energy efficient computation. Such architectures do not require any transistors within the cell, maximizing density. However, all such analog cross-bars require expensive analog to digital converters at the array periphery to perform computation. A number of cross-bar architectures perform digital computation via thresholding within the array itself [32, 356]. Such architectures require 0-1 transistors per cell, enabling high density, but suffer from significant noise due to sneak paths.

7.7 Conclusion

Cross-bars provide high density and two-dimensional logic, which can provide efficient application mapping and scalable architectures. Such designs are promising architectures to help over come the memory wall and further increase performance and energy efficiency. However, they suffer from sneak paths which make them infeasible at large sizes. We claim that a 2T-1R design is in fact necessary to solve all sneak path limitations and enable scaling to larger array sizes. We showed that such an architecture consumes reasonable area and provides more energy efficient in-memory computation.

Chapter 8

Quantum Benchmarking and Impact of Quantum Noise

8.1 Introduction

There are many ways to measure the performance of a computer[1]. Common ways have been measuring *operations per second* (OPS) or *floating-point operations per second* (FLOPS). These are intuitive and easy to understand, however, they are generally poor metrics. The problem is that hardware could be designed to have a very high OPS/FLOPS but could perform poorly on real world applications, which do not consist of monolithic blocks of arithmetic operations. A way to improve upon this is to measure the progress of a program rather than the number of operations it performs. For example, the HINT benchmark measures quality improvements per second (QUIPS), which measures the numerical accuracy improvement of the output in a given time [141]. While this can be insightful, again the main concern is that this does not accurately represent the real-world programs that will be run on the hardware.

Generally, the best metric is the wall-time required to complete a program [212], if the program is representative of real-world applications. This concept has led to the creation of benchmarks which are samples of larger, industrially useful applications. SpecCPU [336] and Parsec [33] are popular suites in this vein. While this is a clear improvement, it is not without its issues. For one, the reduced size of the programs introduces estimation error on the performance. There are other, less obvious complications. For example, academic work commonly reports performance on Parsec for system evaluation. Now, the human made choices of which benchmarks to include in Parsec determine what the academic community considers to be important. This makes these choices critical, because if the selection is not representative these results can be misleading. Even further, having established benchmarks would allow for a hardware designer to “cheat” by making a system particularly good on only the specific applications.

The takeaway is that benchmarking is possible and useful, yet is tricky and can be misleading. It is difficult to create useful benchmarks, and it may be impossible to create universal ones. This same construct applies to quantum computing, except it is much more intricate. There are a number

[1]For clarification, we note benchmarks are operations that the system is asked to perform (programs) and metrics are measurable characteristics of the system when performing the benchmark.

of complicating factors:

1) Quantum hardware is more diverse than classical hardware;

2) Quantum hardware is less developed, most systems have only a few qubits and cannot perform useful applications;

3) Quantum algorithms are still being developed and it is unknown what applications will be the most useful;

4) Quantum noise is not well understood and difficult to simulate, making characterization particularly challenging.

The dissimilarity of quantum hardware makes it hard to compare them to each other. This is not as much of an issue for classical hardware. While there has been a trend towards more diversified and specialized hardware in recent years, such as application specific integrated circuits (ASIC), there is a general framework and almost all hardware is silicon CMOS based. This makes metrics, benchmarks, and general intuition portable across different devices. Currently, there are many different hardware approaches competing in quantum computing. Each is based on a different physical system with entirely different dynamics. For example, quantum computing can be performed in superconducting circuits, ions isolated in a vacuum, or in atoms embedded in silicon. These systems look very different from each other and each have unique advantages and deficiencies. Is it fair to compare them directly?

As quantum computing is early on in its development, there are only small-to-medium sized quantum computers in existence. Most systems are not capable of performing useful programs. This makes it difficult to create benchmarks for these systems that are representative of future real-world applications. Scaling to larger sizes is particularly difficult for quantum computers, hence benchmarks that can be run on these smaller systems are less likely to accurately represent the performance of scaled-up versions. This is where one would normally turn to simulation. Unfortunately, as the states in quantum computers are highly complex, they are not able to be efficiently simulated by classical computers. Hence, benchmarks must be tied to a physical experiment.

On a more fundamental level, it is even unsure what quantum applications will be useful. As the field of quantum computing is largely unexplored, and not well understood, it is believed that many of its advantages and potential are currently unknown. Exploration of quantum potential is not well captured by benchmarking [37].

Quantum computing faces many hardware challenges. Information is easily lost due to quantum noise, which causes decoherence of quantum states. The physical devices need near absolute isolation from the environment, making the systems large and difficult to scale. Due to this fragility, benchmarking begins much lower in the system stack. Benchmarks even for 1-bit operations have been developed [185, 82, 99]. Even at this level, performance has been difficult to quantify. Accurately modeling quantum noise and determining the robustness of quantum operations has become the subject of much research [376, 100, 185, 285]. Noise can affect quantum programs differently, depending on their length and structure. Hence, noise is a significant complicating factor.

Thus, quantum computing inherits all the benchmarking complexity of classical computing, but introduces many additional complications. This makes it quite unclear what the best way is to evaluate a quantum system. In fact, the authors of [37] argue that it is too early to develop a standard approach. They warn that quantum research is currently exploratory in nature and that benchmarks are inappropriate for this kind of work. In fact, it could even be detrimental due to the possibility of misguiding research efforts.

In this chapter we provide a quantitative comparison for different quantum strategies from a benchmarking perspective, in the presence of quantum noise, to pinpoint pitfalls and fallacies.

8.2 Quantum Primer

In this section we introduce background and context for quantum computing. Necessarily brief, this clearly cannot do it justice. Quantum mechanics is highly complex and defies intuition. To quote Richard Feynman, "If you think you understand quantum mechanics, you don't understand quantum mechanics." This seems even more applicable if one views quantum mechanics from the perspective of computer architecture [152]. But in an attempt to "understand quantum mechanics", we will attempt to cover key concepts. We recommend [135] as an introduction to quantum mechanics and [221] as an introduction to quantum computing.

Quantum mechanics describes the nature of the physical world. High temperatures and large sizes causes quantum mechanical effects to become less noticeable, and classical physics acts as a good approximation. But when one creates a system that is very small or very cold, only quantum mechanics can accurately describe the system and how it evolves in time. Under these conditions, states are noticeably quantized (take on discrete values), such as the discrete possible energy levels of electrons around the nucleus of atoms. We can assign logical values to these distinct states, which are then called *qudits*. Transitions between these states correspond to quantum logical operations. Qudits can have many possible values, for example there are many possible non-degenerate energy levels for an electron. However, it is often convenient to use only two of the possible states, such as the ground and first excited states, as these become analogous to classical bits and are less susceptible to noise [266]. These two-level qudits are called *qubits*. Qubits can be in both of their states simultaneously (superposition) and multiple qubits can have their states intertwined (entanglement). Hence, there is not only information in each qubit, but *between* each qubit. As a direct consequence, quantum states can store an amount of information that is exponential in the number of qubits. This enables extreme compute capabilities if one is able to create a complex quantum state and reliably transform it in a meaningful way. Unfortunately, this is a difficult task. Pure [2] quantum states are extremely fragile and need near perfect isolation from the environment to exist. At the same time, we need to be able to interact with the quantum state in order to transform it.

Large scale quantum computing, despite the fragility of quantum states, remains a possibility due to quantum error correction (QEC). By encoding quantum information for a single qubit using multiple qubits, the quantum state can be restored if only a subset of the qubits become corrupted. Encoded qubits are called *logical* qubits, which are composed of multiple *physical* qubits. QEC is a rich field [127], and there is much work devoted to studying how QEC works under different error models [143, 145, 45, 132, 238]. However, modern quantum computers do not yet have sufficient numbers of qubits or required qubit quality to practically implement QEC. Hence, modern quantum applications operate on physical qubits and try to make use of limited resources. Therefore, it is of

[2]Pure states are quantum states which can be completely specified by state vectors.

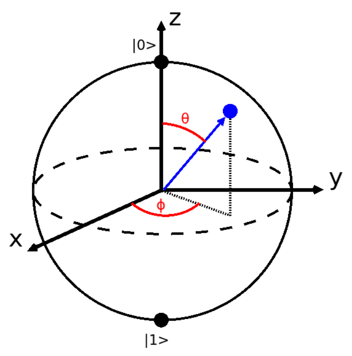


Figure 8.1: Bloch Sphere representation of a single qubit.

interest not only how quantum error affects QEC, but also how it affects algorithms running without QEC.

If a quantum state is completely isolated, it is called a *pure state*. Quantum pure states can be represented by *kets*, which are column vectors of complex numbers. The elements of the kets are called the *amplitudes*. For example, a single qubit can be represented by a ket of length 2

$$\alpha 0 + \beta 1 = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \tag{8.1}$$

where α is the amplitude associated with state 0 and β with state 1. The amplitudes determine probabilities when performing measurements, $|\alpha|^2$ is the probability of measuring this single qubit in the state 0 and $|\beta|^2$ is the probability of measuring the qubit in the 1 state. The probabilities must sum to 1 for pure states. A single qubit can be visualized as a vector from the origin to a point on the Bloch Sphere, shown in Figure 8.1, where a pair of angles, θ and ϕ, can be used to specify the state of the qubit, related to the amplitudes by the equations

$$\alpha = cos(\theta/2) \tag{8.2}$$

$$\beta = e^{i\phi} sin(\theta/2) \tag{8.3}$$

Operations on qubits are called *gates*, which are represented by matrices. Common gates are the Pauli **I** (identity), **X** (NOT), **Z** (phase-flip), and **Y** (NOT and phase flip) gates, shown in Equation 8.4.

$$\mathbf{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \mathbf{X} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad \mathbf{Y} = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \quad \mathbf{Z} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{8.4}$$

Single qubit gates represent rotations on the Bloch sphere. Performing a gate is logically equivalent to multiplying the ket column vector by the matrix of the gate. The **I** gate leaves the qubit unmodified. The **X** gate rotates the qubit by π around the X-axis, which flips the amplitudes for the 0 and 1 states. Similarly, the **Z** gate rotates the qubit around the Z-axis and the **Y** gate rotates the qubit around the Y-axis. Other common gates include the Hadamard (**H**) gate and phase gates **S** and **T**, shown in Equation 8.5.

$$\mathbf{H} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad \mathbf{S} = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \quad \mathbf{T} = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix} \tag{8.5}$$

The **S** gate is a rotation around the Z-axis by $\pi/2$ and the **T** gate is a rotation around the Z-axis by $\pi/4$. Performing two **T** gates back to back is equivalent to an **S** gate, and two **S** gates back to back is equivalent to a **Z** gate. The **H** gate is commonly used at the beginning of quantum algorithms to put qubits into a perfect superposition state.

Two-qubit gates commonly involve a control qubit and a target qubit. In this case, a gate is performed on the target qubit if the control qubit is in the 1 state. For example, the controlled-NOT, **CNOT** gate is a controlled-**X** gate. Two-qubit gates are used to create entanglement between the qubits.

Quantum gates must be *unitary*. This means they must be linear, reversible, and preserve the magnitude of the column vector. E.g., the **X** gate is its own inverse, and if two **X** gates are applied sequentially, the qubit returns to the original state. In other words, unitary operations coherently transform the quantum state. Conversely, measurements are non-unitary and irreversible. If a qubit in a superposition of 0 and 1 is measured and found to be 0, it is then entirely in the state 0. This is an incoherent process, as the quantum state has effectively been destroyed, containing only classical information. Whether operations are unitary or not is important not only for quantum gates and measurements, but also for the noise that affects the quantum state.

Quantum states that are not pure are called *mixed states*. These states are combinations of pure states, each with an associated classical probability. Mixed states occur as the result of imperfect isolation and manipulation of the quantum state, which applies to all physically realizable quantum states. *Density matrices* are the equivalent of kets for mixed states. Density matrices can represent all pure and mixed states. A ket representation can be converted to a density matrix by taking the outer product of the ket with its conjugate transpose (adjoint)

$$\begin{bmatrix} \alpha \\ \beta \end{bmatrix} \longrightarrow \begin{bmatrix} \alpha\alpha^* & \alpha\beta^* \\ \beta\alpha^* & \beta\beta^* \end{bmatrix} \tag{8.6}$$

where * denotes the complex-conjugate. A common metric to quantify the “quality” of a quantum state is the fidelity [105], which is defined as

$$Tr[\rho\sigma] \tag{8.7}$$

where the density matrix ρ represents the “correct” quantum state; σ, the actual quantum state; and Tr, the trace (diagonal sum) of the matrices multiplied. Simulating with density matrices allows one to keep track of classical probabilities and possible errors, in addition to the quantum transformations. This can be convenient in many cases [144], however the computational resources required increase significantly [28].

8.3 Quantum Noise

Noise is present in all computing systems. However, it is quite a force to be reckoned with for quantum systems. In fact, noise is so pervasive that it is impossible to have a meaningful discussion about practical quantum computing without an in-depth consideration of its effects. Clearly, no benchmarking approach can succeed without considering noise and the resulting impact on measured or simulated results. This has been unfortunate, as quantum noise is difficult to characterize and, in many cases, its effects are not well understood. Here, we provide a brief overview of quantum noise and the models used to represent it.

8.3.1 Physical Sources of Noise

Quantum noise can come from a variety of sources. There is never only a single source of noise in any given quantum system. Determining what the sources are and what their relative contributions are is a hard problem. Possible sources are unwanted interaction with the environment (both distinct events and inevitable decay of quantum states), unwanted interaction between qubits, or imperfect control operations. Each of these introduce error with significantly different characteristics, giving rise to different models. Here, we go over different common sources and discuss their physical impact. A summary of physical noise sources is provided in Table 8.1.

Interaction with the Environment

Qubits need to be perfectly isolated from the environment to maintain their state. If such a system could be constructed, there would be no quantum noise. But no real system can be perfect, hence there is inevitably some interaction. This can be seen as a “measurement” of the system [24], as information is leaving the quantum state. As measurements are non-unitary, this kind of noise is also non-unitary. The expected amount of time a system can remain unperturbed is called the coherence time. Commonly reported are the T_1 and T_2 times. T_1 measures the expected loss of energy from the system; if a qubit is put into an excited 1 state, T_1 is a measure of how long it takes to collapse to the 0 state. This is also called the qubit relaxation time [357]. T_2 measures the dephasing time; if a qubit is placed in the superposition state $0{+}1$, T_2 determines how long it takes to polarize either to 0 or 1 [266]. Risk of interaction with the environment is increased when performing operations on the qubits, as the driving force of the operation comes from an external input. This is an unfortunate situation as two critical requirements have conflicting needs. The quantum state needs near perfect isolation to remain intact, yet also must interact with control mechanisms in order to perform useful computation. This is referred to as the coherence-controllability trade-off [395].

Interaction with the environment can also produce unitary errors, such as global external fields which act on the qubits [132, 20, 192, 374, 373]. Such interactions can cause unitary rotations of the quantum state.

Interaction with Other Qubits

As previously mentioned, qubits can become entangled with each other. This means their states become correlated. While this is a frequently used tool in quantum computation, it needs to occur only when desired. If qubits interact accidentally, this can lead to a mixture of their quantum states or decoherence [46, 283, 313]. This is referred to as cross-talk. This type of error has been particular difficult to characterize.

If left in perfect isolation, cross-talk between qubits would lead to a unitary evolution of the state. Hence, all the information is still contained in the quantum state. However, the quantum state would be different than the one desired, which destroys the ability to manipulate it in a meaningful way. For example, we may wish to have two qubits that are far apart and highly entangled, in order to perform quantum teleportation [3]. However, if they interact with other, nearby qubits, this will disperse and decay the entanglement [283]. If not in perfect isolation, cross-talk can cause increased degradation of the quantum state. Say one is performing error correction, which involves interacting extra (ancilla) qubits with the qubits that store the data, and then measuring the ancilla to extract error information (syndromes). Cross-talk may cause the unintentional interaction of a data qubit with an ancilla qubit. Thereby some of the quantum information in the data qubit can get transferred into the ancilla. As a result, when the ancilla is measured, the computer would unknowingly extract information from the data qubit, corrupting its quantum state.

Imperfect Operations

Imperfect application of quantum gates can generate incorrect quantum states. Often, these are slight over- or under- rotations which are the result of imperfect calibration [45]. These kinds of errors do not directly destroy the quantum state but lead to coherent evolution of the quantum state into an undesired state [24]. This type of noise is predominantly seen in modern experiments [376] and its potentially catastrophic impact on the ability to perform error correction has been a concern in recent years [45].

Leakage

Many quantum systems that are used as qubits actually have more than two possible states. In this case, two of the possible states are selected to represent 0 and 1. In other words, the qubit is encoded in a subspace of a larger quantum system [388]. This subspace is called the *computational subspace*. It is assumed that the quantum systems remain in these two states (though other states may be used temporarily, such as in the implementation of two-qubit gates [49]). If a qubit unintentionally enters one of these other states, it is referred to as leakage, and it can be particularly detrimental [107]. The return of a qubit back into the compuational subspace is called *seepage* [388]. Leakage and seepage can be either unitary or non-unitary, and can be caused by imperfect control or unwanted interactions with the environment [388].

	Environment	Other Qubits
Unitary	External Fields Over/under Rotations from Imperfect Control [24]	Cross-talk [283, 46]
Non-Unitary	Unintentional Measurements [24]	

Table 8.1: Categorization of physical noise into its sources and whether it is unitary or not.

8.3.2 Noise Models

Working at higher levels of the system stack, it is more critical to know how quantum noise will affect quantum operations. In this section, we transfer focus from physical sources of noise to the

[3]Quantum teleportation is the transfer of quantum information between qubits by means of quantum entanglement in combination with a classical channel. It can be used to transfer information between non-adjacent qubits in a quantum computer or over long distances via a quantum network.

process of modeling them. The goal is to learn how noise disrupts the correctness of quantum algorithms during operation. As most research labs do not have their own physical quantum computer, and publicly available machines have low qubit counts, accurate and efficient noise models are greatly desired to facilitate simulation. Quantum noise is notoriously difficult to model accurately [376]. There are variations of quantum noise and it is possible that it may even be non-Markovian. Knowing what specific type of noise is present and how it affects a particular system is difficult to determine without extensive, physical experiments. However, there are a number of possible methods to estimate noise, with varying degrees of accuracy and computational efficiency. Realistic noise models are often intractable to simulate at scale, so simplifying assumptions are made to reduce complexity [237]. It is important to know when these assumptions are appropriate to make in order to produce realistic results. Here, we give a brief, high-level overview of different noise models and discuss their implications. A summary of noise models is provided in Table 8.2 and a summary of the physical noise processes they emulate is shown in Table 8.3.

Stochastic Pauli Noise

Stochastic Pauli noise is the simplest and most intuitive noise model. Additionally, according to the well-known Gottesman-Knill theorem [126, 6], it is easy to simulate using classical computers [45] and, at the same time, easy to correct using standard error correction procedures [376]. Hence, it has become popular [204, 7, 177]. It is most applicable for modeling unwanted interactions with the environment, which is effectively unintentional "measurements" of the quantum state [24]. It can be implemented by inserting an **X**, **Y**, or **Z** gate into a circuit at random with some specified probability. The effect on the overall fidelity can be estimated with Monte Carlo simulation [204]. Alternatively, representing the quantum state as a density matrix, ρ, the noise can be modeled as

$$N_i(\rho) = (1-\epsilon_i)\rho + \epsilon_i^x \mathbf{X}\rho\mathbf{X} + \epsilon_i^y \mathbf{Y}\rho\mathbf{Y} + \epsilon_i^z \mathbf{Z}\rho\mathbf{Z} \tag{8.8}$$

where ϵ_i is the total error rate on qubit i and ϵ_i^x, ϵ_i^y, and ϵ_i^z are the rates for each type of error, corresponding to the probabilities of inserting each gate [45]. This is also referred to as *depolarizing noise* [357]. If $\epsilon_i^x = \epsilon_i^y = \epsilon_i^z$, it is called *symmetric depolarizing noise*. **X**, **Y**, and **Z** are operators performing the respective gate on the qubit. While **X**, **Y**, and **Z** gates are unitary operations (causing a coherent transformation of the quantum state), inserting them in a probabilistic manner does not represent a coherent process. Additionally, the linear combination of unitary operations, as in Equation 8.8, can represent a non-unitary operation. Hence, stochastic Pauli noise is an incoherent source [45].

A common strategy is to inject error only after each gate. However, this is not realistic as qubits can acquire error even when remaining idle [372]. Therefore, Pauli noise should be injected in every cycle. Numerous studies have found that stochastic Pauli noise models often lead to inaccurate and overly optimistic results [262, 45, 132, 143, 145, 24], but that they still can provide reasonable approximations in certain conditions. These include errors at the logical level under QEC [45, 132, 143, 28].

There are some natural extensions to this model which can result in more accurate simulations. Significant improvements can be made, while remaining efficiently simulable, by augmenting Pauli gates with Clifford group operators (i.e., Hadamard (**H**), phase (**S**), and **CNOT** gates) and Pauli measurements [145]. This involves the same process of inserting gates at random, but using a larger gate set, which, in addition to the Pauli gates (**I**,**X**,**Y**,**Z**), has **H**, **S**, and **CNOT** gates [238].

A fundamental problem with stochastic Pauli noise is that it is "not quantum enough" [45]. While the inserted Pauli gates are quantum operations, the choice of whether to insert them is based on a classical probability. While a classical noise model is familiar and intuitive from a computer architecture perspective, it is not necessarily true depiction of the real errors occurring at the physical level.

Coherent Noise

Coherent noise models attempt to capture evolution of the quantum state that, while not destructive, is still undesired. One could see this as coherently performing a quantum program, just one that is different than intended. Physical coherent noise can be caused by imprecise classical control of the quantum operations [45], external fields, and cross-talk [132]. Modeling coherent noise can be difficult as some of the relevant sources of noise are not well understood. Hence, simplifying assumptions are made. While not exact, the goal is for the model to affect the quantum state similar to realistic sources. Examples include static Z-rotations [45], X-rotations in combination with Pauli-X errors [132], or rotations about a non-Pauli axis [145]. Coherent noise is not efficiently simulable, meaning the classical resources required to simulate grow exponentially with the system size. Hence, these simulations are limited to relatively small systems [208, 357, 24, 54].

Coherent noise is typically much more detrimental, with a much higher worst case error rate [376, 45, 24]. Additionally, many quantum algorithms consist of periodic circuits, where the same sequences of gates are repeated many times. Coherent noise is particularly harmful in this case, where its effects get amplified with each iteration [36, 35]. From this, it may appear that coherent noise is more important, and should be assumed unless known otherwise. However, this may not be true for all circumstances. The effects of coherent noise were analyzed on quantum error correction [132, 45]. It was found that the coherence of the error is reduced at the logical level, and is further decreased with a higher code distance. This means that it may be sufficient to assume stochastic Pauli noise at the logical level, even if the physical noise is coherent. However, it was noted that using a stochastic Pauli model for physical noise would significantly under estimate the error rate at the logical level.

Unfortunately, as modern quantum computers are not capable of QEC, they cannot make use of this resilience to coherent error. Randomized Compiling (RC) [376] is a novel approach which may help in this domain. The basic idea is to perform randomizing Pauli gates during the run of a quantum circuit, which are interleaved with the gates of the program. At each location that randomization is introduced, the previous randomization is undone to return the quantum state to the desired state. These randomizing operations disrupt the coherent noise and tailor it effectively into stochastic Pauli noise. Prior to execution, these additional randomizing gates can be fused with (compiled "into") the actual gates in the circuit. Depending on the gate set available in the system, RC can be performed with no overhead.

	Efficient	Not Efficient
Unitary	NA	Coherent Rotations [45]
Non-Unitary	Stochastic Pauli Pauli Twirling [115]Clifford Channels [143, 145]	Amplitude/Phase Damping[357]

Table 8.2: Categorization of noise models into whether they are unitary and whether they are efficiently simulable or not.

Physical Noise Source	Noise Models
Interaction with Environment	@c@Stochastic Pauli Noise Amplitude/Phase Damping Pauli Measurements
Imperfect Control	Coherent Over/Under Rotations

Table 8.3: Physical noise and corresponding noise models

Amplitude/Phase Damping

Amplitude Damping (AD) is a non-coherent error model which captures energy loss from the quantum state into the environment, such as spontaneous emission of a photon [357]. This noise model is relevant to any quantum system with multiple energy levels, where there is an excited state and ground state, with a tendency for the excited state to decay to the ground state, such as ion-trap quantum computers which use the excitation levels of electrons. Additionally, the loss of energy must be to some environment – i.e., in the previous example, if the energy loss is a spontaneous emission of a photon, there must be an environment for the photon to escape into. Hence, if the quantum system was *perfectly* isolated, AD would not occur. AD is a realistic noise model but is also not efficiently simulable. However, models have been designed to approximate the effect of AD, but which remain simulable [145], such as Pauli Twirling [333, 117, 314, 357]. Phase damping (PD), sometimes call pure dephasing, is equivalent to a phase flip channel [357]. This does not change the probability of a qubit being in either the 0 or 1 state, but it changes the phase between the two states. AD is closely related to the T_1 time and PD is closely related to the T_2 time, which are discussed in Section 8.3.1.

The nature of quantum noise greatly affects the quantum state, and by direct result, the performance of any potential quantum computer. Hence, even at higher levels in the system stack, one must give serious attention to the expected noise present in the system and be sure it is adequately accounted for. This impact of quantum noise is often overlooked or given secondary consideration. In many cases stochastic Pauli noise may be an overly simplified model. If so, it will produce incorrectly optimistic results, especially for modern systems.

8.4 Qubit Benchmarking

Before talking about benchmarking a quantum computer, we have to talk about benchmarking the qubits themselves, even in a single (or two) qubit system. If one is constructing a quantum computer, it is clearly of great interest how reliable the quantum operations (gates) are. The average gate fidelity coarsely predicts how many gates can be applied before the quantum state gets too corrupted. Additionally, quantum error correcting codes only work if the error is below a certain threshold [184]. On top of this, the overhead of the error correction strongly depends on the fidelity [272]. Quantum gates suffer from high error rates. Errors in classical switches could be less than 1 in 10^{15}, whereas effective quantum error rates are frequently above 1%, where the previously mentioned error sources and models apply and determining their impact is a complex task.

Unfortunately, experimentally determining quantum states, and the fidelity of quantum operations, is not straight forward. Measurements of a quantum system are destructive, so the state will need to be prepared or the operation performed for each measurement. In addition to this time overhead, errors in the initial state preparation and the measurements, (SPAM) errors, can obscure the error in the operation.

A brute force approach to learn a quantum state is quantum state tomography [371]. This enables the classical extraction of all quantum information, and hence answers any questions we may have about its structure. This requires measuring a complete set of observables (physical properties that can be measured) which determines the quantum state. For example, say there is a single qubit in the quantum state

$$\psi = \sqrt{0.8}0 + \sqrt{0.2}1 \tag{8.9}$$

For numerical clarity we use a pure state, however this process works for mixed states, as well. While this is a pure state, it can be described equivalently in density matrix form:

$$\rho = \begin{bmatrix} 0.8 & 0.4 \\ 0.4 & 0.2 \end{bmatrix} \tag{8.10}$$

This state is unknown to the outside world, but the state can reliably be prepared by a quantum operation. The goal then is to determine this density matrix only by performing repeated measurements on the prepared state ψ. All single qubit density matrices can be represented as a linear sum of the Pauli matrices [274]:

$$\rho = \frac{1}{2}\left(S_0\mathbf{I} + S_1\mathbf{X} + S_2\mathbf{Y} + S_3\mathbf{Z}\right) \tag{8.11}$$

Therefore, if we find the coefficients, we can reconstruct the density matrix. These coefficients can be found by determining measurement probabilities when measuring along the X, Y, and Z bases [274]

$$\begin{aligned} S_0 &= P_0 + P_1 && = 0.8 + 0.2 = 1.0 \\ S_1 &= P_{0+1} - P_{0-1} && = 0.9 - 0.1 = 0.8 \\ S_2 &= P_{0+i1} - P_{0-i1} && = 0.5 - 0.5 = 0.0 \\ S_3 &= P_0 - P_1 && = 0.8 - 0.2 = 0.6 \end{aligned} \tag{8.12}$$

where P_ϕ is the probability of measuring ϕ. Note that $S_0 = 1$ by construction, as 0 and 1 are the only two measurement outcome possibilities. S_3 only requires measurements along the standard Z-basis. To find S_1 and S_2, measurements need to be performed along the X and Y axes. This can be done with a basis change prior to measurement, which is done by applying the appropriate quantum gate just prior to measurement. For example, performing a Hadamard gate then a Z-basis measurement is effectively an X-basis measurement. Note that each measurement must be performed many times to achieve accurate probability estimates. The exact probabilities are shown in Equation 8.12. For the example we computed them directly with $Tr\left[\xi\rho\right]$, where ξ is the density matrix of the corresponding basis state. Once the coefficients are known the state can be described

$$\rho = \frac{1}{2}\left(1 * \begin{bmatrix}1 & 0\\0 & 1\end{bmatrix} + 0.8 * \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix} + 0 * \begin{bmatrix}0 & -i\\i & 0\end{bmatrix} + 0.6 * \begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}\right) = \begin{bmatrix}0.8 & 0.4\\0.4 & 0.2\end{bmatrix} \tag{8.13}$$

This process can be generalized to multi-qubit systems as well, we refer the reader to a thorough introduction in [274]. However, it does not scale well as it requires a number of measurements which is exponential in the system size. In addition to this, it is difficult to distinguish states with low probability from those with zero [185] and the computation to convert measured results into an estimate of the quantum state is intractable [77]. Hence, this approach is not feasible for large systems. There are notable variations of this process, such as Shadow Tomography [3]. While quantum state tomography has exponential cost because it tries to answer all possible questions, shadow tomography attempts to only learn certain features by learning from measurements [162]. The name comes from the idea that one is not trying to learn the full density matrix, only the "shadow" it casts on the chosen measurements [3].

Quantum process tomography (QPT) uses the same approach as quantum state tomography, except that the goal is to identify a quantum operation, rather than the quantum state. Known input quantum states are generated and then the operation is performed. Quantum state tomography is then applied to the output states allowing identification of the process [64, 282]. As it follows the same procedure as quantum state tomography, it also requires exponential resources in the number of qubits. The key assumption that QPT makes is that the input state is known, and therefore only the quantum operation needs to be found. This simplifies the problem, and finding the parameters of the operation is equivalent to maximum likelihood estimation of a convex objective function (which has a single, global minimum) [131]. Hence, standard convex optimization techniques can be used to solve for it [61, 42]. This simplification comes at a cost however. The input state isn't necessarily known, as there can be errors in the state preparation process, which can lead to inaccuracy [131]. Compressed quantum process tomography [327] uses compressed sensing, a known classical signal processing strategy, to reduce the number of measurements required. The unitary operations performed by quantum computers can typically be represented by nearly-sparse process matrices, and hence can be well approximated by sparse matrices. These sparse matrices can be found with exponentially fewer measurements [327].

Direct Fidelity Estimation [105] is another clever method used to extract meaningful information without exponential overhead. This procedure estimates the fidelity of an arbitrary quantum state relative to a pure state. Here, the pure state is an error free state which is the intended result of a quantum operation; and the arbitrary state, what is actually produced. Note that complexity can be reduced by not attempting to learn everything about the state, seeking only to get the estimate of the fidelity of the arbitrary state. The estimate is achieved by measuring a constant number of Pauli observables (applying standard Pauli operations and then measuring). This is possible by making educated guesses about which Pauli observables are likely to reveal errors.

Gate set tomography (GST) is an extension of QPT [131, 264, 246]. Like QPT, the resources it requires increase exponentially with the number of qubits [131]. Hence, it cannot target large quantum systems and is also used mainly for 1- and 2-qubit systems. However, a significant advantage it has over QPT is that it is calibration-free, i.e., it does not depend on accurate descriptions of the initial prepared states [264]. This is significant, as QPT can generate highly inaccurate results when the gates used to prepare the input states have systematic error [246]. Like QPT, the process of finding the parameters of the quantum operations (gate set) involve maximum likelihood estimation of an objective function based on measurement results. However, the inclusion of state preparation and measurement (SPAM) errors in the gate set produces a non-convex objective function [131]. Hence, standard convex optimization techniques do not work, and a combination of approximate and iterative methods must be used. Another consequence of including SPAM errors is that it is not possible to characterize a single gate at a time, as in QPT. Rather, there is a minimal set of gates which must be estimated simultaneously [131]. The mathematical background of GST is provided in [264] and the protocol is provided in [131].

A scalable approach which has been utilized frequently in recent years is *Randomized Benchmarking* [185, 82, 99]. Randomized Benchmarking attempts to go beyond tomography by determining error probability per gate in computational contexts. Like GST, it is calibration free [264]. A further strength of randomized benchmarking is that is insensitive to variations in error between the different types of gates used [151, 375, 285, 247].

There are variations on the specific implementation [39], including numerous extensions [237, 373, 182, 111, 11]. We provide a high level description of the general approach, but note that different formulations may vary on the specifics. First, a random sequence of operations (gates) is generated. Commonly, gates are chosen uniformly at random from the Clifford set, as these can be efficiently performed on a quantum processor [234] and also can be efficiently simulated on a classical computer [125]. The sequence is then appended with a final operation which undoes the action of the entire sequence. For example, if the sequence has a length of m and the operation (set of gates) applied at cycle i is denoted as c_i, the effect of the first $m-1$ operations is

$$C = \prod_{i=0}^{m-1} c_i \tag{8.14}$$

where $\prod$ denotes matrix multiplication. The final operation is then chosen to be the inverse, or adjoint, of C

$$c_m = C^\dagger \tag{8.15}$$

and hence the full sequence will become the Identity operation.

$$\prod_{i=0}^{m} c_i = c_m C = C^\dagger C = \mathbf{I} \tag{8.16}$$

This final corrective operation is easy to determine, due to the Clifford set being efficient to simulate classically [234]. Hence, a quantum state subjected to this sequence will be returned back to its initial state (in the case of no error). This is often chosen to be the $00...0$ state. In experiment, the fidelity can be found by finding the probability of measuring the initial state [286]. Many sequences of the same length should be generated, and the results averaged over all cases. This process is then repeated, using sequences of various lengths. The fidelity will be a function of the sequence length and will drop exponentially as the the length increases. The fidelity can then be fit to a model, where constants will absorb the state preparation and measurement errors [234]. This enables the extraction of the average error per gate. For example, in a single qubit case with independent gate errors, the average fidelity of the sequence can be modeled as [235]

$$\langle F \rangle = A(-2r+1)^m + B \tag{8.17}$$

where r is the average gate error and A and B are the fitting constants. This enables r to depend exclusively on the fidelity of the gate operations [151]. This process can be applied to many qubit quantum states with relative ease, as the resources required scale polynomially in the number of qubits [131, 235, 236]. As a result, it has become a standard approach to benchmarking quantum systems. The common practice of using the Clifford set, while convenient, is also a limitation, as by itself it is not universal [131]. However, strategies have been found to extend randomized benchmarking beyond the Clifford set [182, 53, 151]. Additionally, note that the scalability of randomized benchmarking is enabled by only seeking a subset of the quantum information, it does not provide the full tomographic information about the gates [246, 101].

Modern analog quantum computers have been able to simulate systems that are hard even for classical supercomputers [384, 334, 48, 178]. This raises the concern of validating their output, as they may be untrustworthy [149, 123] and it is challenging to classically certify them [384]. *Hamiltonian Learning* is a process by which the Hamiltonian of a system is estimated in order to validate that it is simulating the correct dynamics [128, 101, 384, 18, 383]. Additionally, reconstructing the Hamiltonian of a quantum system will provide detailed diagnostic information which can enable noise reduction in experiment [101]. The Hamiltonian of an n-qubit system, which has dimension $d = 2^n$ can be described by d^2 parameters, though most Hamiltonians of interest can be described by $m = O(poly(n))$ [101]. Utilizing known information about the system can significantly reduce the number of measurements required and make the process tractible [128]. *Compressed quantum Hamiltonian learning* [382] is a process in which the dynamics of subsystems of a large device are measured against the dynamics of a smaller system, enabling a model to be created for the larger system. Using a *trusted simulator* [384, 383], which has a firm known mathematical model, enables an absolute model to be generated for the larger system[382].

8.5 (Quantum) Computer Benchmarking

This kind of benchmarking is more similar to classical benchmarking. The idea is to determine the compute capability of a quantum system running a program. This is sometimes referred to as *holistic benchmarking* [264, 265]. Note that this is still considering near-term computers that do not use error correction. It shares a number of considerations with classical computing, such as latency and available parallelism. However, these metrics are not as informative for quantum computers. As to be expected, there are a number of additional considerations. When discussing benchmarking for full quantum computers, it is important to reconsider the role of quantum error and, if applicable, quantum error correction. As previously mentioned, quantum noise is not equivalent to classical noise, and these differences get more pronounced the larger the system is.

In classical computing it would be feasible to obtain an error rate per gate and stitch these error rates together to generate an error model for a larger circuit. The quantum equivalent would be finding a fidelity for each gate, and then assuming this error rate for each gate on each qubit in the system throughout the entire program. This is *not* accurate for quantum circuits as quantum noise is context dependent [372]. Even if good estimates of gate fidelity can be obtained, using this information to model larger systems with more qubits is not straightforward. A gate performed on one qubit may induce error in another, via quantum entanglement or physical proximity. Hence, qubits must be considered as a monolithic system and their error rates cannot be considered independently [100, 104]. This is problematic as it makes it difficult to understand how noise affects large quantum computers. The whole point of creating large quantum computers is to create states that cannot be efficiently classically simulated. Unfortunately, that also means the noise becomes impossible to simulate. Hence, accurately characterizing the noise, and its significant effects on the reliability and performance, is not straightforward.

This has a few key impacts on benchmarking quantum computers. One is that it intensifies the error that is introduced when using a reduced program size as a benchmark. The error rates per qubit or per operation may be higher on a larger system. Some experimental evidence has shown that this may be overly pessimistic [100], but increased system sizes will no doubt increase susceptibility to noise due to complexity [243]. Hence, the rate of success of a program on a small system may be significantly different from that of a larger system, and extrapolating results is not straightforward.

8.5.1 Program Benchmarks

An intuitive approach is establishing a set of programs and measuring the performance of a computing system performing each one. As previously mentioned, this is common practice for classical computers. Replicating this for quantum computing would be collecting a set of quantum programs which are representative of the algorithms we would like to run on them. Common examples may be Shor's [331] for prime factorization or Grover's [138] for unstructured search. There are intuitive advantages to this approach, particularly in making the system perform a "real world" task. IonQ, a quantum start up which has an 11-qubit ion-trap quantum computer, appears to favor this approach. They tested their computer on the Bernstein-Vazirani [30] and Hidden Shift [367, 304] algorithms, and their metric for performance was the likelihood of measuring the correct output [389]. They claim that these algorithms are representative benchmarks and the results proved their system was the best as of early 2019.

A challenge that this approach introduces is that someone must decide which programs are important. This introduces the issue of invested interests [212]. In the classical computing domain, a lot of money is on the line when benchmarking hardware [152]. This problem is exacerbated by the fact that different, competing quantum technologies are superior at different programs. For example, the program benchmark approach was used in [213], where a handful of small quantum circuits [4] were used to benchmark and compare the performance of an Ion-Trap quantum computer with a superconducting quantum computer. Amongst the chosen benchmarks were three-qubit circuits implementing the Toffoli and Margolis gates. It was found that the higher connectivity (ability of different qubits on the machine to interact) of the ion-trap computer allowed it to have a much higher relative success rate on the Toffoli circuit, which contained more two-qubit gates during the program. The success rate was more comparable on the Margolis circuit, which required fewer two-qubit gates. The authors note that how well the quantum architecture matches with the requirements of the algorithm is a major determinant of the performance [213]. So, which of the two

[4] In literature on quantum computing, circuit refers to a sequence of quantum gates (instructions). It is analogous to "program" in classical computing.

benchmarks are more insightful? This question highlights the difficulty of creating a representative set quantum program benchmarks. While quantum computing is more complicated, much can be learned from previously discovered pitfalls and fallacies of classical program benchmarking. Intuitively, the benchmarks will be set by the customer, rather than the manufacturer. To the extent possible, the benchmarks should reflect the applications that customers will be running on them. Another key component of accurate (and honest) benchmarking with a set of programs is transparency [152]. This means providing data on all the programs and not unfairly weighting some results over others. For example, a quantum benchmark suite could be created which contains many circuits which have few two-qubit interactions, but also a few prominent circuits which require many two-qubit interactions, such as the Quantum Fourier Transform (QFT) [331]. A quantum computer with low connectivity could be benchmarked on all circuits, with only the average performance being reported. Such analysis could falsely inflate perception of the performance.

While impressive, these modern computers are very small compared to the computers we hope to build in the coming years. Hence, these benchmarks are also very small compared to truly useful programs. While running smaller versions of real world applications introduces error, and is accepted in classical benchmarking, this is exacerbated for quantum computers. Entirely new issues may be introduced when scaling up and it is difficult to say whether measurements taken today are good indicators of future performance. For example, IonQs computer [389] has all 11 qubits fully-connected, meaning each qubit can directly interact with every other qubit. This configuration is possible at this scale, but may not be for a system with hundreds or thousands of qubits. Such a system will likely require multiple fully-connected groups of qubits and communication will need to be orchestrated between them [243]. This introduces additional complexity which is not found in these small scale benchmarks. This is analogous to the classical benchmarking of machine learning inference accelerators. An accelerator which performs well on MNIST [198] digit recognition will not necessarily perform well on 1000-class ImageNet [87] classification. While the problem and computation is similar, ImageNet requires much more data, and memory management becomes the bottleneck.

A more fundamental question is what quantum programs will be useful in the future. Famous algorithms such as Shor's [331], Grover's [138], and quantum chemistry [271, 245, 51] are obvious examples. However, for the most part, these algorithms will remain well out of reach for some time. Currently, classical-quantum hybrid algorithms [320, 381, 103, 278] are popular due to their ability to make use of the limited resources of modern quantum computers. It is important to remember that quantum algorithm design is still an emerging field, and what actually is the best use of quantum computers is still unknown. This presents a moving target, which suggests quantum research should not too heavily invest in any one direction [37].

8.5.2 Quantifying Capability

Considering the current limitations of quantum computers, it may be more insightful to focus on how much work a quantum computer is capable of, in contrast to performance results on specific algorithms. Because quantum technology is not mature, modern quantum computers are not yet capable of performing commercially useful algorithms. However, creating larger, more functional computers is of great interest, regardless of the applications they perform. This type of benchmarking is a shift away from traditional, classical benchmarking, and is an attempt to uniquely and objectively quantify the "capability" of modern quantum computers. The aim is to abstract out as much as possible, such as unique architecture characteristics and performance on specific algorithms, and create a simple metric which indicates the general computational power of the machine. While this benchmarking approach can be applied to all modern quantum computers, it is intrinsically tied to the concept of *quantum supremacy* - demonstrating that a quantum computer can perform work that is not possible with classical computers. Hence, this type of benchmarking is not just used as a comparison between different points in the design space of quantum computers, but is also the experimental assessment of the capabilities of quantum computers with respect to classical machines. Such benchmarks seek practical demonstrations of quantum computers solving well-defined problems, which are also intractable (in terms of time and hardware resources required) for classical computers. The best known classical algorithm/implementation should be used as a baseline for comparison in this case, while accounting for all sources of noise on the quantum side.

The importance of demonstrating quantum supremacy cannot be overstated. A central motivation for creating quantum computers is to solve problems which cannot be solved by other means [346], which will only be possible if quantum computers can achieve (and go beyond) quantum supremacy. It has been argued that this will be fundamentally impossible due to noise [174]. Hence, even if an experimental demonstration of quantum supremacy does not provide scientifically or commercially useful results, it is an invaluable proof of concept for future research efforts. The success of quantum computing research is contingent upon reaching this goal. The key question then is how to quantify capability of quantum machines accurately and how to be sure when they truly achieve quantum supremacy.

Cross-Entropy benchmarking [38] can be used to validate the output of a quantum computer and was created specifically as method to test for quantum supremacy. The previously described program benchmarks consider decision problems, where the measurement at the end provides an answer to a specific question. In significant contrast, Cross-Entropy benchmarking considers *sampling problems*. Here, the measurement at the end effectively allows for the sampling from a certain distribution. It is the ability of the quantum computer to create such a distribution which acts as the demonstration of quantum supremacy. Hence, it must be shown that the quantum computer produces the distribution with a sufficient fidelity and that a classical computer (using polynomial resources) cannot provide a sampling from that same distribution. Beyond demonstrating supremacy, validating sampling problems is important because they correspond to important quantum applications, such as quantum simulation.

To produce the output distribution, a quantum circuit is constructed by choosing quantum gates (from a universal set) at random. Using a random circuit allows for limited guarantees of computational hardness, as it does not have structure [17, 47, 38, 41]. Executing this circuit on a quantum computer will produce a quantum state in which some states are much more likely than others, yet is also widely distributed over the 2^n possible measurement outcomes, where n is the number of qubits. The distribution resembles a speckled intensity pattern produced by light interference in laser scatter [17]. For a large quantum computer, 2^n will be much greater than the number of samples (executions of the circuit) that can be taken. Hence, it is very unlikely that two different measurements will be same [38]. This makes it difficult to discern it from a uniform random number generator by simply looking at the samples. However, the output can be distinguished and validated if the circuit is also classically simulated [220, 281, 123, 361]. The classical simulation (which requires exponential resources on a classical computer) will provide the exact output quantum state, and hence also the exact probabilities of measuring each result (in the absence of noise). This allows the comparison of the output of the experimental quantum circuit with the ideal quantum circuit, and also with any classical algorithm that attempts to generate the same distribution. The information theory definition of entropy is [328]

$$H(X) = -\sum_{i=0}^{N-1} P(x_i) log P(x_i) \tag{8.18}$$

where X is a random variable which can take on N values and $P(x_i)$ is the probability of observing $X = x_i$. The output of the quantum circuit, and a classical algorithm attempting to emulate it, is a random variable which can take $2^n = N$ values. The cross-entropy between the distribution of the ideal quantum probabilities (p_U) and the polynomial classical algorithm probabilities (p_{pcl}) can be defined as [38]

$$H(p_{pcl}, p_U) = -\sum_{j=1}^{N} p_{pcl}(x_j|U)\ log\ p_U(x_j) \tag{8.19}$$

where $p_U(x_j)$ is the probability that the quantum circuit, U, will produce the output x_j and $p_{pcl}(x_j|U)$ is the probability that the classical algorithm emulating U will also produce x_j. The value of interest is the average quality of the classical algorithm, which can be found by averaging the cross-entropy over an ensemble of random quantum circuits, U:

$$E_U[H(p_{pcl}, p_U)] = E_U\left[\sum_{j=1}^{N} p_{pcl}(x_j|U) \frac{1}{log\ p_U(x_j)}\right] \tag{8.20}$$

It was argued in [38, 41, 47] that the polynomial classical algorithm cannot accurately reproduce the distribution.

When performing the circuit on a quantum computer, a single error (such as X or Z error) will cause the output to become nearly statistically uncorrelated with p_U. Hence, depolarizing noise will cause an increase in the entropy of the output of the quantum computer, and it will resemble that of a uniform distribution. This means that the quantum computer, which is subject to noise, will only barely be able to produce a superior cross-entropy to that of a uniform random number generator.

$$E_U[H(Uniform, p_U)] = log\ N + \gamma = H_0 \tag{8.21}$$

where $\gamma = 0.577$ is the Euler constant. On this concept the quantum supremacy test is based. Any classical or quantum algorithm, A, which produces bitstring x_j with probability $p_A(x_j|U)$, can be evaluated on how well it predicts the output of an ideal quantum random circuit, U, by comparing its cross-entropy relative to that of a uniform classical sampler. This metric is called the cross-entropy difference [38]:

$$\Delta H(p_A) = H_0 - H(p_A, p_U) = \sum_j \left(\frac{1}{N} - p_A(x_j|U)\right)\ log\ \frac{1}{p_U(x_j)} \tag{8.22}$$

If the algorithm A produces the distribution perfectly, with no errors, the cross-entropy difference will be 1. If the algorithm A produces an uncorrelated distribution, the cross-entropy difference will be 0 [17, 38]. Effectively, this is a measure of how consistent the outcomes are with the predicted probabilities [36]. A classical algorithm (using polynomial resources) should fail, and a quantum algorithm running on a sufficiently powerful quantum computer should succeed. The experimentally determined cross-entropy difference, denoted by α, can be found by taking m samples from the quantum computer, with each producing a bit string x^{exp}, and then using classical simulation to find the value $p_U(x^{exp})$. α is then estimated by [38]

$$\alpha = H_0 - \frac{1}{m}\sum_{j=1}^{m} log\frac{1}{p_U(x_j^{exp})} \tag{8.23}$$

Quantum supremacy is achieved if a quantum computer can produce a higher α than a classical computer can. Unfortunately, quantum supremacy also implies that the p_U values required to determine α, which are produced by classical simulation (using exponential resources), can no longer be computed by a classical computer in a reasonable amount of time. Hence, measuring α directly is not possible if the quantum computer truly has achieved quantum supremacy. However, it should be possible to extrapolate α, if it has been reliably found at sizes which are slightly below the limit of quantum supremacy (p_U can still be computed by sufficiently powerful classical supercomputers [38]). In the summer of 2019 Google used cross-entropy benchmarking [5] to quantify the capability of their 52-qubit quantum computer, Sycamore, where they claim to have demonstrated quantum supremacy [17]. It is also possible to use cross-entropy benchmarking to find the average fidelity of individual gates [17, 38], similar to randomized benchmarking.

Note that tests for quantum supremacy doesn't necessarily require a universal set of gates on the quantum computer. For example, in the case of Boson Sampling, recent work has demonstrated that these algorithms can be composed by notably more noise-tolerant (yet not necessarily universal) set of gates [363].

A similar and influential sampling-based benchmark is quantum volume from IBM [34, 79]. The process of determining quantum volume shares many of the steps with cross-entropy benchmarking. According to the authors [34], there are 4 factors which determine the capability of a quantum machine:

1. The number of physical qubits
2. The number of gates that can be applied before errors make the the output unreliable
3. The connectivity of the machine
4. The number of operations that can be run in parallel

It is assumed that quantum volume targets modern, noisy quantum computers. Hence, factor 2) is referring to running a quantum algorithm without error correction and it is assumed there is an upper limit on the number of gates possible. In the future, it will be of more interest to determine whether the error rate is low enough to enable efficient error correction, or how often error correction needs to be applied. Hence, quantum volume will need to be adapted or superseded in the future [79].

[5]Google used linear cross-entropy benchmarking, which differs from the standard cross-entropy benchmarking process explained here. However, the argument is similar for why linear cross-entropy benchmarking is easy for a quantum computer and hard for a classical computer.

Similar to cross-entropy benchmarking, quantum volume attempts to abstract out all considerations and generates a single number which quantifies the capability of a quantum computer. The idea is to measure something which can be improved by each of the 4 considerations, meaning systems that are superior in each consideration will generally achieve a higher quantum volume. The score is determined by the largest random quantum circuit a quantum computer is able to complete successfully. Whereas cross-entropy benchmarking uses the cross-entropy to quantify the validity of the output, quantum volume uses the heavy output generation problem [5]. Heavy output generation also requires classical exponential time to validate the output from the quantum computer, as it relies on the ability to classically simulate the quantum operations [5]. Again, this will not be possible once physical quantum computers reach the scale where their quantum states are not efficiently classically simulable. By simulating the quantum operation, the output measurement probabilities can be exactly determined. When ordering all possible outputs from most likely to least likely, all outputs which have a greater probability than the median are called *heavy outputs*. The quantum circuit is then repeatedly performed and the output measured. If the measurements produce *heavy outputs* more than 2/3 of the time, the quantum computer “passes”; otherwise it fails. If noise has completely destroyed the information in the quantum state, heavy outputs will be generated 50% of the time [79]. The largest quantum circuit that the computer can get a pass on corresponds to the quantum volume, V_Q [79]

$$log_2 V_Q = argmax_m \left(\; min(m, d(m)) \; \right) \tag{8.24}$$

where m is the number of qubits and $d(m)$ is the maximum achievable depth of an m-qubit circuit. Hence, quantum volume is the largest square ($m = d$) circuit that can be successfully run.

Quantum volume has some limitations. A number of which were addressed in [36], which proposes a framework called Volumetric Benchmarks. The idea is to make quantum volume more general, by allowing different shapes of circuits, kinds of circuits (random, periodic, subroutines of algorithms), and different criteria for success. As noted by the authors of [36], errors will affect different kinds of circuits differently, such as coherent errors getting magnified by periodic circuits but getting smeared out by random circuits. This provides evidence that universal benchmarks should be avoided. While quantum volume is a novel and useful concept, it uses exclusively random circuits. As we show in our simulations, random circuits have a similar affect to applying randomized compiling [376]. Another potential drawback is that quantum volume can produce potentially misleading results. For example, a Honeywell quantum computer achieves a high quantum volume score due to its extremely high fidelity operations [339]. However, given the relatively low qubit count, its dynamics can be simulated classically with ease [4].

Another significant approach is that of Cycle Benchmarking [100]. This is similar to the process of randomized benchmarking. The core idea is to break a quantum program into sets of operations on all the qubits (cycles) and then individually characterize the fidelity of each cycle. This allows one to quantify how well the computer can do specific operations. Additionally, arbitrary programs can be broken into a finite set of cycles [372]. This prevents the number of required characterizations from growing exponentially with the number of qubits. Using cycle benchmarking, the “benchmarks” would be individual cycles and the metric is the process fidelity.

8.6 Fault Tolerant (Quantum) Computer Benchmarking

Thus far we've considered qubit benchmarking and holistic computer benchmarking for near-term quantum machines. While there has been much effort to create practically useful quantum algorithms for these machines, as of yet their performance falls far short of their classical competitors. Hence, the main goal of these near-term demonstrations has been for research and to learn how to achieve scalability. Once computers are built that have sufficient qubit counts and sufficiently low error rates, full-scale quantum error correction (QEC) [131] will become possible. QEC can detect and correct errors during the run of a program, and hence enable computers to perform arbitrarily long quantum programs. This is referred to as fault-tolerant quantum computing [284]. Such computers will be capable of solving real-world problems with much greater speed than classical computers, performing well beyond the threshold for quantum supremacy. Naturally, such fault-tolerant computers will significantly change the landscape of benchmarking as it breaks many of the assumptions of previously described techniques. For example, the maximum circuit depth is effectively infinite, hence a metric such as quantum volume would be limited only by qubit count. More fundamentally, the quantum states of these machines would be much too large to simulate, and hence the classical validation of the heavy output generation problem would not be possible. A typical use case for large-scale fault-tolerant quantum computers will be running algorithms which will be impossible to run on a classical computer, but for which the output can efficiently be checked for correctness on one [221], such as Shor's algorithm [331].

However, no fault-tolerant computers have been built to date, and hence there is not much research dedicated to benchmarking them. Likely, quantum benchmarking will look much more similar to classical benchmarking, where the emphasis will transition to performance (latency to produce the final result) amongst quantum processors which have sufficiently many qubits to perform the algorithm of interest. Benchmarking may take a similar form as program benchmarks, as described in Section 8.5.1, except the chosen benchmarks would be real-world, full-size applications rather than sample toy cases. Additionally, unlike the modern computers performing the program benchmarks described in Section 8.5.1, a fault-tolerant computer should not fail due to noise. Hence, the probability of success will not be a representative metric.

The performance of fault-tolerant computers will have a strong dependence on the method QEC used as it has a high overhead. QEC utilizes many qubits [108], involves many, repeated quantum operations, and also requires significant classical hardware resources to manage its orchestration [248]. Both the qubit chip [95] and the supporting classical architecture [354] will need to be specifically designed to support QEC. For an introduction to quantum error correction, we refer the interested reader to [127].

8.7 Simulations

To illustrate the impact of the different types of quantum noise in different computational contexts, and the resulting difficulty of choosing representative benchmarks, we run representative key quantum algorithms at sizes that are experimentally feasible. We use a Quantum Adder [60], the Quantum Fourier Transform (QFT) [331], and the Quantum Approximation Optimization Algorithm (QAOA) [103]. In addition, we use an idle circuit (which has no computational gates) and a random circuit (composed of random X, Y, Z, H, and CNOT gates), for reference. Note that this is not the same randomized circuit as used in Quantum Volume [79], which generates a set of arbitrary random unitary operations, which need to be decomposed into a universal gate set. For our random circuit, we want to view the effects of performing our gate set in random fashion, without introducing the complexity of gate compilation (which is needed for our other algorithms). Unless otherwise stated, our metric is the process fidelity [105]. We note that other metrics may be equally suitable. However, we chose process fidelity as it is widely used in the quantum information science community [118] and it shines light on the complexities of benchmarking mentioned earlier.

The QFT and QAOA benchmarks contain gates which are precise rotations around the X- or Z-axis of the bloch sphere. $R_z(\theta)$ is a rotation around the Z-axis by angle θ and $R_x(\theta)$ is the equivalent for the X-axis. Currently available modern quantum computers, such as IBM's machines [7], can perform these operations directly. This can be powerful, as it enables quantum computation to occur relatively quickly, which also mitigates the effects of noise. As this gate set is likely to be used for some time into the future, we include it in our simulations of the QFT and QAOA. However, this gate set is not compatible with randomized compiling (RC) or quantum error correction. Hence, it is not scalable to the level of fault-tolerant quantum computers. For progressing into the fault-tolerant regime, the Clifford+T set is a good option. This gate set is universal and enables use of RC. Hence, we include Clifford+T versions of the QFT and QAOA as well. While we are using the Clifford+T gate set, we are not performing error correction in the simulation. Hence, these do not represent fault-tolerant computations. However, using this gate set can provide insight on how the impact of noise is affected by the chosen gate set. Each gate in Clifford+T can be implemented with the previously mentioned X- and Z- rotations – using the Clifford-T set just restricts the angles used. When performing these gates on physical qubits, as we do here, the operations are not changed in any fundamental way. While we do not show simulations of error correction, it should be noted that, in order to perform error correction, groups of physical qubits would form a single logical qubit, and the logical operations consist of many individual gates on the physical qubits [108].

The drawback of Clifford+T is that precise rotations must be broken into sequences of gates which can perform the operation approximately. This increases the length of the circuit. For single qubit gates, we use the gridsynth decomposition method from [323] which finds an approximation to Z rotations with Hadamard (H), T and S gates. This is sufficient, as any single qubit gate U can be implemented using the Euler angles [323]

$$U = R_z(\beta)\ R_x(\gamma)\ R_z(\delta) = R_z(\beta)\ H\ R_z(\gamma)\ H\ R_z(\delta) \tag{8.25}$$

for some β, γ, and δ. Hence, any quantum gate can be approximated by a sequence of approximate R_z gates and H gates. An additional complication exists in that controlled versions of these gates cannot be implemented directly [183]. For a general case, controlled 2-qubit versions of the gates can be implemented by including an additional ancilla (scratch) qubit and using an alternative sequence of gates [13]. However, both the QFT and QAOA can be implemented with just CNOT gates and single qubit R_z gates. Hence, controlled- H, S, and T gates are not required.

The Quantum Adder performs binary addition using a sequence of quantum full-adders on two input integers which are basis state encoded (1 qubit for each bit). If performing n-bit addition, the quantum adder requires $3n + 1$ qubits. We perform 2-bit addition (using 7 qubits). The adder only uses gates in the Clifford+T set and hence does not require gate decomposition. It does use the

Toffoli (doubly controlled X gate), which we implement with a sequence of CNOT, H, and T gates.

The QFT effectively performs the Discrete-time Fourier Transform (DFT) on the amplitudes of the input quantum state, and is the core of Shor's algorithm. The width and depth of the circuit are determined by the number of input qubits. We use 4 input qubits. The circuit contains Hadamard gates and controlled-Z rotations, which can be implemented with CNOT and R_z gates.

QAOA has gained a lot of attention recently as it is considered to be a strong candidate to demonstrate quantum supremacy. QAOA is a *variational* algorithm, where quantum computation is used in tandem with classical optimization [381]. It is commonly applied to the Max-Cut problem. The input to Max-Cut is a graph which represents a combinatorial optimization problem. Vertices are variables and the edges between vertices are constraints. The goal is to partition the vertices into two sets, maximizing the number of edges between the sets. The *maximum cut* is the number of edges between the sets in a best possible partition, which is the most constraints that can be satisfied simultaneously. Max-Cut is NP-Hard, however, it is *not* expected that quantum computers will be able to solve NP-Hard problems in polynomial time. QAOA only provides an approximate solution and currently has worse performance than classical approximate algorithms [122, 193]. QAOA consists of a sequence of stages. Each stage consists of controlled-Z rotations (determined by the input graph) followed by single qubit X rotations. The hyperparameter p determines the number of stages in the quantum circuit. For each stage, there is a parameter γ, which determines the angle of the controlled-Z rotation, and a parameter β, which determines the angle of the X rotations. The main challenge of QAOA is to determine an optimal set of parameters in order to produce good output [139]. When implemented on a real quantum computer, optimization algorithms such as SPSA [335] or gradient descent can be used to update the parameters [343]. This requires many optimization passes, where each pass requires many (thousands or millions of) samples of the quantum circuit output. The number of samples required heavily depends on the problem size and noise level in the system. Here, we consider only a pre-optimized circuit with $p = 1$. When $p = 1$, the optimal circuit is guaranteed to produce an output with an expectation value that is at least 0.6924 times the maximum cut [103, 137] on 3-regular graphs. Here, we use a 3-regular input graph with 8 vertices (represented by 8 qubits), which has a maximum cut of 12. This means QAOA should produce an expectation value of 8.3 in the case of no error. For this problem, we use expectation value of the output as the metric. As QAOA is designed to also work on a small scale, it can be used effectively with continuously parametrized rotation gates available on near term machines. We implement QAOA with these gates as well as with the Clifford+T set. A useful tutorial for QAOA is provided in [277].

For our simulations, we assume the quantum computer has an all-to-all connectivity and full parallelism. This means 2-qubit gates can be performed without any overhead for movement, and multiple single qubit gates can be performed simultaneously, given that they do not operate on the same qubits. Moving qubits with swap gates in order to compute on machines with limited connectivity is a well-studied problem [205, 407]. Our observations here will also apply, but there will be further complicating factors depending on the specific topology. For discussion on how noise and topology interact see [355].

We also view the impact of RC [376] in each of our simulations. RC was designed as a method to convert coherent noise effectively into stochastic Pauli noise. This is significant, as coherent noise can be much more destructive and is predominantly seen in physical experiments. Hence,

Benchmark	# Qubits	Logical Depth	Physical Depth	Gate Set	Metric
IDLE	4	2-70	2-70	I	Process Fidelity
RANDOM	4	2-70	2-70	I,X,Y,Z,H,CNOT	Process Fidelity
Adder	7	30	30	H, T, CNOT	Process Fidelity
QFT	4	10	10	H,R_z,CNOT	Process Fidelity
QFT (Clifford+T)	4	10	229	H, T, S, CNOT	Process Fidelity
QAOA	8	10	10	H, R_z ,R_x, CNOT	Expectation Value
QAOA (Clifford+T)	8	10	113	H, T, S, CNOT	Expectation Value

Table 8.4: Benchmarks used in simulations. # Qubits is the number of qubits required for each input size. Logical depth is the depth of the quantum circuit (number of sequential gates) required before compilation into Clifford+T set. Physical depth is the depth after compilation. The Adder and QFT have two different input sizes.

RC not only provides a significant noise mitigation technique, but also maintains the validity of previous theorems and results which have been generated by assuming stochastic Pauli noise. We emphasize, as noted by the original authors [376], that RC is not designed to have any impact on a Pauli noise model, hence we expect to see no improvement in the fidelity if a Pauli noise model is used. Additionally, for most of our simulations we are using process fidelity as the figure of merit, for which RC is not expected to add as much benefit. RC will provide greater impact if using a norm-based measure, such as the trace distance. In order to perform RC, we need to divide the gate set into "easy" and "hard" gates. Easy and hard can generally be thought of as the difficulty in implementing the gate, such as the expected error rate, but the essential requirement is that the physical noise on the easy gates be independent of which easy gate is performed. We follow [376] and set easy gates as the Pauli gates and the phase gate S; where the hard gates are H, T, and all 2-qubit gates. However, instead of controlled-Z, we use CNOT. Nearly all gates in the decomposed circuits are "hard" gates. Hence, it is necessary to interleave these gates with idle cycles in order to implement RC. From a high level, simulation perspective, this initially seems to imply that the cost of RC is a near doubling of the circuit length. However, as noted by the authors of [376], on real hardware (such as ion traps and superconductors) entangling operations, which are required for "hard" multiple qubit gates, must be inserted between local gates. These local gates, which are required even without RC, can be "compiled into" the randomized gates. Hence, in practice RC can be implemented with no additional circuit length overhead. Therefore, to make our simulations more representative of physical experiments, we insert the additional idle operations into our circuits whether RC is performed or not.

We use four representative noise models which fall into different categories discussed in Section 8.3. While no quantum system will have a single source of noise, we use them in isolation to demonstrate how the nature of the noise (along with the assumptions made in noise models) significantly impacts the results. The first is a standard Pauli noise, where the probability of X, Y, and Z errors are all equally likely. While the most commonly used noise model, it generally provides the worst (and overly optimistic) estimates or error correcting threshold error rates [143]. Pauli noise is a non-unitary and incoherent model. To model purely coherent and unitary noise, we follow the same approach as in [45], where we assume constant Z-rotations by angle θ for each qubit, $(e^{i\theta Z})^{\otimes n}$, for various values of θ. While this model makes some simplifying assumptions, it is representative. The third is a combination of Pauli and coherent noise, where we follow the model

Error Level	Pauli	Coherent	Pauli+Coherent	Amplitude Damping
0	0	0π	0 , 0π	0
1	0.01	$\pi/30$	0.01 , $\pi/30$	0.01
2	0.02	$\pi/15$	0.02 , $\pi/15$	0.02
3	0.03	$\pi/10$	0.03 , $\pi/10$	0.03

Table 8.5: Noise levels tested for each noise type. Noise is inserted in every qubit in every cycle, regardless if it is being operated on. Pauli noise rate refers to probability of inserting and X, Y, or Z gate. Coherent noise is the rotation angle applied. For Pauli+Coherent, only X Pauli gates and X rotations are applied to align with the model in [132]. Amplitude Damping error rate refers to the parameter γ [357], which is the probability of relaxation to the ground state.

in [132]. This includes static X rotations and X Pauli errors. The fourth error model is Amplitude Damping, which is a commonly used and realistic noise model. It models loss of energy from the system to the environment and is a non-unitary process. We sweep the noise over a range of values which are similar to experimental error rates and are expected in near term computers, as listed in Table 8.5. Each noise type is injected in every qubit in every cycle, regardless if the qubit is operated on or not. We assume the same noise rates for single and two-qubit gates. While two-qubit gates will typically have a higher rate of noise in a physical experiment, our values are swept over ranges typically seen for both single- and two-qubit gates. Unless otherwise stated, our metric of choice is the process fidelity of the noisy operation, $\tilde{G}$, to the noiseless operation, G, [100, 105]. If the noisy operation $\tilde{G}$ is free of error, the process fidelity will be 1. We repeat experiments with different input pure states. We generate the input states at random in the same manner as [24], by selecting random polar coordinates on the Bloch sphere for each qubit and report the average process fidelity.

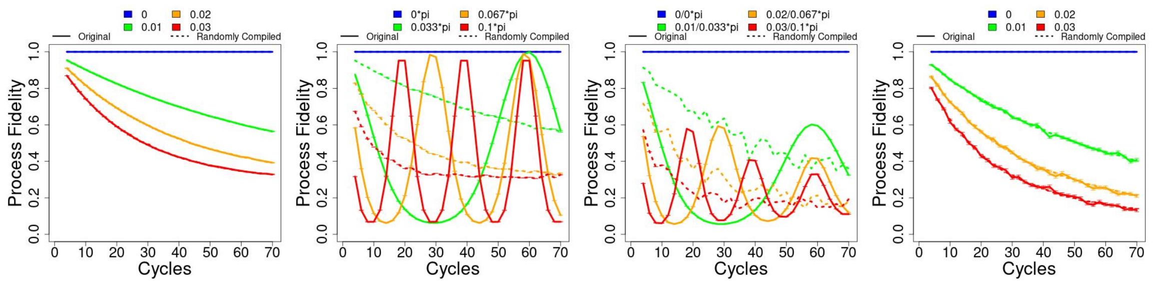
Numerous quantum simulators exist, [147, 338, 345, 169, 302], many of which would be suitable to run our simulations. However, as we are implementing algorithms and incorporating noise models from a variety of sources, and did not want to unintentionally bias our experiments by relying on any specific software, we chose to run our simulations with the statistical programming language R [288]. This allows us to fully and independently define our experiments. Additionally, R is highly optimized to perform matrix multiplication, which is the essential component for density matrix simulation. Our source code is available at [294], which is an extension of our R package QuantumOps [295]. To perform the gate decomposition, we rely on the gridsynth algorithm [322].

Performing simulation with density matrices allows us to avoid much Monte Carlo simulation. However, Monte Carlo simulation is still needed due to the randomness introduced by our random input states, use of RC, and the random circuits for the random circuit benchmark.

8.8 Results

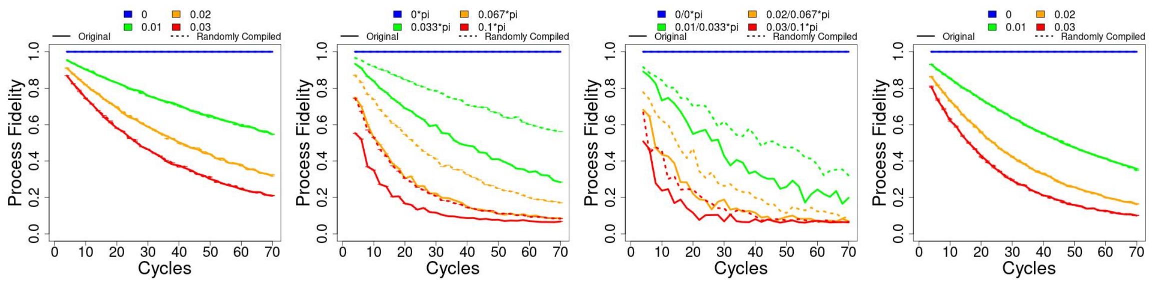
For the Idle and Random circuits we sweep the noise levels over the values listed in Table 8.5 and plot the effect on the fidelity for circuits of different lengths. As the Addition, QFT, and QAOA circuits have a constant depth, we sweep the 4 error models over a fine-grained range and plot the process fidelity or accuracy of the output versus the error rate.

Results for the Idle circuit are shown in Figure 8.2 and results for the Random circuit are shown



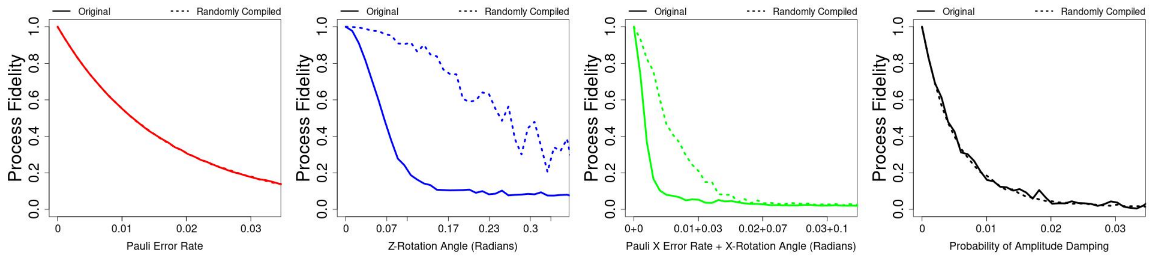
(a) Pauli Noise (b) Coherent Noise (c) Pauli+Coherent Noise (d) Amplitude Damping

Figure 8.2: Idle circuit with 4 qubits under various noise models.



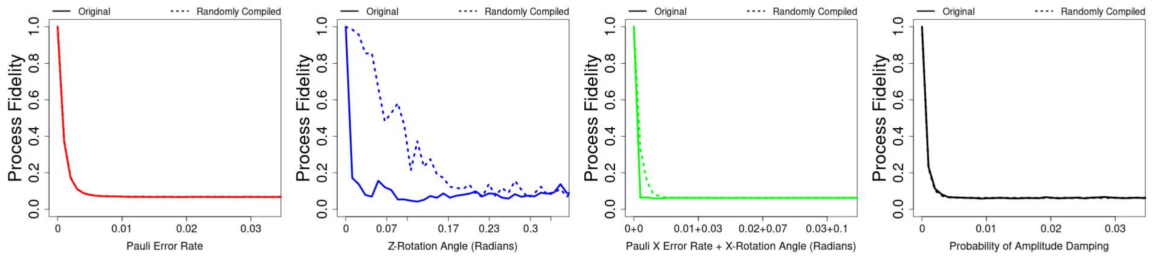
(a) Pauli Noise (b) Coherent Noise (c) Pauli+Coherent Noise (d) Amplitude Damping

Figure 8.3: Random circuit with 4 qubits under various noise models.



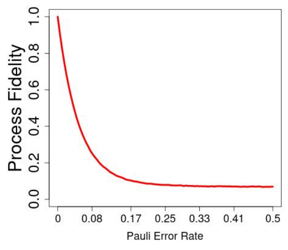
(a) Pauli Noise (b) Coherent Noise (c) Pauli+Coherent Noise (d) Amplitude Damping

Figure 8.4: 2-bit addition circuit under various noise models.

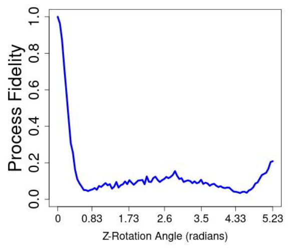


(a) Pauli Noise (b) Coherent Noise (c) Pauli+Coherent Noise (d) Amplitude Damping

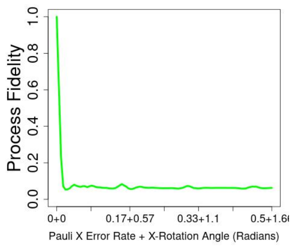
Figure 8.5: 4-qubit QFT (Clifford+T) under various noise models.



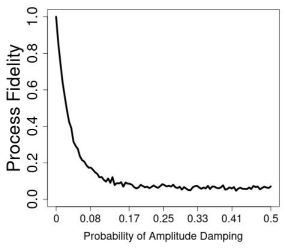
(a) Pauli Noise



(b) Coherent Noise

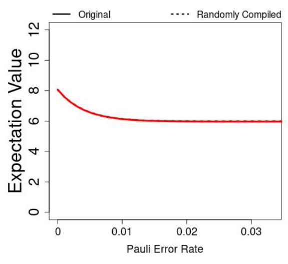


(c) Pauli+Coherent Noise

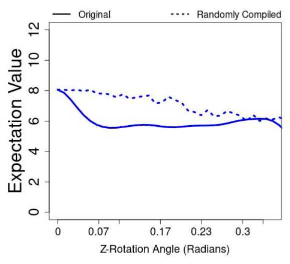


(d) Amplitude Damping

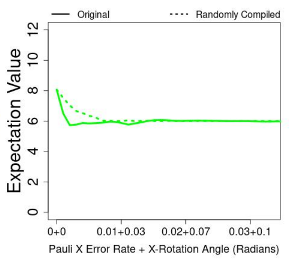
Figure 8.6: 4-qubit QFT with parameterized rotation gates under various noise models.



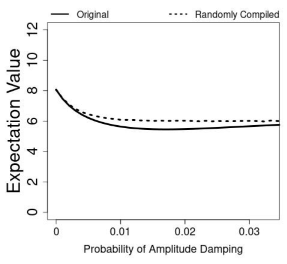
(a) Pauli Noise



(b) Coherent Noise

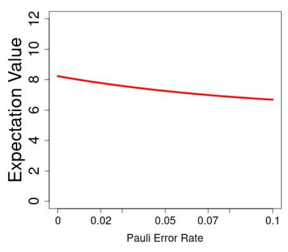


(c) Pauli+Coherent Noise

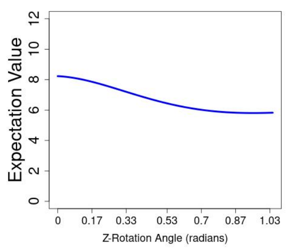


(d) Amplitude Damping

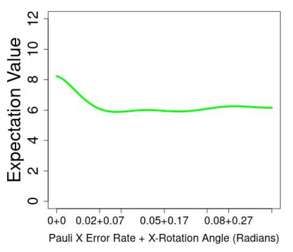
Figure 8.7: QAOA (Clifford+T) expectation value under different noise models.



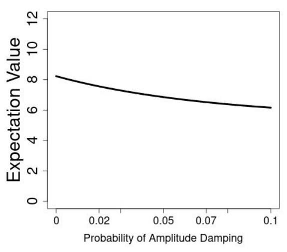
(a) Pauli Noise



(b) Coherent Noise



(c) Pauli+Coherent Noise



(d) Amplitude Damping

Figure 8.8: QAOA with parameterized rotation gates expectation value under different noise models. As the circuit is significantly shorter than Clifford+T QAOA, the noise is tested over a significantly larger range.

in Figure 8.3. Both circuits are performed from 2 to 70 cycles. Cycles here indicate the length of a single gate. Note that error level 0 shows a process fidelity of 1, meaning there is no corruption of the quantum state. It is highly noticeable how coherent noise affects the Idle and Random circuits differently. For the Idle circuit, coherent noise has an immediate drastic impact on the fidelity. We must note the strange behavior of the Idle circuit under coherent noise. Due to our simplified coherent noise model, the fidelity returns with a periodicity determined by the constant angle of rotation. This is unlikely to be a physically realistic phenomenon, and even if it was, *it would not be possible to exploit this fact unless one was completely aware of the exact effects of the physical noise*. As physical noise is difficult to model and predict, it is highly unlikely one would have such knowledge. Note that randomized compiling removes this periodic effect.

The true observation from the simulation of coherent noise on the Idle circuit is the immediate destructive nature of even a slight coherent noise source. Note that randomized compiling mitigates this impact and causes the coherent noise to have the same effect as stochastic Pauli noise. Interestingly, this same coherent noise is not as destructive on the Random circuit. In fact, even without Randomized Compiling, the error decays exponentially just as it does under stochastic Pauli noise. This finding is consistent with [36], which says that coherent noise will affect randomized circuits much differently than idle or cyclic circuits. The coherent noise gets “smeared out” by the randomization inherent in a random circuit. Additionally, this suggests that the randomized benchmark circuits in Quantum Volume [79], depending on the gate decomposition, may not be representative of many quantum circuits. Noteworthy is that Randomized Compiling may not be necessary if the circuit already contains a high degree of randomness. However, the vast majority of useful quantum algorithms do not have such structure.

The chosen quantum noise model has a drastic impact on the performance of quantum algorithms. Hence, one must be sure that the assumptions on the noise present in a physical system are appropriate. Additionally, the effect of the quantum noise is largely determined by the nature of the quantum algorithm being performed. Hence, one must be cautious when choosing quantum algorithms for benchmarks.

Results for the addition circuit are shown in Figure 8.4, the Clifford+T version of QFT in Figure 8.6, and the parameterized rotation gate version of QFT in Figure 9.3. Note that circuits using the parameterized rotation gates cannot be randomly compiled. Randomized Compiling produces a significant increase in fidelity when coherent noise is present.

Note that under Pauli noise or Amplitude Damping, Randomized Compiling does not provide a significant improvement. For Pauli noise, as it is already entirely random, it remains unmodified by the random compilation. This is noted by the inventors of RC [376]. As RC is designed to mitigate coherent rotations of the qubit state, it is also intuitive that it would not significantly mitigate Amplitude Damping, which models energy loss of the system. However, RC may help in some specific circumstances, such as when a qubit is held in the excited 1 for an extended period of time. As Amplitude Damping models the collapse from 1 to 0, the 1 state is more vulnerable. As noted by the authors of [310], Amplitude Damping is more destructive if the quantum data in a program contains more qubits in the 1 state. RC could make the state oscillate, reducing the amount of time the qubit will spend in the excited, more vulnerable state. However, this condition was not present in our benchmarks.

While a critical tool to enable scalable quantum computing, especially for modern machines, Randomized Compiling (RC) cannot be used as a generalized noise mitigation technique. As noted by the original paper [376], randomized compiling is used to "convert" coherent noise to stochastic Pauli noise. This is significant, as modern, physical quantum computers are dominated by coherent errors. Additionally, this conversion to Pauli noise maintains the validity of previously established proofs which have assumed Pauli noise.

Results for the Clifford+T QAOA are shown in Figure 9.2. If no error is present, the QAOA circuit will produce an expectation value of 8.3, as this is 0.6924$\times$ of the maximum cut (12), which is expected for a QAOA circuit with $p = 1$. The expectation value of a random guess for the max-cut problem is at least 50% that of the maximum cut [137]. For the specific problem we used, a random guess produced an expectation value exactly 50% of that maximum. Hence, increasing the noise, which increases the entropy of the output, generally caused the expectation value to decay to 6. Noticeably, the expectation value drops to that of a random guess even for very low levels of noise. Amplitude Damping noise has significant potential to drop the expectation value below 50%, as high levels of this noise will cause the state to transition more towards the $00...0$ state. The $00...0$ state does not satisfy any of the input problem's constraints and hence produces an expectation value of 0. Notice that RC prevents this drop below 50%.

QAOA using parametrized rotation gates is shown in Figure 8.8. Due to not using decomposition, the circuit is much shorter and hence significantly less impacted by noise. Note that this gate set is not compatible with randomized compiling or quantum error correction. Hence, it is not scalable but still feasible for the small cases which we are considering. Despite the difference in gate sets, this version of QAOA shows similar expectation value patterns due to the different noise types. The expectation value tends to converge towards a value that is 50% of the maximum. Again, a combination of Pauli and coherent noise is particularly destructive.

8.9 Conclusion

Computer architecture uses layers of abstraction to manage complex problems. This approach has already been applied to quantum computing. Unfortunately, quantum systems are notorious at defying abstraction and simplifying assumptions. It is easy to make invalid assumptions and generate inaccurate results. Here, we showed that quantum noise is more complex and difficult to model than is often assumed. This has profound effects everywhere, and can be felt significantly even at higher levels of the system stack. This complicates the task of benchmarking, which is already challenging and full of subtlety for classical computers. The noise model, the target application, and the performance metric all need to be carefully considered.

Chapter 9

A day in the life of a quantum error

9.1 Introduction

In the future, quantum computers will have sufficient qubit counts and fidelity in order to perform quantum error correction, and hence will be capable of computations of arbitrary size. This is unfortunately not true for current machines, which have both limited qubit counts and fidelity. Although small in size and noisy, the existing quantum computers with fifty-plus qubits are capable of powerful computations. To leverage existing quantum computers, researchers are developing software techniques to mitigate hardware errors. To that end, recent proposals use machine-specific noise characteristics to increase the likelihood of measuring the correct output on Noisy Intermediate Scale Quantum (NISQ) computers [352, 256, 355]. However, there is a lack of studies on understanding how an error affects the later instructions and the output of quantum programs. Such analysis can be beneficial as it can allow the execution of of noise tolerant parts of a program on less reliable qubits. By exploiting the unique properties of specific programs, potentially larger applications can be run reliably.

Our approach to sensitivity analysis corresponds to the quantum equivalent of well-explored, classical statistical fault injection, where each experiment entails (i) injecting a fault in space or time at a specific point in computation (e.g., by corrupting temporal state, using a representative noise model to best capture the actual manifestation of a given type of fault); (ii) tracking the fault propagation all the way to the end results; and (iii) repeating this procedure for a statistically significant number of times to be able to draw a meaningful conclusion. Similar to this procedure, our experiments (in simulation) involve corruption of one qubit (to capture spatial noise tolerance) and one gate operation (to capture temporal noise tolerance) at a time, in performing a quantum algorithm, and simulating the respective experiment with density matrices to achieve a statistically significant result.

Of interest, is how errors on qubits temporally and spatially propagate when performing a quantum algorithm. If this can be properly characterized, such information can enable novel optimization opportunities in mapping an application on to quantum hardware, to best match the spatio-temporal variation in quantum noise tolerance of the underlying quantum substrate. The idea is mapping noise-sensitive computations to the least noisy components of the quantum computer (spatial scheduling). Additionally, performing gates earlier or later can reduce sensitivity to noise (temporal scheduling). Going even further, characterizing the noise tolerance of specific algorithms could aid in the design of application-specific quantum hardware [203].

In many ways, quantum computing significantly increases the complexity and difficulty of statistical fault injection. As accurate noise models are difficult to develop, simplifying assumptions must be made. Additionally, noise rates are much higher in quantum systems, which makes validation by physical experiment difficult, as errors are not manifested as single isolated events. This leaves as an open question if classically-inspired fault-injection analysis can be effectively used for quantum computers.

9.2 Background

9.2.1 Noise Types

Quantum noise is complex, and often simplifying assumptions are made to ease simulation and analysis [376]. Stochastic Pauli Noise is commonly used, which corresponds to the random insertion of extra X or Z gates. However, this is not a complete representation [44]. More realistic noise models capture Amplitude Damping (AD) and coherent errors. AD is the collapse of a qubit into its low energy state, and is the result of imperfect isolation from the environment. The likelihood of this occurrence can be calculated as a function of experimentally determined coherence times [357]. Notably this can cause loss of entanglement between qubits. For example, if two qubits are in the maximally entangled bell state, $00 + 11$, both qubits are in a superposition and their states depend on each other. If AD occurs on the second qubit, the state will transform to $00 + 10$, where the second qubit is now guaranteed to be in state 0. Additionally, the state of the second qubit no longer depends on the first qubit, and they are no longer entangled.

Coherent noise can be slight over- or under-rotation of qubits due to imperfect control hardware [44, 130, 19, 191, 373], and is the dominant noise source in modern quantum computing systems [376]. Unlike AD, this noise source is unitary. Hence it does not destroy the quantum state, but will move it into a different, and undesired, coherent state [24]. A phase error (Z-rotation) will change the phase between the 0 and 1 states. A Z-rotation by angle θ will transform the quantum state from $\alpha 0 + \beta 1$ to $\alpha 0 + e^{i\theta}\beta 1$. A rotation by π will effectively be a Z gate.

We model noise by causing AD with 100% chance of collapse and by performing Z-rotations by π.[1] This is not to suggest that physical quantum noise manifests in this manner. Rather, we test the spatial and temporal susceptibility of quantum programs to noise of this nature.

In practice, it may be beneficial to use machine-specific noise models. The approach developed here still applies, and can be made machine-specific by simply swapping out the specific noise model. Such machine-specific noise models will change over time and across re-calibration cycles. Regardless, AD and coherent rotations are representative of real-world processes and can provide insight into quantum program sensitivity.

[1]For the case studies used in this work, different AD probabilities and Z-rotation angles affect the severity of the impact, but not do not change the nature of the impact.

9.2.2 Metrics

Evaluating the effects of quantum noise also depends on the metrics of interest. *Process Fidelity* [105] is an intuitive metric as this effectively measures the *distance* between the actual quantum state and the ideal. However, depending on context, this could be misleading. Phase errors will cause degradation of the process fidelity, but will not directly affect measurement outcomes (in the Z-basis). Hence, this may be overly pessimistic. A similar metric, the *Hellinger Fidelity*, focuses exclusively on measurement probabilities. It is a measure of how similar samples from a probability distribution are. This can easily be determined by performing a quantum program multiple times and comparing the measured results to the expected. This metric is useful as it roughly corresponds to how likely one is to get results from the correct set of possible answers. However, it has its own drawbacks. The Hellinger fidelity can vary widely and report pessimistic results if there are only a few possible measurement outcomes in the ideal case. For example, say the ideal case has a 100% chance of measuring 0010010. If there is an X error causing a flip on the last qubit, there will be a 100% chance of measuring 0010011. While these outputs are highly similar, and it may be possible to find the correct answer with post processing, the Hellinger fidelity will be 0 due to no overlap in the output measurements.

Probability of Successful Trial (PST) is another widely used metric. It is the probability of measuring the correct output. In some cases it may also be overly pessimistic. In the above example, the PST would also be 0. PST finds greater applicability if there is no classical means to extrapolate the correct output from a noisy measurement.

9.3 Case Studies

Different quantum circuits (programs) can vary in how they are affected by noise. We use circuits with starkly different characteristics to understand how faults evolve in time and space. Moreover, we use quantum benchmarks, which can be parametrized and scaled without significantly changing the workload structure.

Quantum Adder (ADDER): Quantum adder circuits perform addition on two quantum states. We use quantum adders because (1) Output of quantum adder is trivial to verify using conventional computers (2) Adders have a rich structure that uses Toffoli gates, which are sensitive to both amplitude and phase errors. Moreover, quantum adders are used in Shor's prime factorization, and they are a basic building block of many quantum arithmetic functions.

Bernstein-Vazirani (BV): is a quantum algorithm that demonstrates quantum speedup by querying the oracle only once to find the encoded secrete. On execution, `BV` outputs a binary string corresponding to the secret key. We use an oracle function that uses CNOTs and leverage phase kickback. The sensitivity to phase errors but tolerance against bit-flip errors make BV an interesting case study.

Quantum Approximate Optimization Algorithm: The Quantum Approximation Optimization Algorithm(QAOA) can solve combinatorial optimization problems such as MAX-CUT. Certain characteristics of QAOA make it tolerant to noise. For example, QAOA can be run in a variational mode where it is run multiple times with classical parameter updates in order to achieve an optimal output. One of the unique structural traits of QAOA is that all qubits have an equal number of connections (participate in 2-qubit gates), and the circuit is typically chosen to have a shallow depth. We use QAOA implementation that requires only nearest-neighbor connections. This greatly eases execution on a physical machine as it does not require any qubit swap operations. The input to QAOA is the $00..0$ state.

Quantum Fourier Transform (QFT): The Quantum Fourier Transform (QFT) performs the Discrete-Time Fourier Transform on the amplitudes of a quantum state. It is the core of Shor's algorithm for factorization [331]. The QFT circuit is asymmetrical, for an n-qubit QFT, the least-significant qubit is the target of $n-1$ controlled-rotations and the most-significant qubit is not the target of any controlled-rotations. However, it is the control of $n-1$ such operations.

The input to the QFT circuit is typically a highly entangled state. In the case of Shor's algorithm, the amplitudes are the outputs of the modular exponentiation function. We use an entangled state where the amplitudes form a periodic function, representative of input to the QFT in Shor's algorithm.

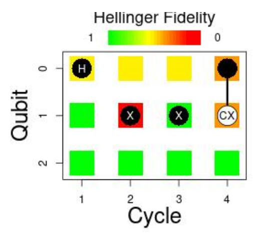
9.4 Fault-Injection Analysis

To determine the sensitivity of quantum circuits we need to insert noise at specific locations and times. This can be done by inserting a noise event on a single qubit during a single cycle of the circuit. While this noise event will have an immediate impact on the quality of the quantum state, and analyzing this immediate impact can build intuition, it is of more interest on how it will impact the state once the entire circuit has been executed. This is for two reasons. One is that only effects visible at the output will degrade the measurement results. Hence, if a noise event induces an error which becomes obscured, we don't care. The second reason is that, unless performing quantum error correction, we will not be able to detect, mitigate, or correct the error in real time during execution. The effects remain invisible until a measurement is performed, which is at the end of the circuit for these near term applications.

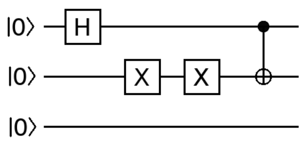
For our simulations, we insert a single noise event on a single qubit in a single cycle and view the impact on the output of the circuit. We repeat this for every qubit and every cycle in each circuit. To accentuate the impact of the noise event, all other operations are noiseless, including gates, idling, and measurement. To visualize the impact, we plot the circuit over the heatmap of the output quality. This way, it is easy to spot the sensitive regions of each circuit.

An example is shown in Figure 9.1. The figure follows the same conventions as quantum circuits, the qubits are stacked vertically and the time flows from left to right. Gates are represented by black circles. White circles are the targets of 2-qubit gates. For clarity, we omit the traditional horizontal lines tracing a qubit in traditional circuit diagrams. The heatmap shows the hellinger fidelity of the *output* if a *single* error occurred at that location in space and time. Hence, each noise location represents an individual experiment, where noise occurred at that location and nowhere else. The color on a gate represents an error which occurs immediately after the gate. The heatmap scales from perfect output, green meaning the hellinger fidelity is 1, to yellow at 0.5, and finally red at 0.

The ideal output of the circuit in Figure 9.1 is $000 + 110$. If AD occurs on qubit 0 on cycle 1, after the H gate, the output will be 000. This has considerable overlap with the desired output, and the hellinger fidelity is 0.46 and the location on the heatmap circuit is indicated as such. If AD occurs after the first X gate on qubit 1, the final output will be $010 + 100$. This has no overlap with ideal



(a) Heatmap Circuit

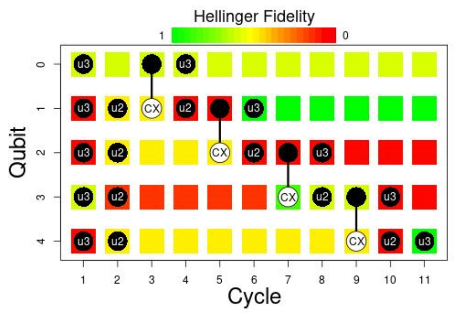


(b) Circuit

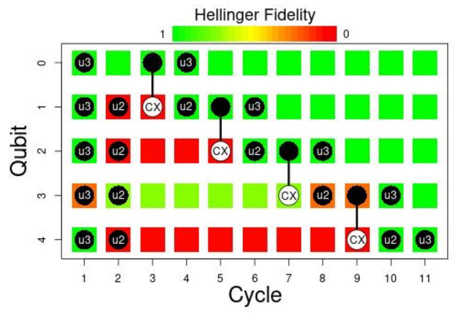
Figure 9.1: Heat map circuit alongside equivalent circuit diagram. Heatmap shows hellinger fidelity of output if AD (with 100% chance of collapse) were to occur at each location.

output, hence the hellinger fidelity is 0. If AD occurs on qubit 1 after the final CX (CNOT) gate, this will cause a loss of entanglement between qubits 0 and 1. The output state will be $000 + 100$, and the hellinger fidelity is 0.29. As qubit 2 remains at 0 the entire time, and is not entangled with the other qubits, AD on qubit 2 has no effect on output.

For noise events, we use both Amplitude Damping (AD), which has a 100% chance of collapse, and coherent Z-rotations by π. For metrics, we show the hellinger fidelity.



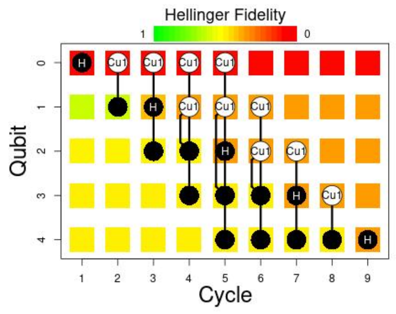
(a) AD



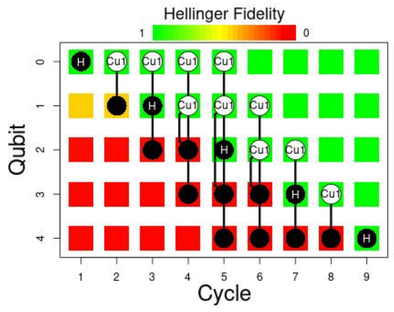
(b) Coherent-Z

Figure 9.2: QAOA

As can be seen, the effect of noise highly depends on the circuit. For circuits which create highly entangled output states, AD tends to have more destructive results later in the program, whereas coherent Z-rotation tends to have more impact earlier in the program. This is intuitive, as AD causes a loss of information and entanglement. The further along the program is, the more entangled the quantum state is and hence a greater loss of information with the non-unitary collapse in AD. Coherent noise will not cause loss of entanglement, and as it is unitary (reversible), does not destroy information in the quantum state. Effectively, it changes the program that is being performed. The earlier this occurs in the circuit, the more opportunity it has to spread to other qubits and further



(a) AD



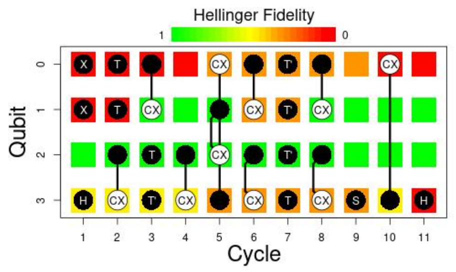
(b) Coherent-Z

Figure 9.3: QFT

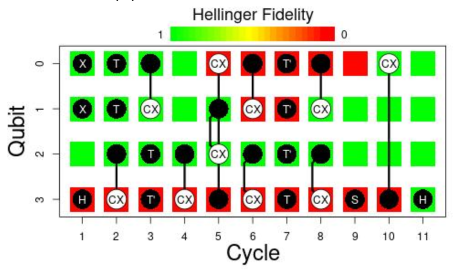
corrupt the state. This trend is observable even in these small scale simulations. It is expected to have an even more significant impact on larger circuits, where the increased circuit depth provides more opportunities for spreading error and entangled states exist for longer periods. Notably phase errors do not cause a change in the measurement outcomes if they occur immediately before measurement. Hence, Z-rotation errors show no degradation if occurring after the last gate on a qubit.

If spatial sensitivity is observed (some qubits/gates in a cycle have worse errors than others), the sensitive gates/qubits can be scheduled on a machine's most reliable physical qubits (spatial gate scheduling). This will only help if these qubits can be mapped correctly to the physical machine without introducing additional swaps, or if the swaps introduce less error than is saved. For example, qubit 0 of QAOA (Figure 9.2) and qubit 2 of the adder (Figure 9.4) show low sensitivity to both AD and coherent noise, relative to other qubits in the circuit. Hence, some robustness may be gained by mapping these logical qubits to less reliable physical qubits in physical experiment. This technique can be added on top of well-studied methods of mapping quantum circuits to the most reliable physical qubits [355]. Effectively, in addition to using the overall most reliable qubits, the reliability of the physical qubits used are matched to sensitivity of the logical circuit.

Temporal gate scheduling also plays an important role. Exemplified by BV (Figure 9.5), performing gates as late as possible can reduce sensitivity. If all qubits are initiated and put into a superposition with H gates early, this leaves them vulnerable to both AD and coherent errors for extended periods of time. In this case, scheduling the initial H gates later clearly provides superior performance, as the qubits are more resilient in the 0 state. However, this is not universally true. For QAOA (Figure 9.2), performing the second u2 gate on qubit 3 later (immediately prior to following CX gate) would reduce its susceptibility to AD noise. However, this would have the opposite effect if the noise source is coherent, where the delay in u2 would leave the state more susceptible.



(a) AD



(b) Coherent-Z

Figure 9.4: Adder

Table 9.1: PST of BV-5 on IBM hardware for eager and delayed scheduling

Schedule	Valencia	Casablanca	Bagota
Eager	0.61	0.74	0.45
Delayed	0.73	0.82	0.65

This condition is also seen for qubit 4, but with the reverse conclusion. Hence, gate scheduling not only critically depends on the circuit but the noise source. Both must be considered for optimal scheduling.

The high-level insights from fault-injection studies hold on a real hardware. For instance, we observe substantial reliability improvements by using a delayed schedule, similar to what we observe for the fault-injection studies with coherent-Z and AD noise in the Figure 9.5. Table 9.1 reports a Probability of Successful Trial (PST) for five qubit Bernstein-Vazirani(BV) for two scheduling policies - Eager and Delayed when executed on IBM quantum computers with 5 to 7 qubits. Note that both instance of circuits with eager and delayed schedule have identical gate count and physical to logical mapping, the only difference is the instruction order.

9.5 Conclusion

The well studied classical error injection process is also applicable to quantum computers. Finding critical and noise-tolerant regions will enable an application to be adapted to both mitigate the effect of quantum noise and save precious resources. The errors that arise from noise are highly spatially and temporally dependent and are determined by the circuit that is being performed. Unfortunately, quantum computers have additional complications. The errors are also highly depended on the nature of the quantum noise, which is not well understood. Assumptions made about the noise become critical to finding meaningful results and techniques which will remain valid in experimentation.

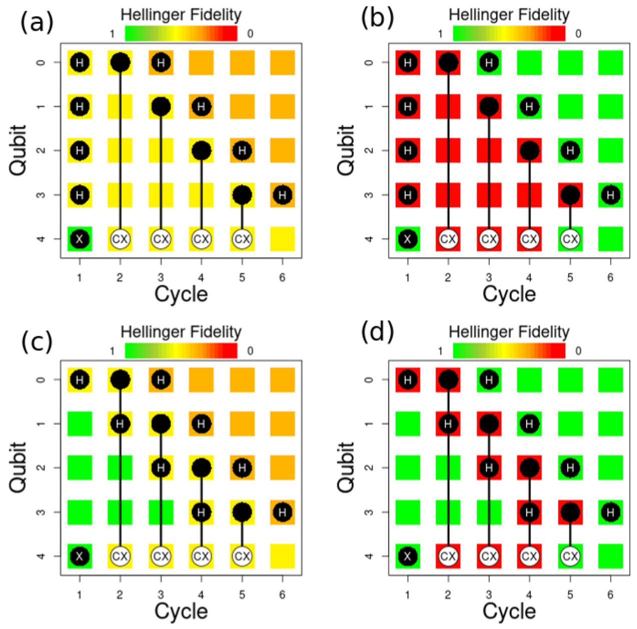


Figure 9.5: **Eager Bernstein-Vazirani:** (a) with AD noise (b) Coherent-Z noise, **Late Berstein-Vazirani:** (c) with AD noise (b) Coherent-Z noise

Chapter 10

Variable Strength QEC

10.1 Introduction

Quantum noise is hard to model accurately as it is highly complex and can have counter-intuitive impacts [100]. Additionally, the noise present in a physical system can change over time [355], making it difficult to properly characterize it via benchmarking procedures [386].

Quantum error correcting codes (QECC) group collections of physical qubits together to represent one logical qubit (each qubit in a quantum algorithm corresponds to a logical qubit). The logical qubit is much more resilient to noise than the individual physical qubits it is made of. A key property of QECC is that it translates the arbitrary noise on physical qubits into a discrete set of noise events on logical qubits. This allows us to accurately model the quantum noise acting on logical qubits with just a few types of noise events (i.e., X and Z errors [337]).

For example, physical quantum noise can lead to slight over- or under-rotations (considering individual qubit state representation in polar coordinates) [44]. This noise is analog in nature, as the angle of over- or under- rotation can be arbitrary. QEC involves measurement, which introduces *non-unitary* transformations to the physical quantum state (while leaving the computational quantum state intact): Effectively, each measurement forces a binary decision on the impact of noise – either the qubit snaps back to the uncorrupted state, removing the noise, or a complete noise event is the case, “flipping” the state of the qubit. This digitizes noise into a set of discrete errors, enabling detection and correction.

QECC can only detect and correct errors on physical qubits if the physical noise induced error rates are sufficiently low. An excessive number of noise events on multiple physical qubits, occurring simultaneously, may result in a *logical* error which is either undetectable or uncorrectable. QECC is typically designed with the target of removing logical noise events entirely. The maximum number of simultaneous physical errors QECC can tolerate is called the *code distance* which determines the *code strength*.

Unfortunately, the cost of creating large distance QECC is very high. The number of physical qubits required for each logical qubit grows quadratically with the code distance, potentially reaching thousands of physical qubits for each logical qubit [108] even for modest error rates. The resource requirement quickly grows beyond what currently available quantum hardware can realistically support.

In this work we explore how opportunistically reducing the code distance and thereby risking rare logical errors can help mitigate the hardware resource overhead required to build a (nearly) fault-tolerant quantum computer. This strategy can prove effective to the extent the quantum application can tolerate rare errors – i.e., can still produce the correct output with high probability. Two basic methods span the entire design space:

1. Lowering the code distance overall (for all logical qubits at all times)
2. Selectively lowering the code distance for a subset of the qubits at specific times

Method 2) is especially suitable for Surface Codes [108] where changing the number of physical qubits dedicated to each logical qubit is possible. For both cases, we use statistical fault injection (i) to accurately estimate the probability of success in the presence of errors; (ii) to differentiate more noise-sensitive regions of the application from the less noise-sensitive to allocate scarce QECC resources based on need.

10.2 Background

Improving the reliability of quantum programs considering physical qubit characteristics is a well studied problem. Numerous papers such as [355] explored application mapping strategies considering variation in the reliability of physical qubits, while others focused on minimizing the number of gate evaluations to reduce the exposure time to noise [93, 205]. Statistical fault injection based studies of program sensitivity to physical noise, to optimize gate scheduling and qubit placement, also exist [297]. An important distinction of our study from this lineage of work is that we operate at the logical qubit level rather than the physical. Noise can be modeled more accurately at the logical level as QECC effectively forces noise to manifest as X and Z errors. This is in stark contrast to modeling noise at the physical level, which involves many different models. Worse, each noise model can result in a different outcome [299], and the noise in a physical experiment changes over time [386]. As a result, statistical fault injection at the physical level can produce contradicting conclusions depending on the noise model used [297]. At the same time, we primarily focus on fine-tuning QECC distance according to algorithmic needs, as opposed to common noise mitigation strategies at the physical level which focus on application mapping/scheduling.

10.2.1 Surface Codes

Surface codes are a promising form of QECC. They logically arrange qubits into a two-dimensional lattice, and only require interactions between nearest neighbors [108, 214], allowing the qubits to remain in place. This makes them easy to use with modern quantum computers, such as superconducting quantum computers, which have stationary qubits and only allow interactions between physically adjacent qubits. Surface codes can be conceptually visualized as "patches" of physical qubits, where patches can move and interact with each other by performing operations and measurements on the qubits [214]. This is referred to as *lattice surgery*, and represents the state of the art [214, 160]. A surface code of code distance d requires d^2 physical qubits per logical qubit. Logical gates on logical qubits require roughly d time cycles [214]. Changing d for each qubit also roughly takes d cycles. To increase d: initialize more physical qubits and involve them in the next round of error correction. To decrease d: measure a subset of the physical qubits in the logical qubit, and then exclude them in the next round [108]. For both, then perform d rounds of error correction to remove any errors which occurred during the process.

10.2.2 Gate Decomposition

A qubit can be logically represented as a point on the unit sphere (called Bloch sphere), where operations (gates) on it become rotations around different axis. $R_x(\theta)$, $R_y(\theta)$, and $R_z(\theta)$ gates correspond to rotations around the x, y, and z axes by an arbitrary angle θ. Such rotation gates are often used when operating quantum computers without QEC, where logical gates correspond directly to physical rotations on individual qubits. The specific gate set available depends on the specific machine, but generally precise rotations around at least two axes are typically available. However, such rotations do not work directly with surface codes (or most QECC). When the physical qubits are grouped together to form a logical qubit, only a specific set of gates are possible [108]. In this work we use the Clifford+T gate set, the most widely used universal gate set which can also be used on surface codes. It includes, among others, X, Y, Z, H, S, and T gates, which all correspond to rotations by π, $\pi/2$, or $\pi/4$ around various axes. Two-qubit variations of these gates exist, where a gate is performed on a *target* qubit only if the *control* qubit is in a specific (i.e., the 1) state. For example, the CNOT gate is a controlled X gate. A controlled phase gate is a controlled R_z gate.

Logical versions of $R_x(\theta)$, $R_y(\theta)$, and $R_z(\theta)$ gates are required for many algorithms, hence it is still necessary to implement them. To perform them on surface codes, they are *approximated* with sequences of gates. For this, we use the *gridsynth* algorithm [323, 322], which converts $R_z(\theta)$ into sequences of X, H, S, and T gates. The length of the sequence depends on the angle of rotation, θ, and the precision to which we need to approximate it. For example, a rotation by $\pi/3$ can be approximated with Equation 10.1.

$$R_z(\pi/3) \approx HSTHSHTHSHTST \tag{10.1}$$

There is also a trade-off between the sequence length and the achieved accuracy. Shorter sequences take less time to perform but can lead to errors in the program (even in the absence of noise) due to the higher degree of approximation. We tested different sequence lengths when performing a noiseless simulation of a benchmark quantum algorithm (which we discuss in Section 10.3.2). As an example, based on these experiments, we observe that for an average sequence length of 34 gates per rotation, the correct output was produced only 50% of the time. Increasing the sequence length to an average of 44 gates increased the probability of correct output to 98.5%.

Luckily, it is sufficient to approximate R_z rotations, as any quantum operation U can be decomposed into a set of three R_z rotations (using angles β, γ, and δ) and H gates:

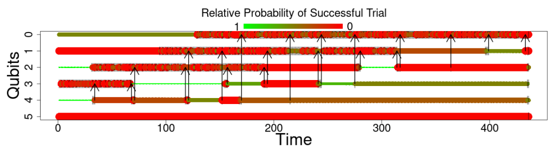
$$U = R_z(\beta)\ R_x(\gamma)\ R_z(\delta) = R_z(\beta)\ H\ R_z(\gamma)\ H\ R_z(\delta) \tag{10.2}$$

In our case study we use standard Clifford+T gates, however, we note that further optimizations exist to tailor the operations specifically for surface codes [214].

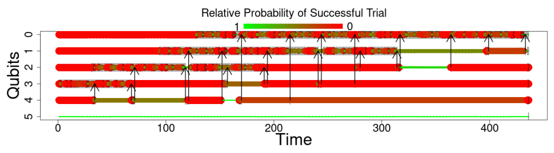
10.3 The Case for Variable Strength QECC

10.3.1 Latency vs. Reliability Trade-Off

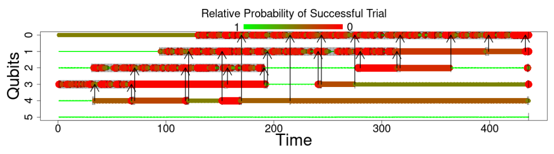
Reducing the QECC distance (i.e., making QECC weaker), by construction reduces the overhead, and hence, the time it takes to complete the program. On the other hand, a smaller code distance reduces the probability of success. Most quantum algorithms, even in the absence of noise, pro-



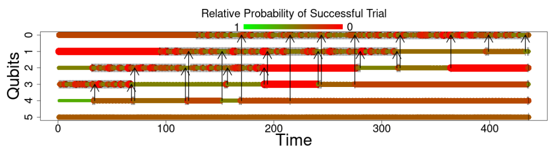
(a) X error only



(b) Z error only



(c) XZ error



(d) Realistic 50% X and 50% Z error

Figure 10.1: Noise sensitivity heatmap of 6-qubit QPE alrogithm to different logical errors. Thin green lines represent insensitive; thick red lines, sensitive regions. Vertical lines show single qubit gates. Arrows between qubits show two qubit gates. Qubit 5 has no operations on it after state preparation. In our noise model, an error on a two-qubit gate affects both qubits.

duce the correct result with some probability. Therefore, the algorithm has to be run multiple times before the correct answer is produced statistically. The number of repetitions depends on the probability of a successful trial (PST). $1/PST$ runs are required to get the correct result, on average. Hence, *mean time to success* is a key metric of interest. If the algorithm has a latency of L, the mean time to success becomes L/PST. Let L_{weak} and PST_{weak} denote the latency and probability of success of the target algorithm using a weaker (i.e., smaller distance) QECC; where L and PST denote the latency and probability of success of the target algorithm with default strength (i.e., distance) QECC.

$$\begin{aligned} L_{weak} &< L \\ PST_{weak} &< PST \end{aligned} \tag{10.3}$$

applies, and a weaker QECC would only work if

$$\frac{L_{weak}}{PST_{weak}} < \frac{L}{PST}. \tag{10.4}$$

In the following, we quantitatively analyze this trade-off.

10.3.2 Sensitivity to Noise in Time and Space

The impact of a logical error on the success of a quantum program depends on when (at which cycle in the execution) and where (in which qubit) the error occurs [297]. We use *statistical fault injection* to characterize this behavior, where we inject errors at different locations at different times and track the propagation to the output of the program. Since we are working with QEC, we can assume that errors are restricted to X and Z errors (Y errors are a combination of X and Z errors). As a representative case study, we profile the Quantum Phase Estimation (QPE) algorithm, which incorporates the inverse Quantum Fourier Transform (QFT) as the main computational kernel. QPE is representative of algorithms a fault-tolerant quantum computer (i.e., a quantum computer which features QEC) is likely to run, and has important applications in quantum chemistry [197]. The input to QPE is a quantum state which has a phase angle difference between its constituent qubits. QPE detects this difference and produces a bitstring representing the angle. To produce the input quantum state, we perform noiseless state preparation. A sequence of controlled-phase gates are performed with qubits 0-4 as control and qubit 5 as the target. A noiseless QPE implementation acting on this state produces a single output bitstring with high probability ($> 98\%$). This eases validation of the output.

For statistical significance, we simulate each fault with 1,000,000 shots on Qiskit [78]. Each fault is a logical error on a single qubit at a single moment in time, leaving all other qubits undisturbed. We test with X error, Z error, and XZ error (both). To model realistic noise, we assume the error is 50% X and and 50% Z, excluding the rare case where both occur. We use this model for the remainder of the paper.

This model is realistic because if the probability of X and Z errors is p, the probability of X (without Z) and Z (without X) is $p(1-p)$, but the probability of both is p^2. For small p, $p(1-p) >> p^2$. Thus, error is predominantly either X or Z.

We evaluate the quality of the output by comparing the probability of successful trial (PST) to that of the noiseless output. As we set the input state so that there is only one correct answer, this metric is Relative PST (Equation 10.5).

$$Relative\ PST = \frac{PST_{noisy}}{PST_{ideal}} \tag{10.5}$$

Figure 10.1 depicts the relative PST as a heatmap, to help visually inspect the significance of the error at each location and at each point in time. The x-axis is time; the y-axis, space (i.e., different qubits). Thick, red lines indicate sensitive regions. Some regions are more tolerant to noise than others, enabling exploitation of variable distance codes.

10.4 Variable Strength QEC

Knowledge about the underlying noise sensitivity of a quantum program, both in time and space, makes variable strength (i.e., distance) QEC possible, where different logical qubits get protected by different levels (i.e., different code distances d) of QEC at different times. If a logical qubit is less susceptible to logical noise at a point in time, it can us lighter weight QEC than more susceptible qubits.

We start by acknowledging a fundamental limitation to this idea. The logical error rate decreases *exponentially* with d. The physical qubit count increases with d^2. The time overhead increases linearly with d. ***Hence, we can save quadratic space and linear time, but risk exponential increases in error rates.*** This suggests that variable-strength QEC can easily backfire if applied too aggressively.

10.4.1 Success Rate as a Function of Code Distance

We estimate the probability of logical error, P_L, from the physical error rate, p, for code distance d with the analytical formula provided by Fowler et al. for surface codes [108]:

$$P_L \approx 0.03 \times (p/0.0057)^{(d+1)/2} \tag{10.6}$$

The probability of successful trial (PST) is the probability of measuring the correct result at the end of the quantum program. With sufficiently high d, no logical errors would occur, and PST would be 1 (for quantum programs that have a single correct output). As d decreases, PST decreases exponentially until it hits nearly 0. The smallest d that is acceptable is a function of the physical error rate.

Using information from Figure 10.1, we know which logical qubits are more sensitive to noise, which we leverage to designate a variable distance QECC. We experiment with single- and two-

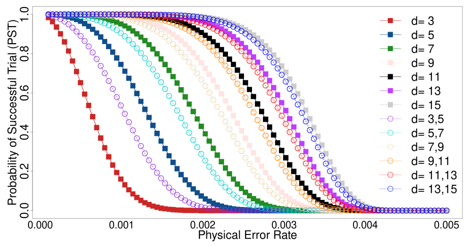


Figure 10.2: PST of QPE for different (including variable) surface code distances, over a range of physical noise rates.

distance QECC: For example, $3, 5$ is a QECC which uses $d = 3$ on less susceptible qubits and $d = 5$ on more susceptible qubits. We consider qubits to be susceptible if the PST is below 40% in Figure 10.1d.

We estimate PST from the probability of two events:

1. No logical error occurs

2. A single logical error occurs, but the output is the same as in the case of no error

These two probabilities combined provide a lower bound on PST. It is also possible that two or more logical errors occur and the output remains the same. However, counting these possibilities quickly becomes intractable because a total of $(N_{gates})^{N_{errors}}$ must be considered, where N_{gates} is the number of quantum gates performed in the algorithm and N_{errors} is the number of possible errors. Hence, in our analysis, two or more errors represent a failure.

The PST for all d are shown in 10.2. The dashed lines capture the variable (two-) distance QECC, which achieve resilience in-between their constituent code distances.

10.4.2 Time to Solution

We now combine the PST information from Section 10.3.2 with the latency of each logic operation for a given code distance, to estimate the time to solution. As noted in Section 10.3.1, the time to solution corresponds to L/PST. A gate on a logical qubit with distance d takes d cycles to complete. Since the distance of each logical qubit is known throughout the program, we can easily find the corresponding latency, i.e., L/PST which provides the time to solution as shown in Figure 10.3. This is a best-case analysis for variable QEC, considering only the overhead for (logical) gate times and code distance conversion. There are additional overheads in lattice surgery or magic state distillation, see Section 10.4.3.

It is noteworthy that the optimal code distance depends on the error rate. As intuition suggests, at low error rates lower code distances are preferable due to the lower overhead. However, as the error rate increases the codes begin to fail. Since error suppression is exponential, once the codes begin to fail, the reliability degrades dramatically and $P_{success}$ drops quickly. This necessitates a larger number of trials to obtain the correct solution, causing high latency.

Each variable distance code is optimal within a range of error rates. For example, the QECC $d = 3, 5$ is optimal at error rates where $d = 3$ begins to fail. $d = 3, 5$ can provide a faster solution than $d = 5$, until it breaks down and $d = 5$ becomes necessary to tolerate errors. The range where each code distance is optimal is listed in Table 10.1. It is possible to combine significantly different code distances. For example, using $d = 7, 15$ instead of $d = 7, 9$. However, this is sub-optimal. When $d = 7, 9$ is optimal, $d = 7$ is failing, but $d = 9$ remains strong. Hence, the additional protection provided by $d = 15$ is overkill, and consumes resources unnecessarily.

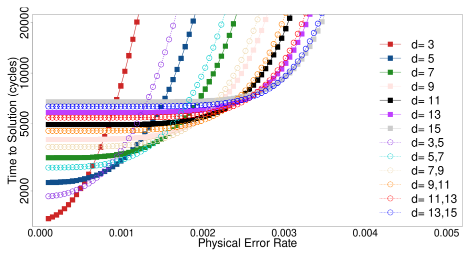


Figure 10.3: Time to solution for different code distances.

Table 10.1: Physical noise ranges of code distance optimality.

Noise Range	Optimal Code Distance
$<=$ 5e-4	3
5.1e-4 - 7.6e-4	3,5
7.7e-4 - 1.0e-3	5
1.01e-3 - 1.38e3	5,7
1.39e-3 - 1.47e-3	7
1.48e-3 - 1.89e-3	7,9
1.90e-3 - 2.26e-3	9,11
2.27e-3 - 2.58e-3	11,13
2.59e-3 - 2.95e-3	13,15
$>$ 2.96e-3	15

10.4.3 Practical Considerations

Variable distance QECC is promising, but practical limitations exist. Superconducting quantum architectures, for which surface codes are most appropriate, logical qubits are arranged next to each other in a two-dimensional lattice [168]. This enables trading physical qubits between neighboring logical qubits. In Figure 10.1d, qubit 3 can use more physical qubits initially, but then transfer them to qubit 4 when qubit 4 becomes more sensitive. However, trading qubits may create non-uniform layouts which do not match well with the physical topology, possibly wasting qubits. Additionally, latency of such state-of-the-art quantum computers is not limited by logic gates, but by the preparation of special quantum states required to perform specific operations (such as magic state distillation for T gates [214]). Hence, improvements in the gate latency from variable strength QECC may not be significant. Also, quantum fault injection is tractable only for small circuits. Obtaining accurate sensitivity estimates for larger circuits poses a challenge, due to the inability to simulate such circuits. However, it may be possible to identify patterns in small circuits (Figure 10.1), and use this to predict sensitive regions in larger circuits.

Finally, physical hardware noise rates will also impact optimal code distances. Our analysis exploits “software” sensitivity at the *logical* level. In practice, this information can be combined with *physical* hardware noise rates to find the optimal distance, a machine dependent optimization.

10.5 Conclusion

Our profiling analysis based on statistical fault injection shows that quantum programs can be relatively insensitive to isolated logical errors, and that variable distance QECC can reduce the time to solution by exploiting the spatio-temporal differences in noise sensitivity. However, any decrease in code distance due to variable strength QECC creates an exponential increase in logical failure rates, which may eliminate the benefits if not carefully administered.

Chapter 11

Conclusion

In this thesis we showed that architectures can be specifically tailored to their environment by exploiting unique properties of their hardware or characteristics of their target applications. These accelerators in extreme environments are able to function well where more traditional architectures struggle, as they are especially well suited to tolerate distinctive challenges in their respective environments.

We showed the non-volatile processing-in-memory (NV-PIM) architectures provide efficient and automatic checkpointing, which enables them to operate with ease in the beyond edge domain. The high cost checkpointing mechanisms of more traditional architectures are not necessary, allowing NV-PIM maintain near ideal performance and energy-efficiency. We also showed that NV-PIM has an inherent resilience to radiation and wide operating temperature. This allows it to operate in physically harsh environments. This opens up a wide range of applications. We developed an architecture which can function in low-earth orbit, and by exploiting the ideal properties of NV-PIM, remains highly performant and energy efficient. The extreme energy efficient of NV-PIM allows it to perform computations beyond the edge that would otherwise be highly impractical. We developed an architecture which is capable of performing encrypted computing on a modest power budget, which goes a long way to solving the security issues that beyond edge devices introduce.

We also investigated NV-PIM performance in cryogenic environments to enable high performance applications like cryogenic computing and quantum computing. Despite some non ideal characteristics of low-temperature operation, NV-PIM remains a suitable solution to provide both memory and computational support to cryogenic systems. The extreme energy efficiency of NV-PIM enables it to operate while generating relatively low amounts of heat, preventing the cooling budget of cryogenic systems of growing to large.

Much of our work advanced the art of quantum accelerators, where we applied traditional computer architecture concepts to quantum computers. We showed that statistical fault injection can be performed to identify sensitive regions of quantum programs, allowing proper allocation of quantum hardware resources. This increases the reliability of near-term quantum accelerators. We combined this method with ideas related to approximate computing to lower the overhead of quantum error correcting codes. Tuning the strength of error correction to match the need to quantum programs allows us to accelerate quantum programs. This strategy can also lower the quantum hardware resources in order to each fault-tolerance, and may help speed the development of large scale quantum accelerators.

Chapter 12

Bibliography

[1] https://www.everspin.com/supportdocs/EMD3D256M08G1-150CBS1, 2019. Accessed: 2019-08-10.

[2] https://www.everspin.com/family/emd4e001g?npath=3557, 2019. Accessed: 2019-11-25.

[3] Scott Aaronson. Shadow tomography of quantum states. *SIAM Journal on Computing*, (0):STOC18–368, 2020.

[4] Scott Aaronson. Turn down the quantum volume, 2020.

[5] Scott Aaronson and Lijie Chen. Complexity-theoretic foundations of quantum supremacy experiments. *arXiv preprint arXiv:1612.05903*, 2016.

[6] Scott Aaronson and Daniel Gottesman. Improved simulation of stabilizer circuits. *Physical Review A*, 70(5):052328, 2004.

[7] Héctor Abraham, Ismail Yunus Akhalwaya, Gadi Aleksandrowicz, Thomas Alexander, Gadi Alexandrowics, Eli Arbel, Abraham Asfaw, Carlos Azaustre, Panagiotis Barkoutsos, George Barron, Luciano Bello, Yael Ben-Haim, Lev S. Bishop, Samuel Bosch, David Bucher, CZ, Fran Cabrera, Padraic Calpin, Lauren Capelluto, Jorge Carballo, Chun-Fu Chen, Adrian Chen, Richard Chen, Jerry M. Chow, Christian Claus, Andrew W. Cross, Abigail J. Cross, Juan Cruz-Benito, Cryoris, Chris Culver, Antonio D. Córcoles-Gonzales, Sean Dague, Matthieu Dartiailh, Abdón Rodríguez Davila, Delton Ding, Eugene Dumitrescu, Karel Dumon, Ivan Duran, Pieter Eendebak, Daniel Egger, Mark Everitt, Paco Martín Fernández, Albert Frisch, Andreas Fuhrer, Julien Gacon, Gadi, Borja Godoy Gago, Jay M. Gambetta, Luis Garcia, Shelly Garion, Gawel-Kus, Leron Gil, Juan Gomez-Mosquera, Salvador de la Puente González, Donny Greenberg, John A. Gunnels, Isabel Haide, Ikko Hamamura, Vojtech Havlicek, Joe Hellmers, Łukasz Herok, Hiroshi Horii, Connor Howington, Wei Hu, Shaohan Hu, Haruki Imai, Takashi Imamichi, Raban Iten, Toshinari Itoko, Ali Javadi-Abhari, Jessica, Kiran Johns, Naoki Kanazawa, Anton Karazeev, Paul Kassebaum, Vivek Krishnan, Kevin Krsulich, Gawel Kus, Ryan LaRose, Raphaël Lambert, Joe Latone, Scott Lawrence, Peng Liu, Panagiotis Barkoutsos ZRL Mac, Yunho Maeng, Aleksei Malyshev, Jakub Marecek, Manoel Marques, Dolph Mathews, Atsushi Matsuo, Douglas T. McClure, Cameron McGarry, David McKay, Srujan Meesala, Antonio Mezzacapo, Rohit Midha, Zlatko Minev, Renier Morales, Prakash Murali, Jan Müggenburg, David Nadlinger, Giacomo Nannicini, Paul Nation, Yehuda Naveh, Nick-Singstock, Pradeep Niroula, Hassi Norlen, Lee James O'Riordan, Pauline Ollitrault, Steven Oud, Dan Padilha, Hanhee Paik, Simone Perriello, Anna Phan, Marco Pistoia, Alejandro Pozas-iKerstjens, Viktor Prutyanov, Jesús Pérez, Quintiii, Rudy Raymond, Rafael Martín-Cuevas Redondo, Max Reuter, Diego M. Rodríguez, Mingi Ryu, Martin Sandberg, Ninad Sathaye, Bruno Schmitt, Chris Schnabel, Travis L. Scholten, Eddie Schoute, Ismael Faro

Sertage, Yunong Shi, Adenilton Silva, Yukio Siraichi, Seyon Sivarajah, John A. Smolin, Mathias Soeken, Dominik Steenken, Matt Stypulkoski, Hitomi Takahashi, Charles Taylor, Pete Taylour, Soolu Thomas, Mathieu Tillet, Maddy Tod, Enrique de la Torre, Kenso Trabing, Matthew Treinish, TrishaPe, Wes Turner, Yotam Vaknin, Carmen Recio Valcarce, Francois Varchon, Desiree Vogt-Lee, Christophe Vuillot, James Weaver, Rafal Wieczorek, Jonathan A. Wildstrom, Robert Wille, Erick Winston, Jack J. Woehr, Stefan Woerner, Ryan Woo, Christopher J. Wood, Ryan Wood, Stephen Wood, James Wootton, Daniyar Yeralin, Jessie Yu, Laura Zdanski, Zoufalc, anedumla, azulehner, bcamorrison, drholmie, fanizzamarco, kanejess, klinvill, merav aharoni, ordmoj, tigerjack, yang.luh, and yotamvakninibm. Qiskit: An open-source framework for quantum computing, 2019.

[8] Shaizeen Aga and et.al. Compute caches. In *HPCA 2017*, 2017.

[9] Amogh Agrawal and et.al. X-sram: Enabling in-memory boolean computations in cmos static random access memories. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 65(12), 2018.

[10] Ahmad Qaisar Ahmad Al Badawi, Yuriy Polyakov, Khin Mi Mi Aung, Bharadwaj Veeravalli, and Kurt Rohloff. Implementation and performance evaluation of rns variants of the bfv homomorphic encryption scheme. *IEEE Transactions on Emerging Topics in Computing*, 2019.

[11] Rafael N Alexander, Peter S Turner, and Stephen D Bartlett. Randomized benchmarking in measurement-based quantum computing. *Physical Review A*, 94(3):032303, 2016.

[12] Elia Ambrosi, Alessandro Bricalli, Mario Laudato, and Daniele Ielmini. Impact of oxide and electrode materials on the switching characteristics of oxide reram devices. *Faraday discussions*, 213:87–98, 2019.

[13] Matthew Amy, Dmitri Maslov, Michele Mosca, and Martin Roetteler. A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(6):818–830, 2013.

[14] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, and Jorge Luis Reyes-Ortiz. A public domain dataset for human activity recognition using smartphones. In *Esann*, 2013.

[15] Faycal Ait Aouda, Kevin Marquet, and Guillaume Salagnac. Incremental checkpointing of program state to nvram for transiently-powered systems. In *2014 9th International Symposium on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*, pages 1–4. IEEE, 2014.

[16] Sara Arabi, Essaid Sabir, and Halima Elbiaze. Information-centric networking meets delay tolerant networking: Beyond edge caching. In *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6. IEEE, 2018.

[17] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

[18] Eyal Bairey, Itai Arad, and Netanel H Lindner. Learning a local hamiltonian from local measurements. *Physical review letters*, 122(2):020504, 2019.

[19] Ball et al. Effect of noise correlations on randomized benchmarking. *Physical Review A*, 2016.

[20] Harrison Ball, Thomas M Stace, Steven T Flammia, and Michael J Biercuk. Effect of noise correlations on randomized benchmarking. *Physical Review A*, 93(2):022303, 2016.

[21] Domenico Balsamo, Anup Das, Alex S Weddell, Davide Brunelli, Bashir M Al-Hashimi, Geoff V Merrett, and Luca Benini. Graceful performance modulation for power-neutral transient computing systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(5):738–749, 2016.

[22] Domenico Balsamo, Alex S Weddell, Anup Das, Alberto Rodriguez Arreola, Davide Brunelli, Bashir M Al-Hashimi, Geoff V Merrett, and Luca Benini. Hibernus++: a self-calibrating and adaptive system for transiently-powered embedded devices. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(12):1968–1980, 2016.

[23] Domenico Balsamo, Alex S Weddell, Geoff V Merrett, Bashir M Al-Hashimi, Davide Brunelli, and Luca Benini. Hibernus: Sustaining computation during intermittent supply for energy-harvesting systems. *IEEE Embedded Systems Letters*, 7(1):15–18, 2014.

[24] Jeff P Barnes, Colin J Trout, Dennis Lucarelli, and BD Clader. Quantum error-correction failure distributions: Comparison of coherent and stochastic error models. *Physical Review A*, 95(6):062338, 2017.

[25] Mark Barton and Jon Miller. Modular thermal design concepts: Thermal design of a spacecraft on a module level for leo missions. 2005.

[26] Robert Baumann. Soft errors in advanced computer systems. *IEEE Design & Test of Computers*, 22(3):258–266, 2005.

[27] Todd Bayer, Brent Buffington, Jean-Francois Castet, Maddalena Jackson, Gene Lee, Kari Lewis, Jason Kastner, Kathy Schimmels, and Karen Kirby. Europa mission update: Beyond payload selection. In *2017 IEEE Aerospace Conference*, pages 1–12. IEEE, 2017.

[28] Stefanie J Beale, Joel J Wallman, Mauricio Gutiérrez, Kenneth R Brown, and Raymond Laflamme. Quantum error correction decoheres noise. *Physical review letters*, 121(19):190501, 2018.

[29] Arnout Beckers and et.al. Cryogenic characterization of 28 nm bulk cmos technology for quantum computing. In *ESSDERC 2017*, 2017.

[30] Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. *SIAM Journal on computing*, 26(5):1411–1473, 1997.

[31] Gautier Berthou, Tristan Delizy, Kevin Marquet, Tanguy Risset, and Guillaume Salagnac. Peripheral state persistence for transiently-powered systems. In *2017 Global Internet of Things Summit (GIoTS)*, pages 1–6. IEEE, 2017.

[32] Debjyoti Bhattacharjee et al. Contra: area-constrained technology mapping framework for memristive memory processing unit. In *ICCAD*, 2020.

[33] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. The parsec benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 72–81. ACM, 2008.

[34] Lev S Bishop, Sergey Bravyi, Andrew Cross, Jay M Gambetta, and John Smolin. Quantum volume. *Quantum Volume. Technical Report*, 2017.

[35] Robin Blume-Kohout, John King Gamble, Erik Nielsen, Kenneth Rudinger, Jonathan Mizrahi, Kevin Fortier, and Peter Maunz. Demonstration of qubit operations below a rigorous fault tolerance threshold with gate set tomography. *Nature communications*, 8:14485, 2017.

[36] Robin Blume-Kohout and Kevin C Young. A volumetric framework for quantum computer benchmarks. *arXiv preprint arXiv:1904.05546*, 2019.

[37] Robin J Blume-Kohout and Kevin Young. Metrics and benchmarks for quantum processors: State of play. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States); Sandia . . . , 2019.

[38] Sergio Boixo, Sergei V Isakov, Vadim N Smelyanskiy, Ryan Babbush, Nan Ding, Zhang Jiang, Michael J Bremner, John M Martinis, and Hartmut Neven. Characterizing quantum supremacy in near-term devices. *Nature Physics*, 14(6):595–600, 2018.

[39] Kristine Boone, Arnaud Carignan-Dugas, Joel J Wallman, and Joseph Emerson. Randomized benchmarking under different gate sets. *Physical Review A*, 99(3):032329, 2019.

[40] Taoufik Bouguera, Jean-François Diouris, Jean-Jacques Chaillout, Randa Jaouadi, and Guillaume Andrieux. Energy consumption model for sensor nodes based on lora and lorawan. *Sensors*, 18(7):2104, 2018.

[41] Adam Bouland, Bill Fefferman, Chinmay Nirkhe, and Umesh Vazirani. On the complexity and verification of quantum random circuit sampling. *Nature Physics*, 15(2):159–163, 2019.

[42] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[43] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *Annual Cryptology Conference*, pages 868–886. Springer, 2012.

[44] Bravyi et al. Correcting coherent errors with surface codes. *npj Quantum Information*, 2018.

[45] Sergey Bravyi, Matthias Englbrecht, Robert König, and Nolan Peard. Correcting coherent errors with surface codes. *npj Quantum Information*, 4(1):55, 2018.

[46] Teresa Brecht, Wolfgang Pfaff, Chen Wang, Yiwen Chu, Luigi Frunzio, Michel H Devoret, and Robert J Schoelkopf. Multilayer microwave integrated quantum circuits for scalable quantum computing. *npj Quantum Information*, 2:16002, 2016.

[47] Michael J Bremner, Ashley Montanaro, and Dan J Shepherd. Average-case complexity versus approximate simulation of commuting quantum computations. *Physical review letters*, 117(8):080501, 2016.

[48] Joseph W Britton, Brian C Sawyer, Adam C Keith, C-C Joseph Wang, James K Freericks, Hermann Uys, Michael J Biercuk, and John J Bollinger. Engineered two-dimensional ising interactions in a trapped-ion quantum simulator with hundreds of spins. *Nature*, 484(7395):489–492, 2012.

[49] Colin D Bruzewicz, John Chiaverini, Robert McConnell, and Jeremy M Sage. Trapped-ion quantum computing: Progress and challenges. *Applied Physics Reviews*, 6(2):021314, 2019.

[50] Ilkwon Byun and et.al. Cryocore: a fast and dense processor architecture for cryogenic computing. In *ISCA 2020*, 2020.

[51] Yudong Cao, Jonathan Romero, Jonathan P Olson, Matthias Degroote, Peter D Johnson, Mária Kieferová, Ian D Kivlichan, Tim Menke, Borja Peropadre, Nicolas PD Sawaya, et al. Quantum chemistry in the age of quantum computing. *arXiv preprint arXiv:1812.09976*, 2018.

[52] Peter R Cappello and Kenneth Steiglitz. A vlsi layout for a pipelined dadda multiplier. *ACM Transactions on Computer Systems (TOCS)*, 1(2):157–174, 1983.

[53] Arnaud Carignan-Dugas, Joel J Wallman, and Joseph Emerson. Characterizing universal gate sets via dihedral benchmarking. *Physical Review A*, 92(6):060302, 2015.

[54] Christopher Chamberland, Joel Wallman, Stefanie Beale, and Raymond Laflamme. Hard decoding algorithm for optimizing thresholds under general markovian noise. *Physical Review A*, 95(4):042332, 2017.

[55] Anantha P Chandrakasan, Denis C Daly, Joyce Kwong, and Yogesh K Ramadass. Next generation micro-power systems. In *2008 IEEE Symposium on VLSI Circuits*, pages 2–5. IEEE, 2008.

[56] Anantha P Chandrakasan, Denis C Daly, Joyce Kwong, and Yogesh K Ramadass. Next generation micro-power systems. In *2008 IEEE Symposium on VLSI Circuits*, pages 2–5. IEEE, 2008.

[57] Chih-Chung Chang and Chih-Jen Lin. Libsvm: A library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):27, 2011.

[58] Jin Chao, Ahmad Al Badawi, Balagopal Unnikrishnan, Jie Lin, Chan Fook Mun, James M Brown, J Peter Campbell, Michael Chiang, Jayashree Kalpathy-Cramer, Vijay Ramaseshan Chandrasekhar, et al. Carenets: compact and resource-efficient cnn for homomorphic inference on encrypted medical images. *arXiv preprint arXiv:1901.10074*, 2019.

[59] Lerong Chen, Jiawen Li, Yiran Chen, Qiuping Deng, Jiyuan Shen, Xiaoyao Liang, and Li Jiang. Accelerator-friendly neural-network training: Learning variations and defects in rram crossbar. In *Proceedings of the Conference on Design, Automation & Test in Europe*, pages 19–24. European Design and Automation Association, 2017.

[60] Kai-Wen Cheng and Chien-Cheng Tseng. Quantum full adder and subtractor. *Electronics Letters*, 38(22):1343–1344, 2002.

[61] Jerry M Chow, Jay M Gambetta, Antonio D Corcoles, Seth T Merkel, John A Smolin, Chad Rigetti, S Poletto, George A Keefe, Mary B Rothwell, John R Rozen, et al. Universal quantum gate set approaching fault-tolerant thresholds with superconducting qubits. *Physical review letters*, 109(6):060501, 2012.

[62] Zamshed Chowdhury, Jonathan D Harms, S Karen Khatamifard, Masoud Zabihi, Yang Lv, Andrew P Lyle, Sachin S Sapatnekar, Ulya R Karpuzcu, and Jian-Ping Wang. Efficient in-memory processing using spintronics. *IEEE Computer Architecture Letters*, 17(1):42–46, 2017.

[63] Zamshed Chowdhury, S Karen Khatamifard, Salonik Resch, Husrev Cilasun, Zhengyang Zhao, Masoud Zabihi, Meisam Razaviyayn, Jian-Ping Wang, Sachin Sapatnekar, and Ulya R Karpuzcu. Cram-seq: Accelerating rna-seq abundance quantification using computational ram. *IEEE Transactions on Emerging Topics in Computing*, 2022.

[64] Isaac L Chuang and Michael A Nielsen. Prescription for experimental determination of the dynamics of a quantum black box. *Journal of Modern Optics*, 44(11-12):2455–2467, 1997.

[65] Hüsrev Cılasun, Salonik Resch, Zamshed Iqbal Chowdhury, Erin Olson, Masoud Zabihi, Zhengyang Zhao, Thomas Peterson, Jian-Ping Wang, Sachin S Sapatnekar, and Ulya Karpuzcu. Crafft: High resolution fft accelerator in spintronic computational ram. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

[66] Lawrence T Clark et al. Asap7: A 7-nm finfet predictive process design kit. *Microelectronics Journal*, 53, 2016.

[67] Alexei Colin and Brandon Lucia. Chain: tasks and channels for reliable intermittent programs. In *ACM SIGPLAN Notices*, volume 51, pages 514–530. ACM, 2016.

[68] Alexei Colin and Brandon Lucia. Termination checking and task decomposition for task-based intermittent programs. In *Proceedings of the 27th International Conference on Compiler Construction*, pages 116–127. ACM, 2018.

[69] Alexei Colin and Brandon Lucia. Termination checking and task decomposition for task-based intermittent programs. In *Proceedings of the 27th International Conference on Compiler Construction*, pages 116–127, 2018.

[70] Alexei Colin, Emily Ruppel, and Brandon Lucia. A reconfigurable energy storage architecture for energy-harvesting devices. In *ACM SIGPLAN Notices*, volume 53, pages 767–781. ACM, 2018.

[71] Alexei Colin, Emily Ruppel, and Brandon Lucia. A reconfigurable energy storage architecture for energy-harvesting devices. In *ACM SIGPLAN Notices*, volume 53, pages 767–781. ACM, 2018.

[72] Alexei Colin, Emily Ruppel, and Brandon Lucia. A reconfigurable energy storage architecture for energy-harvesting devices. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 767–781, 2018.

[73] Ana Collado and Apostolos Georgiadis. Conformal hybrid solar and electromagnetic (em) energy harvesting rectenna. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 60(8):2225–2234, 2013.

[74] Y Conraux, JP Nozieres, V Da Costa, M Toulemonde, and K Ounadjela. Effects of swift heavy ion bombardment on magnetic tunnel junction functional properties. *Journal of applied physics*, 93(10):7301–7303, 2003.

[75] Francesco Conti, Pasquale Davide Schiavone, and Luca Benini. Xnor neural engine: A hardware accelerator ip for 21.6-fj/op binary neural network inference. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2940–2951, 2018.

[76] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[77] Marcus Cramer, Martin B Plenio, Steven T Flammia, Rolando Somma, David Gross, Stephen D Bartlett, Olivier Landon-Cardinal, David Poulin, and Yi-Kai Liu. Efficient quantum state tomography. *Nature communications*, 1:149, 2010.

[78] Andrew Cross. The ibm q experience and qiskit open-source quantum computing software. In *APS March Meeting Abstracts*, volume 2018, pages L58–003, 2018.

[79] Andrew W Cross, Lev S Bishop, Sarah Sheldon, Paul D Nation, and Jay M Gambetta. Validating quantum computers using randomized model circuits. *Physical Review A*, 100(3):032328, 2019.

[80] Husrev Cılasun, Salonik Resch, Zamshed Iqbal Chowdhury, Erin Olson, Masoud Zabihi, Zhengyang Zhao, Thomas Peterson, Jian-Ping Wang, Sachin S. Sapatnekar, and Ulya Karpuzcu. Crafft: High resolution fft accelerator in spintronic computational ram. In *Proceedings of the 57th Annual ACM/IEEE Design Automation Conference*, 2020.

[81] Mina Danesh and John R Long. Photovoltaic antennas for autonomous wireless systems. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 58(12):807–811, 2011.

[82] Christoph Dankert, Richard Cleve, Joseph Emerson, and Etera Livine. Exact and approximate unitary 2-designs and their application to fidelity estimation. *Physical Review A*, 80(1):012304, 2009.

[83] Koen De Bosschere, Albert Cohen, Jonas Maebe, and Harm Munk. Hipeac vision 2015, 2015.

[84] Jasper de Winkel, Carlo Delle Donne, Kasim Sinan Yildirim, Przemysław Pawełczak, and Josiah Hester. Reliable timekeeping for intermittent computing. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 53–67, 2020.

[85] Bradley Denby and Brandon Lucia. Orbital edge computing: Machine inference in space. *IEEE Computer Architecture Letters*, 18(1):59–62, 2019.

[86] Bradley Denby and Brandon Lucia. Orbital edge computing: Nanosatellite constellations as a new class of computer system. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 939–954, 2020.

[87] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[88] Zhitao Diao et al. Spin transfer switching in dual mgo magnetic tunnel junctions. *Applied Physics Letters*, 90(13), 2007.

[89] Zhitao Diao, Alex Panchula, Yunfei Ding, Mahendra Pakala, Shengyuan Wang, Zhanjie Li, Dmytro Apalkov, Hideyasu Nagai, Alexander Driskill-Smith, Lien-Chang Wang, et al. Spin transfer switching in dual mgo magnetic tunnel junctions. *Applied Physics Letters*, 90(13):132508, 2007.

[90] Xiangyu Dong and et.al. Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(7), 2012.

[91] Xiangyu Dong, Xiaoxia Wu, Guangyu Sun, Yuan Xie, Helen Li, and Yiran Chen. Circuit and microarchitecture evaluation of 3d stacking magnetic ram (mram) as a universal memory replacement. In *Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE*, pages 554–559. IEEE, 2008.

[92] Xiangyu Dong, Cong Xu, Yuan Xie, and Norman P Jouppi. Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(7):994–1007, 2012.

[93] Xinglei Dou and Lei Liu. A new qubits mapping mechanism for multi-programming quantum computing. In *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, pages 349–350, 2020.

[94] Kai-Bo Duan and S Sathiya Keerthi. Which is the best multiclass svm method? an empirical study. In *International workshop on multiple classifier systems*, pages 278–285. Springer, 2005.

[95] Casey Duckering, Jonathan M Baker, David I Schuster, and Frederic T Chong. Virtualized logical qubits: A 2.5 d architecture for error-corrected quantum computing. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 173–185. IEEE, 2020.

[96] Philipp Dürrenfeld et al. Tunable damping, saturation magnetization, and exchange stiffness of half-heusler nimnsb thin films. *Physical Review B*, 92(21), 2015.

[97] Philipp Dürrenfeld, Felicitas Gerhard, Jonathan Chico, Randy K Dumas, Mojtaba Ranjbar, Anders Bergman, Lars Bergqvist, Anna Delin, Charles Gould, Laurens W Molenkamp, et al. Tunable damping, saturation magnetization, and exchange stiffness of half-heusler nimnsb thin films. *Physical Review B*, 92(21):214424, 2015.

[98] Charles Eckert, Xiaowei Wang, Jingcheng Wang, Arun Subramaniyan, Ravi Iyer, Dennis Sylvester, David Blaauw, and Reetuparna Das. Neural cache: Bit-serial in-cache acceleration of deep neural networks. In *Proceedings of the 45th Annual International Symposium on Computer Architecture*, pages 383–396. IEEE Press, 2018.

[99] Joseph Emerson, Robert Alicki, and Karol Życzkowski. Scalable noise estimation with random unitary operators. *Journal of Optics B: Quantum and Semiclassical Optics*, 7(10):S347, 2005.

[100] Alexander Erhard, Joel James Wallman, Lukas Postler, Michael Meth, Roman Stricker, Esteban Adrian Martinez, Philipp Schindler, Thomas Monz, Joseph Emerson, and Rainer Blatt. Characterizing large-scale quantum computers via cycle benchmarking. *arXiv preprint arXiv:1902.08543*, 2019.

[101] Tim J Evans, Robin Harper, and Steven T Flammia. Scalable bayesian hamiltonian learning. *arXiv preprint arXiv:1912.07636*, 2019.

[102] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, 2012:144, 2012.

[103] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028*, 2014.

[104] Samuele Ferracin, Theodoros Kapourniotis, and Animesh Datta. Accrediting outputs of noisy intermediate-scale quantum computing devices. *New Journal of Physics*, 21(11):113038, 2019.

[105] Steven T Flammia and Yi-Kai Liu. Direct fidelity estimation from few pauli measurements. *Physical review letters*, 106(23):230501, 2011.

[106] Patrick R Fleming, Brian D Olson, W Timothy Holman, Bharat L Bhuva, and Lloyd W Massengill. Design technique for mitigation of soft errors in differential switched-capacitor circuits. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 55(9):838–842, 2008.

[107] Austin G Fowler. Coping with qubit leakage in topological codes. *Physical Review A*, 88(4):042308, 2013.

[108] Austin G Fowler, Matteo Mariantoni, John M Martinis, and Andrew N Cleland. Surface codes: Towards practical large-scale quantum computation. *Physical Review A*, 86(3):032324, 2012.

[109] Xiang Fu and et.al. eqasm: An executable quantum instruction set architecture. In *HPCA 2019*, 2019.

[110] Daichi Fujiki and et.al. In-memory data parallel processor. *ACM SIGPLAN Notices*, 53(2):1–14, 2018.

[111] Jay M Gambetta, AD Córcoles, Seth T Merkel, Blake R Johnson, John A Smolin, Jerry M Chow, Colm A Ryan, Chad Rigetti, S Poletto, Thomas A Ohki, et al. Characterization of addressability by simultaneous randomized benchmarking. *Physical review letters*, 109(24):240504, 2012.

[112] Karthik Ganesan, Joshua San Miguel, and Natalie Enright Jerger. The what's next intermittent computing architecture. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 211–223. IEEE, 2019.

[113] Karthik Ganesan, Joshua San Miguel, and Natalie Enright Jerger. The what's next intermittent computing architecture. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 211–223. IEEE, 2019.

[114] Kevin Garello, Farrukh Yasin, S Couet, Laurent Souriau, J Swerts, S Rao, Simon Van Beek, Wonsub Kim, Enlong Liu, S Kundu, et al. Sot-mram 300mm integration for low power and ultrafast embedded memories. In *2018 IEEE Symposium on VLSI Circuits*, pages 81–82. IEEE, 2018.

[115] Michael R Geller and Zhongyuan Zhou. Efficient error models for fault-tolerant architectures and the pauli twirling approximation. *Physical Review A*, 88(1):012314, 2013.

[116] Simone Gerardin and Alessandro Paccagnella. Present and future non-volatile memories for space. *IEEE Transactions on Nuclear Science*, 57(6):3016–3039, 2010.

[117] Joydip Ghosh, Austin G Fowler, and Michael R Geller. Surface code with decoherence: An analysis of three superconducting architectures. *Physical Review A*, 86(6):062318, 2012.

[118] Alexei Gilchrist, Nathan K Langford, and Michael A Nielsen. Distance measures to compare real and ideal quantum processes. *Physical Review A*, 71(6):062310, 2005.

[119] Graham Gobieski, Nathan Beckmann, and Brandon Lucia. Intermittent deep neural network inference, 2018.

[120] Graham Gobieski, Brandon Lucia, and Nathan Beckmann. Intelligence beyond the edge: Inference on intermittent embedded systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 199–213, 2019.

[121] Graham Gobieski, Amolak Nagi, Nathan Serafin, Mehmet Meric Isgenc, Nathan Beckmann, and Brandon Lucia. Manic: A vector-dataflow architecture for ultra-low-power embedded systems. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 670–684, 2019.

[122] Michel X Goemans and David P Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145, 1995.

[123] CHRISTIAN Gogolin, MARTIN Kliesch, Leandro Aolita, and Jens Eisert. Boson-sampling in the light of sample complexity. *arXiv preprint arXiv:1306.3995*, 2013.

[124] Carles Gomez, Joaquim Oller, and Josep Paradells. Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology. *Sensors*, 12(9):11734–11753, 2012.

[125] Daniel Gottesman. Stabilizer codes and quantum error correction. *arXiv preprint quant-ph/9705052*, 1997.

[126] Daniel Gottesman. The heisenberg representation of quantum computers. *arXiv preprint quant-ph/9807006*, 1998.

[127] Daniel Gottesman. An introduction to quantum error correction and fault-tolerant quantum computation. In *Quantum information science and its contributions to mathematics, Proceedings of Symposia in Applied Mathematics*, volume 68, pages 13–58, 2010.

[128] Christopher E Granade, Christopher Ferrie, Nathan Wiebe, and David G Cory. Robust online hamiltonian learning. *New Journal of Physics*, 14(10):103013, 2012.

[129] Martin A Green, Yoshihiro Hishikawa, Ewan D Dunlop, Dean H Levi, Jochen Hohl-Ebinger, and Anita WY Ho-Baillie. Solar cell efficiency tables (version 52). *Progress in Photovoltaics: Research and Applications*, 26(7):427–436, 2018.

[130] Greenbaum et al. Modeling coherent errors in quantum error correction. *Quantum Science and Technology*, 2017.

[131] Daniel Greenbaum. Introduction to quantum gate set tomography. *arXiv preprint arXiv:1509.02921*, 2015.

[132] Daniel Greenbaum and Zachary Dutton. Modeling coherent errors in quantum error correction. *Quantum Science and Technology*, 3(1):015007, 2017.

[133] Hayit Greenspan, Bram Van Ginneken, and Ronald M Summers. Guest editorial deep learning in medical imaging: Overview and future promise of an exciting new technique. *IEEE Transactions on Medical Imaging*, 35(5):1153–1159, 2016.

[134] Hayit Greenspan, Bram Van Ginneken, and Ronald M Summers. Guest editorial deep learning in medical imaging: Overview and future promise of an exciting new technique. *IEEE transactions on medical imaging*, 35(5):1153–1159, 2016.

[135] Robert B Griffiths. *Consistent quantum theory*. Cambridge University Press, 2003.

[136] Alessandro Grossi, Elisa Vianello, Mohamed M Sabry, Marios Barlas, Laurent Grenouillet, Jean Coignus, Edith Beigne, Tony Wu, Binh Q Le, Mary K Wootters, et al. Resistive ram endurance: Array-level characterization and correction techniques targeting deep learning applications. *IEEE Transactions on Electron Devices*, 66(3):1281–1288, 2019.

[137] https://grove-docs.readthedocs.io/en/latest/vqe.html, 2019. Accessed: 20120-03-06.

[138] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219. ACM, 1996.

[139] Gian Giacomo Guerreschi and Anne Y Matsuura. Qaoa for max-cut requires hundreds of qubits for quantum speed-up. *Scientific reports*, 9, 2019.

[140] Saransh Gupta, Mohsen Imani, and Tajana Rosing. Felix: Fast and energy-efficient logic in memory. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–7. IEEE, 2018.

[141] John L Gustafson and Quinn O Snell. Hint: A new way to measure computer performance. In *Proceedings of the Twenty-Eighth Annual Hawaii International Conference on System Sciences*, volume 2, pages 392–401. IEEE, 1995.

[142] Matthew R Guthaus, Jeffrey S Ringenberg, Dan Ernst, Todd M Austin, Trevor Mudge, and Richard B Brown. Mibench: A free, commercially representative embedded benchmark suite. In *Proceedings of the fourth annual IEEE international workshop on workload characterization. WWC-4 (Cat. No. 01EX538)*, pages 3–14. IEEE, 2001.

[143] Mauricio Gutiérrez and Kenneth R Brown. Comparison of a quantum error-correction threshold for exact and approximate errors. *Physical Review A*, 91(2):022335, 2015.

[144] Mauricio Gutiérrez, Conor Smith, Livia Lulushi, Smitha Janardan, and Kenneth R Brown. Errors and pseudothresholds for incoherent and coherent noise. *Physical Review A*, 94(4):042338, 2016.

[145] Mauricio Gutiérrez, Lukas Svec, Alexander Vargo, and Kenneth R Brown. Approximation of realistic errors by clifford channels and pauli measurements. *Physical Review A*, 87(3):030302, 2013.

[146] Christian Hakert, Kuan-Hsun Chen, Paul R Genssler, Georg von der Brüggen, Lars Bauer, Hussam Amrouch, Jian-Jia Chen, and Jörg Henkel. Softwear: Software-only in-memory wear-leveling for non-volatile main memory. *arXiv preprint arXiv:2004.03244*, 2020.

[147] Thomas Häner, Damian S Steiger, Mikhail Smelyanskiy, and Matthias Troyer. High performance emulation of quantum circuits. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 866–874. IEEE, 2016.

[148] Ramesh Harjani and Saurabh Chaubey. A unified framework for capacitive series-parallel dc-dc converter design. In *Proceedings of the IEEE 2014 Custom Integrated Circuits Conference*, pages 1–8. IEEE, 2014.

[149] Philipp Hauke, Fernando M Cucchietti, Luca Tagliacozzo, Ivan Deutsch, and Maciej Lewenstein. Can one trust quantum simulators? *Reports on Progress in Physics*, 75(8):082401, 2012.

[150] Zhezhi He, Yang Zhang, Shaahin Angizi, Boqing Gong, and Deliang Fan. Exploring a sot-mram based in-memory computing for data processing. *IEEE Transactions on Multi-Scale Computing Systems*, 4(4):676–685, 2018.

[151] Jonas Helsen, Xiao Xue, Lieven MK Vandersypen, and Stephanie Wehner. A new class of efficient randomized benchmarking protocols. *npj Quantum Information*, 5(1):1–9, 2019.

[152] John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011.

[153] Josiah Hester, Travis Peters, Tianlong Yun, Ronald Peterson, Joseph Skinner, Bhargav Golla, Kevin Storer, Steven Hearndon, Kevin Freeman, Sarah Lord, et al. Amulet: An energy-efficient, multi-application wearable platform. In *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*, pages 216–229. ACM, 2016.

[154] Josiah Hester, Lanny Sitanayah, and Jacob Sorber. Tragedy of the coulombs: Federating energy storage for tiny, intermittently-powered sensors. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, pages 5–16. ACM, 2015.

[155] Josiah Hester and Jacob Sorber. Flicker: Rapid prototyping for the batteryless internet-of-things. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, page 19. ACM, 2017.

[156] Josiah Hester, Kevin Storer, and Jacob Sorber. Timely execution on intermittently powered batteryless sensors. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, page 17. ACM, 2017.

[157] Matthew Hicks. Clank: Architectural support for intermittent computation. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pages 228–240. IEEE, 2017.

[158] Atsufumi Hirohata et al. Roadmap for emerging materials for spintronic device applications. *IEEE Transactions on Magnetics*, 51(10):1–11, 2015.

[159] Tifenn Hirtzlin, Bogdan Penkovsky, Jacques-Olivier Klein, Nicolas Locatelli, Adrien F Vincent, Marc Bocquet, Jean-Michel Portal, and Damien Querlioz. Implementing binarized neural networks with magnetoresistive ram without error correction. In *2019 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, pages 1–5. IEEE, 2019.

[160] Clare Horsman et al. Surface code quantum computing by lattice surgery. *New Journal of Physics*, 14(12):123011, 2012.

[161] G Hu, JH Lee, JJ Nowak, JZ Sun, J Harms, A Annunziata, S Brown, W Chen, YH Kim, G Lauer, et al. Stt-mram with double magnetic tunnel junctions. In *2015 IEEE International Electron Devices Meeting (IEDM)*, pages 26–3. IEEE, 2015.

[162] Hsin-Yuan Huang, Richard Kueng, and John Preskill. Predicting many properties of a quantum system from very few measurements. *arXiv preprint arXiv:2002.08953*, 2020.

[163] Xiao-Di Huang and et.al. Forming-free, fast, uniform, and high endurance resistive switching from cryogenic to high temperatures in w/alo x/al 2 o 3/pt bilayer memristor. *IEEE Electron Device Letters*, 41(4):549–552, 2020.

[164] Harold Hughes, Konrad Bussmann, Patrick J McMarr, Shu-Fan Cheng, Robert Shull, Andrew P Chen, Simon Schafer, Tim Mewes, Adrian Ong, Eugene Chen, et al. Radiation studies of spin-transfer torque materials and devices. *IEEE Transactions on Nuclear Science*, 59(6):3027–3033, 2012.

[165] Kyuyeon Hwang and Wonyong Sung. Fixed-point feedforward deep neural network design using weights+ 1, 0, and- 1. In *2014 IEEE Workshop on Signal Processing Systems (SiPS)*, pages 1–6. IEEE, 2014.

[166] Paul Jaffe and James McSpadden. Energy conversion and transmission modules for space solar power. *Proceedings of the IEEE*, 101(6):1424–1437, 2013.

[167] Guenole Jan, Luc Thomas, Son Le, Yuan-Jen Lee, Huanlong Liu, Jian Zhu, Ru-Ying Tong, Keyu Pi, Yu-Jen Wang, Dongna Shen, et al. Demonstration of fully functional 8mb perpendicular stt-mram chips with sub-5ns writing for non-volatile embedded memories. In *2014 Symposium on VLSI Technology (VLSI-Technology): Digest of Technical Papers*, pages 1–2. IEEE, 2014.

[168] Ali Javadi-Abhari et al. Optimized surface code communication in superconducting quantum computers. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 692–705, 2017.

[169] Ali JavadiAbhari, Shruti Patil, Daniel Kudrow, Jeff Heckey, Alexey Lvov, Frederic T Chong, and Margaret Martonosi. Scaffcc: a framework for compilation and analysis of quantum computing programs. In *Proceedings of the 11th ACM Conference on Computing Frontiers*, page 1. ACM, 2014.

[170] Hrishikesh Jayakumar, Arnab Raha, and Vijay Raghunathan. Quickrecall: A low overhead hw/sw approach for enabling computations across power cycles in transiently powered computers. In *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*, pages 330–335. IEEE, 2014.

[171] Supreet Jeloka and et.al. A 28 nm configurable memory (tcam/bcam/sram) using push-rule 6t bit cell enabling logic-in-memory. *IEEE Journal of Solid-State Circuits*, 51(4), 2016.

[172] Hongyang Jia, H Valavi, Y Tang, J Zhang, and N Verma. A programmable embedded microprocessor for bit-scalable in-memory computing. In *2019 IEEE Hot Chips 31 Symposium (HCS)*, pages 1–29. IEEE, 2019.

[173] Wanyeong Jung, Sechang Oh, Suyoung Bang, Yoonmyung Lee, Dennis Sylvester, and David Blaauw. 23.3 a 3nw fully integrated energy harvester based on self-oscillating switched-capacitor dc-dc converter. In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 398–399. IEEE, 2014.

[174] Gil Kalai and Guy Kindler. Gaussian noise sensitivity and bosonsampling. *arXiv preprint arXiv:1409.3093*, 2014.

[175] Sandeep Kamath and Joakim Lindh. Measuring bluetooth low energy power consumption. *Texas instruments application note AN092, Dallas*, 2010.

[176] Holger Kappert, Norbert Kordas, Stefan Dreiner, Uwe Paschen, and Rainer Kokozinski. High temperature soi cmos technology and circuit realization for applications up to 300°c. In *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1162–1165. IEEE, 2015.

[177] Nader Khammassi, Imran Ashraf, Xiang Fu, Carmen G Almudever, and Koen Bertels. Qx: A high-performance quantum computer simulation platform. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 464–469. IEEE, 2017.

[178] Kihwan Kim, M-S Chang, Simcha Korenblit, Rajibul Islam, Emily E Edwards, James K Freericks, G-D Lin, L-M Duan, and Christopher Monroe. Quantum simulation of frustrated ising spins with trapped ions. *Nature*, 465(7298):590–593, 2010.

[179] SangBum Kim, Geoffrey W Burr, Wanki Kim, and Sung-Wook Nam. Phase-change memory cycling endurance. *MRS Bulletin*, 44(9):710–714, 2019.

[180] Sangkil Kim, Rushi Vyas, Jo Bito, Kyriaki Niotaki, Ana Collado, Apostolos Georgiadis, and Manos M Tentzeris. Ambient rf energy-harvesting technologies for self-sustainable standalone wireless sensor platforms. *Proceedings of the IEEE*, 102(11):1649–1666, 2014.

[181] Sangkil Kim, Rushi Vyas, Jo Bito, Kyriaki Niotaki, Ana Collado, Apostolos Georgiadis, and Manos M Tentzeris. Ambient rf energy-harvesting technologies for self-sustainable standalone wireless sensor platforms. *Proceedings of the IEEE*, 102(11):1649–1666, 2014.

[182] Shelby Kimmel, Marcus P da Silva, Colm A Ryan, Blake R Johnson, and Thomas Ohki. Robust extraction of tomographic information via randomized benchmarking. *Physical Review X*, 4(1):011050, 2014.

[183] Vadym Kliuchnikov, Dmitri Maslov, and Michele Mosca. Fast and efficient exact synthesis of single qubit unitaries generated by clifford and t gates. *arXiv preprint arXiv:1206.5236*, 2012.

[184] Emanuel Knill, Raymond Laflamme, and Wojciech H Zurek. Resilient quantum computation: error models and thresholds. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 454(1969):365–384, 1998.

[185] Emanuel Knill, Dietrich Leibfried, Rolf Reichle, Joe Britton, R Brad Blakestad, John D Jost, Chris Langer, Roee Ozeri, Signe Seidelin, and David J Wineland. Randomized benchmarking of quantum gates. *Physical Review A*, 77(1):012307, 2008.

[186] Daisuke Kobayashi, Kazuyuki Hirose, Takahiro Makino, Shinobu Onoda, Takeshi Ohshima, Shoji Ikeda, Hideo Sato, Eli Christopher Inocencio Enobio, Tetsuo Endoh, and Hideo Ohno. Soft errors in 10-nm-scale magnetic tunnel junctions exposed to high-energy heavy-ion radiation. *Japanese Journal of Applied Physics*, 56(8):0802B4, 2017.

[187] Daisuke Kobayashi, Yuya Kakehashi, Kazuyuki Hirose, Shinobu Onoda, Takahiro Makino, Takeshi Ohshima, Shoji Ikeda, Michihiko Yamanouchi, Hideo Sato, Eli Christopher Enobio, et al. Influence of heavy ion irradiation on perpendicular-anisotropy cofeb-mgo magnetic tunnel junctions. *IEEE Transactions on Nuclear Science*, 61(4):1710–1716, 2014.

[188] Ron Kohavi. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *Kdd*, volume 96, pages 202–207. Citeseer, 1996.

[189] Vito Kortbeek, Kasim Sinan Yildirim, Abu Bakar, Jacob Sorber, Josiah Hester, and Przemysław Pawełczak. Time-sensitive intermittent computing meets legacy software. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 85–99, 2020.

[190] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[191] Kueng et al. Comparing experiments to the fault-tolerance threshold. *Physical review letters*, 2016.

[192] Richard Kueng, David M Long, Andrew C Doherty, and Steven T Flammia. Comparing experiments to the fault-tolerance threshold. *Physical review letters*, 117(17):170502, 2016.

[193] Adrian Kügel. Improved exact solver for the weighted max-sat problem. *Pos@ sat*, 8:15–27, 2010.

[194] Jeetendra Kumar. Use of pass transistor logic to minimize the impact of soft errors in combinational circuits. In *Workshop Syst. Effects of Logic Soft Errors, 2005*, 2005.

[195] Shahar Kvatinsky, Dmitry Belousov, Slavik Liman, Guy Satat, Nimrod Wald, Eby G Friedman, Avinoam Kolodny, and Uri C Weiser. Magic—memristor-aided logic. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 61(11):895–899, 2014.

[196] Lili Lang and et.al. A low temperature functioning cofeb/mgo-based perpendicular magnetic tunnel junction for cryogenic nonvolatile random access memory. *Applied Physics Letters*, 116(2), 2020.

[197] Benjamin P Lanyon et al. Towards quantum chemistry on a quantum computer. *Nature chemistry*, 2(2):106–111, 2010.

[198] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[199] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[200] Gyu-hyeon Lee and et.al. Cryogenic computer architecture modeling with memory-side case studies. In *ISCA 2019*, 2019.

[201] https://www.oreilly.com/library/view/diy-satellite-platforms/9781449312756/ch01s05.html, 2021. Accessed: 2021-07-14.

[202] Vladimir Leonov. Thermoelectric energy harvesting of human body heat for wearable sensors. *IEEE Sensors Journal*, 13(6):2284–2291, 2013.

[203] Li et al. Towards efficient superconducting quantum processor architecture design. In *ASPLOS*, 2020.

[204] Gushu Li, Yufei Ding, and Yuan Xie. Sanq: A simulation framework for architecting noisy intermediate-scale quantum computing system. *arXiv preprint arXiv:1904.11590*, 2019.

[205] Gushu Li, Yufei Ding, and Yuan Xie. Tackling the qubit mapping problem for nisq-era quantum devices. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1001–1014, 2019.

[206] Huihan Li, Shaocong Wang, Xumeng Zhang, Wei Wang, Rui Yang, Zhong Sun, Wanxiang Feng, Peng Lin, Zhongrui Wang, Linfeng Sun, et al. Memristive crossbar arrays for storage and computing applications. *Advanced Intelligent Systems*, 3(9):2100017, 2021.

[207] Huihan Li, Shaocong Wang, Xumeng Zhang, Wei Wang, Rui Yang, Zhong Sun, Wanxiang Feng, Peng Lin, Zhongrui Wang, Linfeng Sun, et al. Memristive crossbar arrays for storage and computing applications. *Advanced Intelligent Systems*, page 2100017, 2021.

[208] Muyuan Li, Mauricio Gutiérrez, Stanley E David, Alonzo Hernandez, and Kenneth R Brown. Fault tolerance with bare ancillary qubits for a [[7, 1, 3]] code. *Physical Review A*, 96(3):032341, 2017.

[209] Shuangchen Li, Cong Xu, Qiaosha Zou, Jishen Zhao, Yu Lu, and Yuan Xie. Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories. In *Proceedings of the 53rd Annual Design Automation Conference*, page 173. ACM, 2016.

[210] Shuangchen Li, Cong Xu, Qiaosha Zou, Jishen Zhao, Yu Lu, and Yuan Xie. Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories. In *Proceedings of the 53rd Annual Design Automation Conference*, pages 1–6, 2016.

[211] Shuang Liang, Shouyi Yin, Leibo Liu, Wayne Luk, and Shaojun Wei. Fp-bnn: Binarized neural network on fpga. *Neurocomputing*, 275:1072–1086, 2018.

[212] David J Lilja. *Measuring computer performance: a practitioner's guide*. Cambridge university press, 2005.

[213] Norbert M Linke, Dmitri Maslov, Martin Roetteler, Shantanu Debnath, Caroline Figgatt, Kevin A Landsman, Kenneth Wright, and Christopher Monroe. Experimental comparison of two quantum computing architectures. *Proceedings of the National Academy of Sciences*, 114(13):3305–3310, 2017.

[214] Daniel Litinski. A game of surface codes: Large-scale quantum computing with lattice surgery. *Quantum*, 3:128, 2019.

[215] Chendong Liu, Yilin Zhang, and Huanyu Zhou. A comprehensive study of bluetooth low energy. In *Journal of Physics: Conference Series*, volume 2093, page 012021. IOP Publishing, 2021.

[216] Daofu Liu, Tianshi Chen, Shaoli Liu, Jinhong Zhou, Shengyuan Zhou, Olivier Teman, Xiaobing Feng, Xuehai Zhou, and Yunji Chen. Pudiannao: A polyvalent machine learning accelerator. In *ACM SIGARCH Computer Architecture News*, volume 43, pages 369–381. ACM, 2015.

[217] Hongtao Liu et al. Uniformity improvement in 1t1r rram with gate voltage ramp programming. *IEEE Electron Device Letters*, 35(12):1224–1226, 2014.

[218] Qingrui Liu and Changhee Jung. Lightweight hardware support for transparent consistency-aware checkpointing in intermittent energy-harvesting systems. In *2016 5th Non-Volatile Memory Systems and Applications Symposium (NVMSA)*, pages 1–6. IEEE, 2016.

[219] Yongpan Liu, Zewei Li, Hehe Li, Yiqun Wang, Xueqing Li, Kaisheng Ma, Shuangchen Li, Meng-Fan Chang, Sampson John, Yuan Xie, et al. Ambient energy harvesting nonvolatile processors: from circuit to system. In *Proceedings of the 52nd Annual Design Automation Conference*, page 150. ACM, 2015.

[220] Seth Lloyd. Pure state quantum statistical mechanics and black holes. *arXiv preprint arXiv:1307.0378*, 2013.

[221] Michael Loceff. A course in quantum computing (for the community college). *Foothill College*, 2015.

[222] Jeffry Louis et al. Performing memristor-aided logic (magic) using stt-mram. In *ICECS*, 2019.

[223] Jeffry Louis, Barak Hoffer, and Shahar Kvatinsky. Performing memristor-aided logic (magic) using stt-mram. In *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 787–790. IEEE, 2019.

[224] R Lowther, W Morris, D Gifford, D Duff, and R Fuller. Latchup immunity in high temperature bulk cmos devices. *Additional Papers and Presentations*, 2011(HITEN):000215–000220, 2011.

[225] Brandon Lucia, Vignesh Balaji, Alexei Colin, Kiwan Maeng, and Emily Ruppel. Intermittent computing: Challenges and opportunities. In *2nd Summit on Advances in Programming Languages (SNAPL 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[226] Brandon Lucia, Brad Denby, Zachary Manchester, Harsh Desai, Emily Ruppel, and Alexei Colin. Computational nanosatellite constellations: Opportunities and challenges. *GetMobile: Mobile Computing and Communications*, 25(1):16–23, 2021.

[227] Brandon Lucia and Benjamin Ransford. A simpler, safer programming and execution model for intermittent systems. In *ACM SIGPLAN Notices*, volume 50, pages 575–585. ACM, 2015.

[228] Giedrius Lukosevicius, Alberto Rodriguez Arreola, and Alex S Weddell. Using sleep states to maximize the active time of transient computing systems. In *Proceedings of the Fifth ACM International Workshop on Energy Harvesting and Energy-Neutral Sensing Systems*, pages 31–36. ACM, 2017.

[229] Kaisheng Ma, Xueqing Li, Jinyang Li, Yongpan Liu, Yuan Xie, Jack Sampson, Mahmut Taylan Kandemir, and Vijaykrishnan Narayanan. Incidental computing on iot nonvolatile processors. In *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 204–218. IEEE, 2017.

[230] Kaisheng Ma, Xueqing Li, Huichu Liu, Xiao Sheng, Yiqun Wang, Karthik Swaminathan, Yongpan Liu, Yuan Xie, John Sampson, and Vijaykrishnan Narayanan. Dynamic power and energy management for energy harvesting nonvolatile processor systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(4):1–23, 2017.

[231] Kaisheng Ma, Yang Zheng, Shuangchen Li, Karthik Swaminathan, Xueqing Li, Yongpan Liu, Jack Sampson, Yuan Xie, and Vijaykrishnan Narayanan. Architecture exploration for ambient energy harvesting nonvolatile processors. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pages 526–537. IEEE, 2015.

[232] Kiwan Maeng, Alexei Colin, and Brandon Lucia. Alpaca: intermittent execution without checkpoints. *Proceedings of the ACM on Programming Languages*, 1(OOPSLA):96, 2017.

[233] Kiwan Maeng and Brandon Lucia. Adaptive dynamic checkpointing for safe efficient intermittent computing. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 129–144, 2018.

[234] E Magesan, JM Gambetta, and J Emerson. Robust randomized benchmarking of quantum processes. *arXiv preprint arXiv:1009.3639*.

[235] Easwar Magesan, Jay M Gambetta, and Joseph Emerson. Scalable and robust randomized benchmarking of quantum processes. *Physical review letters*, 106(18):180504, 2011.

[236] Easwar Magesan, Jay M Gambetta, and Joseph Emerson. Characterizing quantum gates via randomized benchmarking. *Physical Review A*, 85(4):042311, 2012.

[237] Easwar Magesan, Jay M Gambetta, Blake R Johnson, Colm A Ryan, Jerry M Chow, Seth T Merkel, Marcus P Da Silva, George A Keefe, Mary B Rothwell, Thomas A Ohki, et al. Efficient measurement of quantum gate error by interleaved randomized benchmarking. *Physical review letters*, 109(8):080505, 2012.

[238] Easwar Magesan, Daniel Puzzuoli, Christopher E Granade, and David G Cory. Modeling quantum noise for efficient testing of fault-tolerant circuits. *Physical Review A*, 87(1):012324, 2013.

[239] Gerald Mahan and Brian Sales. Thermoelectric materials: New approaches to an old problem. *Physics Today*, 50(3):42–47, 1997.

[240] Marc A Manheimer. Cryogenic computing complexity program: Phase 1 introduction. *IEEE Transactions on Applied Superconductivity*, 25(3), 2015.

[241] Milos Manic, Kasun Amarasinghe, Juan J Rodriguez-Andina, and Craig Rieger. Intelligent buildings of the future: Cyberaware, deep learning powered, and human interacting. *IEEE Industrial Electronics Magazine*, 10(4):32–49, 2016.

[242] Milos Manic, Kasun Amarasinghe, Juan J Rodriguez-Andina, and Craig Rieger. Intelligent buildings of the future: Cyberaware, deep learning powered, and human interacting. *IEEE Industrial Electronics Magazine*, 10(4):32–49, 2016.

[243] Margaret Martonosi and Martin Roetteler. Next steps in quantum computing: Computer science's role. *arXiv preprint arXiv:1903.10541*, 2019.

[244] David G Mavis and Paul H Eaton. Soft error rate mitigation techniques for modern microcircuits. In *2002 IEEE International Reliability Physics Symposium. Proceedings. 40th Annual (Cat. No. 02CH37320)*, pages 216–225. IEEE, 2002.

[245] Sam McArdle, Suguru Endo, Alan Aspuru-Guzik, Simon Benjamin, and Xiao Yuan. Quantum computational chemistry. *arXiv preprint arXiv:1808.10402*, 2018.

[246] Seth T Merkel, Jay M Gambetta, John A Smolin, Stefano Poletto, Antonio D Córcoles, Blake R Johnson, Colm A Ryan, and Matthias Steffen. Self-consistent quantum process tomography. *Physical Review A*, 87(6):062119, 2013.

[247] Seth T Merkel, Emily J Pritchett, and Bryan H Fong. Randomized benchmarking as convolution: Fourier analysis of gate dependent errors. *arXiv preprint arXiv:1804.05951*, 2018.

[248] Tzvetan S Metodi and Frederic T Chong. Quantum computing for computer architects. *Synthesis Lectures in Computer Architecture*, 1(1):1–154, 2006.

[249] David Meyer, Evgenia Dimitriadou, Kurt Hornik, Andreas Weingessel, and Friedrich Leisch. *e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*, 2019. R package version 1.7-2.

[250] Dongmoon Min and et.al. Cryocache: A fast, large, and cost-effective cache architecture for cryogenic computing. In *ASPLOS 2020*, 2020.

[251] Sadahiko Miura, Koichi Nishioka, Hiroshi Naganuma, TV Anh Nguyen, Hiroaki Honjo, Shoji Ikeda, Toshinari Watanabe, Hirofumi Inoue, Masaaki Niwa, Takaho Tanigawa, et al. Scalability of quad interface p-mtj for 1x nm stt-mram with 10-ns low power write operation, 10 years retention and endurance¿ 10^{11}. *IEEE Transactions on Electron Devices*, 67(12):5368–5373, 2020.

[252] S Mizukami, D Watanabe, M Oogane, Y Ando, Y Miura, M Shirai, and T Miyazaki. Low damping constant for co 2 feal heusler alloy films and its correlation with density of states. *Journal of Applied Physics*, 105(7):07D306, 2009.

[253] Javaneh Mohseni et al. Performance modeling and optimization for on-chip interconnects in cross-bar reram memory arrays. In *EPEPS*, 2016.

[254] Ankit Mondal et al. In situ stochastic training of mtj crossbars with machine learning algorithms. *ACM JETC*, 15(2), 2019.

[255] Eric Arturo Montoya, Jen-Ru Chen, Randy Ngelale, Han Kyu Lee, Hsin-Wei Tseng, Lei Wan, En Yang, Patrick Braganca, Ozdal Boyraz, Nader Bagherzadeh, et al. Immunity of nanoscale magnetic tunnel junctions with perpendicular magnetic anisotropy to ionizing radiation. *Scientific reports*, 10(1):1–8, 2020.

[256] Murali et al. Software mitigation of crosstalk on noisy intermediate-scale quantum computers. In *ASPLOS*, 2020.

[257] Onur Mutlu. Intelligent architectures for intelligent machines. In *VLSI-DAT 2020*, pages 1–4. IEEE, 2020.

[258] National Academies of Sciences, Engineering, and Medicine and others. *Quantum computing: progress and prospects*. National Academies Press, 2019.

[259] Hamid Nejatollahi, Saransh Gupta, Mohsen Imani, Tajana Simunic Rosing, Rosario Cammarota, and Nikil Dutt. Cryptopim: in-memory acceleration for lattice-based cryptographic hardware. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

[260] Hamid Nejatollahi, Sina Shahhosseini, Rosario Cammarota, and Nikil Dutt. Exploring energy efficient quantum-resistant signal processing using array processors. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1539–1543. IEEE, 2020.

[261] Leibin Ni et al. An energy-efficient digital reram-crossbar-based cnn with bitwise parallelism. *IEEE Journal on Exploratory solid-state computational devices and circuits*, 3, 2017.

[262] Naomi H Nickerson and Benjamin J Brown. Analysing correlated noise on the surface code using adaptive decoding algorithms. *Quantum*, 3:131, 2019.

[263] Michael Nicolaidis. Time redundancy based soft-error tolerance to rescue nanometer technologies. In *Proceedings 17th IEEE VLSI Test Symposium (Cat. No. PR00146)*, pages 86–94. IEEE, 1999.

[264] Erik Nielsen, John King Gamble, Kenneth Rudinger, Travis Scholten, Kevin Young, and Robin Blume-Kohout. Gate set tomography. *arXiv preprint arXiv:2009.07301*, 2020.

[265] Erik Nielsen, Kenneth Rudinger, Timothy Proctor, Antonio Russo, Kevin Young, and Robin Blume-Kohout. Probing quantum processor performance with pygsti. *arXiv preprint arXiv:2002.12476*, 2020.

[266] Michael A Nielsen and Isaac Chuang. Quantum computation and quantum information, 2002.

[267] William S Noble. What is a support vector machine? *Nature biotechnology*, 24(12):1565–1567, 2006.

[268] Hiroki Noguchi, Kazutaka Ikegami, Keiichi Kushida, Keiko Abe, Shogo Itai, Satoshi Takaya, Naoharu Shimomura, Junichi Ito, Atsushi Kawasumi, Hiroyuki Hara, et al. 7.5 a 3.3 ns-access-time 71.2 μw/mhz 1mb embedded stt-mram using physically eliminated read-disturb scheme and normally-off memory architecture. In *2015 IEEE International Solid-State Circuits Conference-(ISSCC) Digest of Technical Papers*, pages 1–3. IEEE, 2015.

[269] Fabian Oboril, Rajendra Bishnoi, Mojtaba Ebrahimi, and Mehdi B Tahoori. Evaluation of hybrid memory technologies using sot-mram for on-chip cache hierarchy. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(3):367–380, 2015.

[270] Roystein Oliveira, Aditya Jagirdar, and Tapan J Chakraborty. A tmr scheme for seu mitigation in scan flip-flops. In *8th International Symposium on Quality Electronic Design (ISQED'07)*, pages 905–910. IEEE, 2007.

[271] Jonathan Olson, Yudong Cao, Jonathan Romero, Peter Johnson, Pierre-Luc Dallaire-Demers, Nicolas Sawaya, Prineha Narang, Ian Kivlichan, Michael Wasielewski, and Alán Aspuru-Guzik. Quantum information and computation for chemistry. *arXiv preprint arXiv:1706.05413*, 2017.

[272] Mark Oskin, Frederic T Chong, and Isaac L Chuang. A practical architecture for reliable quantum computers. *Computer*, (1):79–87, 2002.

[273] Özgün Özerk, Can Elgezen, Ahmet Can Mert, Erdinç Öztürk, and Erkay Savaş. Efficient number theoretic transform implementation on gpu for homomorphic encryption. *The Journal of Supercomputing*, pages 1–33, 2021.

[274] Matteo Paris and Jaroslav Rehacek. *Quantum state estimation*, volume 649. Springer Science & Business Media, 2004.

[275] Saerom Park, Junyoung Byun, Joohee Lee, Jung Hee Cheon, and Jaewook Lee. He-friendly algorithm for privacy-preserving svm training. *IEEE Access*, 8:57414–57425, 2020.

[276] Bishnu Patra and et.al. Cryo-cmos circuits and systems for quantum computing applications. *IEEE Journal of Solid-State Circuits*, 53(1), 2017.

[277] https://pennylane.ai/qml/demos/tutorial$_{qaoa_m}axcut.html, 2019. Accessed : 2020-9-26.$

[278] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J Love, Alán Aspuru-Guzik, and Jeremy L O'brien. A variational eigenvalue solver on a photonic quantum processor. *Nature communications*, 5:4213, 2014.

[279] Manuel Piñuela, Paul D Mitcheson, and Stepan Lucyszyn. Ambient rf energy harvesting in urban and semi-urban environments. *IEEE Transactions on microwave theory and techniques*, 61(7):2715–2726, 2013.

[280] Yuriy Polyakov, Kurt Rohloff, and Gerard W Ryan. Palisade lattice cryptography library user manual. *Cybersecurity Research Center, New Jersey Institute ofTechnology (NJIT), Tech. Rep*, 15, 2017.

[281] Sandu Popescu, Anthony J Short, and Andreas Winter. Entanglement and the foundations of statistical mechanics. *Nature Physics*, 2(11):754–758, 2006.

[282] JF Poyatos, J Ignacio Cirac, and Peter Zoller. Complete characterization of a quantum process: the two-bit quantum gate. *Physical Review Letters*, 78(2):390, 1997.

[283] JS Pratt and JH Eberly. Qubit cross talk and entanglement decay. *Physical Review B*, 64(19):195314, 2001.

[284] John Preskill. Fault-tolerant quantum computation. In *Introduction to quantum computation and information*, pages 213–269. World Scientific, 1998.

[285] Timothy Proctor, Kenneth Rudinger, Kevin Young, Mohan Sarovar, and Robin Blume-Kohout. What randomized benchmarking actually measures. *Physical review letters*, 119(13):130502, 2017.

[286] https://qiskit.org/textbook/ch-quantum-hardware/randomized-benchmarking.html, 2019. Accessed: 2020-9-29.

[287] Keni Qiu, Nicholas Jao, Mengying Zhao, Cyan Subhra Mishra, Gulsum Gudukbay, Sethu Jose, Jack Sampson, Mahmut Taylan Kandemir, and Vijaykrishnan Narayanan. Resirca: A resilient energy harvesting reram crossbar-based accelerator for intelligent embedded processors. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 315–327. IEEE, 2020.

[288] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2016.

[289] Yogesh K Ramadass and Anantha P Chandrakasan. Voltage scalable switched capacitor dc-dc converter for ultra-low-power on-chip applications. In *2007 IEEE Power Electronics Specialists Conference*, pages 2353–2359. IEEE, 2007.

[290] Benjamin Ransford, Jacob Sorber, and Kevin Fu. Mementos: system support for long-running computation on rfid-scale devices. In *ACM SIGARCH Computer Architecture News*, volume 39, pages 159–170. ACM, 2011.

[291] S Ravi, Govind Shaji Nair, Rajeev Narayan, and Harish M Kittur. Low power and efficient dadda multiplier. *Research Journal of Applied Sciences, Engineering and Technology*, 9(1):53–57, 2015.

[292] Brandon Reagen, Woo-Seok Choi, Yeongil Ko, Vincent T Lee, Hsien-Hsin S Lee, Gu-Yeon Wei, and David Brooks. Cheetah: Optimizing and accelerating homomorphic encryption for private inference. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 26–39. IEEE, 2021.

[293] Fanghui Ren, Albrecht Jander, Pallavi Dhagat, and Cathy Nordman. Radiation tolerance of magnetic tunnel junctions with mgo tunnel barriers. *IEEE Transactions on Nuclear Science*, 59(6):3034–3038, 2012.

[294] Salonik Resch. Quantum noise profiling. https://github.umn.edu/resc0059/QuantumBenchmarkingNoise.

[295] Salonik Resch. *QuantumOps: Performs Common Linear Algebra Operations Used in Quantum Computing and Implements Quantum Algorithms*, 2020. R package version 3.0.1.

[296] Salonik Resch, Husrev Cilasun, and Ulya Karpuzcu. Cryogenic pim: Challenges & opportunities. *IEEE Computer Architecture Letters*, 2021.

[297] Salonik Resch et al. A day in the life of a quantum error. *IEEE Computer Architecture Letters*, 2020.

[298] Salonik Resch and et.al. Mouse: Inference in non-volatile memory for energy harvesting applications. In *MICRO 2020*, 2020.

[299] Salonik Resch and Ulya R Karpuzcu. Benchmarking quantum computers and the impact of quantum noise. *ACM Computing Surveys (CSUR)*, 54(7):1–35, 2021.

[300] Salonik Resch, S Karen Khatamifard, Zamshed I Chowdhury, Masoud Zabihi, Zhengyang Zhao, Husrev Cilasun, Jian-Ping Wang, Sachin S Sapatnekar, and Ulya R Karpuzcu. Energy efficient and reliable inference in nonvolatile memory under extreme operating conditions. *ACM Transactions on Embedded Computing Systems (TECS)*, 2021.

[301] Salonik Resch, S Karen Khatamifard, Zamshed Iqbal Chowdhury, Masoud Zabihi, Zhengyang Zhao, Jian-Ping Wang, Sachin S Sapatnekar, and Ulya R Karpuzcu. Pimball: Binary neural networks in spintronic memory. *ACM Transactions on Architecture and Code Optimization (TACO)*, 16(4):41, 2019.

[302] Martin Roetteler, Krysta M Svore, Dave Wecker, and Nathan Wiebe. Design automation for quantum architectures. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 1312–1317. IEEE, 2017.

[303] James Rosenthal and Matthew S Reynolds. A 158 pj/bit 1.0 mbps bluetooth low energy (ble) compatible backscatter communication system for wireless sensing. In *2019 IEEE Topical Conference on Wireless Sensors and Sensor Networks (WiSNet)*, pages 1–3. IEEE, 2019.

[304] Martin Rötteler. Quantum algorithms for highly non-linear boolean functions. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete algorithms*, pages 448–457. Society for Industrial and Applied Mathematics, 2010.

[305] Emily Ruppel and Brandon Lucia. Transactional concurrency control for intermittent, energy-harvesting computing systems. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 1085–1100. ACM, 2019.

[306] Emily Ruppel and Brandon Lucia. Transactional concurrency control for intermittent, energy-harvesting computing systems. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 1085–1100, 2019.

[307] Daisuke Saida, Saori Kashiwada, Megumi Yakabe, Tadaomi Daibou, Naoki Hase, Miyoshi Fukumoto, Shinji Miwa, Yoshishige Suzuki, Hiroki Noguchi, Shinobu Fujita, et al. Sub-3 ns pulse with sub-100 μa switching of 1x–2x nm perpendicular mtj for high-performance embedded stt-mram towards sub-20 nm cmos. In *2016 IEEE Symposium on VLSI Technology*, pages 1–2. IEEE, 2016.

[308] Daisuke Saida, Saori Kashiwada, Megumi Yakabe, Tadaomi Daibou, Naoki Hase, Miyoshi Fukumoto, Shinji Miwa, Yoshishige Suzuki, Hiroki Noguchi, Shinobu Fujita, et al. Sub-3 ns pulse with sub-100 μa switching of 1x–2x nm perpendicular mtj for high-performance embedded stt-mram towards sub-20 nm cmos. In *2016 IEEE Symposium on VLSI Technology*, pages 1–2. IEEE, 2016.

[309] Daisuke Saida, Saori Kashiwada, Megumi Yakabe, Tadaomi Daibou, Naoki Hase, Miyoshi Fukumoto, Shinji Miwa, Yoshishige Suzuki, Hiroki Noguchi, Shinobu Fujita, et al. Sub-3 ns pulse with sub-100 μa switching of 1x–2x nm perpendicular mtj for high-performance embedded stt-mram towards sub-20 nm cmos. In *2016 IEEE Symposium on VLSI Technology*, pages 1–2. IEEE, 2016.

[310] Abdullah Ash Saki, Mahabubul Alam, and Swaroop Ghosh. Study of decoherence in quantum computers: A circuit-design perspective. *arXiv preprint arXiv:1904.04323*, 2019.

[311] Alanson P Sample, Daniel J Yeager, Pauline S Powledge, Alexander V Mamishev, and Joshua R Smith. Design of an rfid-based battery-free programmable sensing platform. *IEEE transactions on instrumentation and measurement*, 57(11):2608–2615, 2008.

[312] Joshua San Miguel, Karthik Ganesan, Mario Badr, Chunqiu Xia, Rose Li, Hsuan Hsiao, and Natalie Enright Jerger. The eh model: Early design space exploration of intermittent processor architectures. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 600–612. IEEE, 2018.

[313] Mohan Sarovar, Timothy Proctor, Kenneth Rudinger, Kevin Young, Erik Nielsen, and Robin Blume-Kohout. Detecting crosstalk errors in quantum information processors. *arXiv preprint arXiv:1908.09855*, 2019.

[314] Pradeep Kiran Sarvepalli, Andreas Klappenecker, and Martin Rötteler. Asymmetric quantum codes: constructions, bounds and performance. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 465(2105):1645–1672, 2009.

[315] H Sato, ECI Enobio, M Yamanouchi, S Ikeda, S Fukami, S Kanai, F Matsukura, and H Ohno. Properties of magnetic tunnel junctions with a mgo/cofeb/ta/cofeb/mgo recording structure down to junction diameter of 11 nm. *Applied Physics Letters*, 105(6):062403, 2014.

[316] Selahattin Sayil. *Soft error mechanisms, modeling and mitigation*. Springer, 2016.

[317] Selahattin Sayil. A survey of circuit-level soft error mitigation methodologies. *Analog Integrated Circuits and Signal Processing*, 99(1):63–70, 2019.

[318] Selahattin Sayil, Archit H Shah, Md Adnan Zaman, and Mohammad A Islam. Soft error mitigation using transmission gate with varying gate and body bias. *IEEE Design & Test*, 34(1):47–56, 2015.

[319] Ronald D Schrimpf and Daniel M Fleetwood. *Radiation effects and soft errors in integrated circuits and electronic devices*, volume 34. World Scientific, 2004.

[320] Maria Schuld, Alex Bocharov, Krysta Svore, and Nathan Wiebe. Circuit-centric quantum classifiers. *arXiv preprint arXiv:1804.00633*, 2018.

[321] Microsoft SEAL (release 3.7). https://github.com/Microsoft/SEAL, September 2021. Microsoft Research, Redmond, WA.

[322] Peter Selinger. Newsynth: exact and approximate synthesis of quantum circuits. http://www.mathstat.dal.ca/ selinger/newsynth/.

[323] Peter Selinger. Efficient clifford+ t approximation of single-qubit operators. *arXiv preprint arXiv:1212.6253*, 2012.

[324] Mingoo Seok, Scott Hanson, Yu-Shiang Lin, Zhiyoong Foo, Daeyeon Kim, Yoonmyung Lee, Nurrachman Liu, Dennis Sylvester, and David Blaauw. The phoenix processor: A 30pw platform for sensor applications. In *2008 IEEE Symposium on VLSI Circuits*, pages 188–189. IEEE, 2008.

[325] Vivek Seshadri and et.al. Ambit: In-memory accelerator for bulk bitwise operations using commodity dram technology. In *MICRO 2017*, 2017.

[326] Vivek Seshadri, Donghyuk Lee, Thomas Mullins, Hasan Hassan, Amirali Boroumand, Jeremie Kim, Michael A Kozuch, Onur Mutlu, Phillip B Gibbons, and Todd C Mowry. Ambit: In-memory accelerator for bulk bitwise operations using commodity dram technology. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 273–287. ACM, 2017.

[327] A Shabani, RL Kosut, M Mohseni, H Rabitz, MA Broome, MP Almeida, A Fedrizzi, and AG White. Efficient measurement of quantum dynamics via compressive sensing. *Physical review letters*, 106(10):100401, 2011.

[328] Claude Elwood Shannon. A mathematical theory of communication. *ACM SIGMOBILE mobile computing and communications review*, 5(1):3–55, 2001.

[329] M Shin and et.al. Low temperature characterization of 14nm fdsoi cmos devices. In *WOLTE 2014*, 2014.

[330] Yohei Shiokawa, Eiji Komura, Yugo Ishitani, Atsushi Tsumita, Keita Suda, Yuji Kakinuma, and Tomoyuki Sasaki. High write endurance up to 1012 cycles in a spin current-type magnetic memory array. *AIP Advances*, 9(3):035236, 2019.

[331] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[332] Matti Siekkinen, Markus Hiienkari, Jukka K Nurminen, and Johanna Nieminen. How low energy is bluetooth low energy? comparative measurements with zigbee/802.15. 4. In *2012 IEEE wireless communications and networking conference workshops (WCNCW)*, pages 232–237. IEEE, 2012.

[333] Marcus Silva, Easwar Magesan, David W Kribs, and Joseph Emerson. Scalable protocol for identification of correctable codes. *Physical Review A*, 78(1):012347, 2008.

[334] Jonathan Simon, Waseem S Bakr, Ruichao Ma, M Eric Tai, Philipp M Preiss, and Markus Greiner. Quantum simulation of antiferromagnetic spin chains in an optical lattice. *Nature*, 472(7343):307–312, 2011.

[335] James C Spall et al. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE transactions on automatic control*, 37(3):332–341, 1992.

[336] Cloyce D Spradling. Spec cpu2006 benchmark tools. *ACM SIGARCH Computer Architecture News*, 35(1):130–134, 2007.

[337] AM Steane. Introduction to quantum error correction. *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 356(1743):1739–1758, 1998.

[338] Damian S Steiger, Thomas Häner, and Matthias Troyer. Projectq: an open source software framework for quantum computing. *Quantum*, 2(49):10–22331, 2018.

[339] Russell Stutz. Trapped ion quantum computing at honeywell. *Bulletin of the American Physical Society*, 65, 2020.

[340] Fang Su, Wei-Hao Chen, Lixue Xia, Chieh-Pu Lo, Tianqi Tang, Zhibo Wang, Kuo-Hsiang Hsu, Ming Cheng, Jun-Yi Li, Yuan Xie, et al. A 462gops/j rram-based nonvolatile intelligent processor for energy harvesting ioe system featuring nonvolatile logics and processing-in-memory. In *2017 Symposium on VLSI Technology*, pages T260–T261. IEEE, 2017.

[341] Ashish Kumar Sultania and Jeroen Famaey. Energy-aware battery-less bluetooth low energy device prototype powered by ambient light. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, pages 584–585, 2021.

[342] Xiaoyu Sun, Xiaochen Peng, Pai-Yu Chen, Rui Liu, Jae-sun Seo, and Shimeng Yu. Fully parallel rram synaptic array for implementing binary neural network with (+ 1,- 1) weights and (+ 1, 0) neurons. In *Proceedings of the 23rd Asia and South Pacific Design Automation Conference*, pages 574–579. IEEE Press, 2018.

[343] Kevin J Sung, Jiahao Yao, Matthew P Harrigan, Nicholas C Rubin, Zhang Jiang, Lin Lin, Ryan Babbush, and Jarrod R McClean. Using models to improve optimizers for variational quantum algorithms. *Quantum Science and Technology*, 5(4):044008, 2020.

[344] Milijana Surbatovich, Limin Jia, and Brandon Lucia. Automatically enforcing fresh and consistent inputs in intermittent systems. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pages 851–866, 2021.

[345] Krysta M Svore, Alan Geller, Matthias Troyer, John Azariah, Christopher Granade, Bettina Heim, Vadym Kliuchnikov, Mariia Mykhailova, Andres Paz, and Martin Roetteler. Q#: Enabling scalable quantum computing and development with a high-level domain-specific language. *arXiv preprint arXiv:1803.00652*, 2018.

[346] Krysta M Svore and Matthias Troyer. The quantum future of computation. *Computer*, 49(9):21–30, 2016.

[347] Zainab Swaidan, Rouwaida Kanj, Johnny El Hajj, Edward Saad, and Fadi Kurdahi. Rram endurance and retention: Challenges, opportunities and implications on reliable design. In *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 402–405. IEEE, 2019.

[348] Nishil Talati et al. Logic design within memristive memories using memristor-aided logic (magic). *IEEE Transactions on Nanotechnology*, 15(4), 2016.

[349] Tianqi Tang, Lixue Xia, Boxun Li, Yu Wang, and Huazhong Yang. Binary convolutional neural network on rram. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 782–787. IEEE, 2017.

[350] Tianqi Tang, Lixue Xia, Boxun Li, Yu Wang, and Huazhong Yang. Binary convolutional neural network on rram. In *Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific*, pages 782–787. IEEE, 2017.

[351] Tianqi Tang, Lixue Xia, Boxun Li, Yu Wang, and Huazhong Yang. Binary convolutional neural network on rram. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 782–787. IEEE, 2017.

[352] Swamit S Tannu et al. Mitigating measurement errors in quantum computers by exploiting state-dependent bias. In *MICRO*, 2019.

[353] Swamit S Tannu and et.al. Cryogenic-dram based memory system for scalable quantum computers: a feasibility study. In *Proceedings of the International Symposium on Memory Systems*, 2017.

[354] Swamit S Tannu, Zachary A Myers, Prashant J Nair, Douglas M Carmean, and Moinuddin K Qureshi. Taming the instruction bandwidth of quantum computers via hardware-managed error correction. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 679–691, 2017.

[355] Swamit S Tannu and Moinuddin K Qureshi. Not all qubits are created equal: a case for variability-aware policies for nisq-era quantum computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 987–999, 2019.

[356] Phrangboklang Lyngton Thangkhiew et al. Efficient mapping of boolean functions to memristor crossbar using magic nor gates. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 65(8), 2018.

[357] Yu Tomita and Krysta M Svore. Low-distance surface codes under realistic quantum noise. *Physical Review A*, 90(6):062320, 2014.

[358] Whitney J Townsend, Earl E Swartzlander Jr, and Jacob A Abraham. A comparison of dadda and wallace multiplier delays. In *Advanced signal processing algorithms, architectures, and implementations XIII*, volume 5205, pages 552–560. International Society for Optics and Photonics, 2003.

[359] Furkan Turan, Sujoy Sinha Roy, and Ingrid Verbauwhede. Heaws: An accelerator for homomorphic encryption on the amazon aws fpga. *IEEE Transactions on Computers*, 69(8):1185–1196, 2020.

[360] https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones, 2019. Accessed: 2019-06-02.

[361] Cozmin Ududec, Nathan Wiebe, and Joseph Emerson. Information-theoretic equilibration: the appearance of irreversibility under complex quantum dynamics. *Physical review letters*, 111(8):080403, 2013.

[362] Yaman Umuroglu, Nicholas J Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. Finn: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 65–74. ACM, 2017.

[363] Bremner M.J. & Ralph T.C. und, A.P. Quantum sampling problems, bosonsampling and quantum supremacy. *npj Quantum Inf*, 3(15), 2017.

[364] Navnidhi K Upadhyay, Thomas Blum, et al. Engineering tunneling selector to achieve high non-linearity for 1s1r integration. *Frontiers in Nanotechnology*, 3:28, 2021.

[365] Cesar Vaca and et.al. Study from cryogenic to high temperatures of the high-and low-resistance-state currents of reram ni–hfo 2–si capacitors. *IEEE Transactions on Electron Devices*, 63(5):1877–1883, 2016.

[366] Hossein Valavi, Peter J Ramadge, Eric Nestler, and Naveen Verma. A 64-tile 2.4-mb in-memory-computing cnn accelerator employing charge-domain compute. *IEEE Journal of Solid-State Circuits*, 54(6):1789–1799, 2019.

[367] Wim Van Dam, Sean Hallgren, and Lawrence Ip. Quantum algorithms for some hidden shift problems. *SIAM Journal on Computing*, 36(3):763–778, 2006.

[368] McKenzie van der Hagen and Brandon Lucia. Client-optimized algorithms and acceleration for encrypted compute offloading. 2022.

[369] Joel Van Der Woude and Matthew Hicks. Intermittent computation without hardware support or programmer intervention. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 17–32, 2016.

[370] Deepak Vasisht, Zerina Kapetanovic, Jongho Won, Xinxin Jin, Ranveer Chandra, Sudipta Sinha, Ashish Kapoor, Madhusudhan Sudarshan, and Sean Stratman. Farmbeats: An iot platform for data-driven agriculture. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, pages 515–529, 2017.

[371] K Vogel and H Risken. Determination of quasiprobability distributions in terms of probability distributions for the rotated quadrature phase. *Physical Review A*, 40(5):2847, 1989.

[372] Joel Wallman and Joseph Emerson.

[373] Joel Wallman, Chris Granade, Robin Harper, and Steven T Flammia. Estimating the coherence of noise. *New Journal of Physics*, 17(11):113020, 2015.

[374] Joel J Wallman. Bounding experimental quantum error rates relative to fault-tolerant thresholds. *arXiv preprint arXiv:1511.00727*, 2015.

[375] Joel J Wallman. Randomized benchmarking with gate-dependent noise. *Quantum*, 2:47, 2018.

[376] Joel J Wallman and Joseph Emerson. Noise tailoring for scalable quantum computation via randomized compiling. *Physical Review A*, 94(5):052325, 2016.

[377] Fiona Wang and et.al. Dram retention at cryogenic temperatures. In *IMW 2018*, 2018.

[378] Jian-Ping Wang, Mahdi Jamaliz, Angeline Klemm Smith, and Zhengyang Zhao. Magnetic tunnel junction based integrated logics and computational circuits. *Nanomagnetic and Spintronic Devices for Energy-Efficient Memory and Computing*, page 133, 2016.

[379] Jingcheng Wang, Xiaowei Wang, Charles Eckert, Arun Subramaniyan, Reetuparna Das, David Blaauw, and Dennis Sylvester. A 28-nm compute sram with bit-serial logic/arithmetic operations for programmable in-memory vector computing. *IEEE Journal of Solid-State Circuits*, 55(1):76–86, 2019.

[380] Fred Ware and et.al. Do superconducting processors really need cryogenic memories? the case for cold dram. In *Proceedings of the International Symposium on Memory Systems*, 2017.

[381] Dave Wecker, Matthew B Hastings, and Matthias Troyer. Progress towards practical quantum variational algorithms. *Physical Review A*, 92(4):042303, 2015.

[382] Nathan Wiebe, Christopher Granade, and David G Cory. Quantum bootstrapping via compressed quantum hamiltonian learning. *New Journal of Physics*, 17(2):022005, 2015.

[383] Nathan Wiebe, Christopher Granade, Christopher Ferrie, and David Cory. Quantum hamiltonian learning using imperfect quantum resources. *Physical Review A*, 89(4):042314, 2014.

[384] Nathan Wiebe, Christopher Granade, Christopher Ferrie, and David G Cory. Hamiltonian learning and certification using quantum resources. *Physical review letters*, 112(19):190501, 2014.

[385] Harrison Williams, Xun Jian, and Matthew Hicks. Forget failure: Exploiting sram data remanence for low-overhead intermittent computation. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 69–84, 2020.

[386] Ellis Wilson et al. Just-in-time quantum circuit transpilation reduces noise. In *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 345–355. IEEE, 2020.

[387] H-S Philip Wong, Simone Raoux, SangBum Kim, Jiale Liang, John P Reifenberg, Bipin Rajendran, Mehdi Asheghi, and Kenneth E Goodson. Phase change memory. *Proceedings of the IEEE*, 98(12):2201–2227, 2010.

[388] Christopher J Wood and Jay M Gambetta. Quantification and characterization of leakage errors. *Physical Review A*, 97(3):032306, 2018.

[389] K Wright, KM Beck, S Debnath, JM Amini, Y Nam, N Grzesiak, J-S Chen, NC Pisenti, M Chmielewski, C Collins, et al. Benchmarking an 11-qubit quantum computer. *arXiv preprint arXiv:1903.08181*, 2019.

[390] Lixue Xia, Tianqi Tang, Wenqin Huangfu, Ming Cheng, Xiling Yin, Boxun Li, Yu Wang, and Huazhong Yang. Switched by input: power efficient structure for rram-based convolutional neural network. In *Proceedings of the 53rd Annual Design Automation Conference*, page 125. ACM, 2016.

[391] Kodai Yamada, Haruki Maruoka, Jun Furuta, and Kazutoshi Kobayashi. Sensitivity to soft errors of nmos and pmos transistors evaluated by latches with stacking structures in a 65 nm fdsoi process. In *2018 IEEE International Reliability Physics Symposium (IRPS)*, pages P–SE. IEEE, 2018.

[392] Jeng-Bang Yau and et.al. Hybrid cryogenic memory cells for superconducting computing applications. In *ICRC 2017*, 2017.

[393] MUSTAFA BERKE Yelten. Cryogenic dc characteristics of low threshold voltage (vth) n-channel mosfets. *Balkan Journal of Electrical and Computer Engineering*, 7(3).

[394] Faruk Yildiz. Potential ambient energy-harvesting sources and techniques. *Journal of technology Studies*, 35(1):40–48, 2009.

[395] Jun Yoneda, Kenta Takeda, Tomohiro Otsuka, Takashi Nakajima, Matthieu R Delbecq, Giles Allison, Takumu Honda, Tetsuo Kodera, Shunri Oda, Yusuke Hoshi, et al. A quantum-dot spin qubit with coherence limited by charge noise and fidelity higher than 99.9%. *Nature nanotechnology*, 13(2):102, 2018.

[396] Shimeng Yu, Zhiwei Li, Pai-Yu Chen, Huaqiang Wu, Bin Gao, Deli Wang, Wei Wu, and He Qian. Binary neural network with 16 mb rram macro chip for classification and online training. In *Electron Devices Meeting (IEDM), 2016 IEEE International*, pages 16–2. IEEE, 2016.

[397] L Yuan and et.al. Temperature dependence of magnetoresistance in magnetic tunnel junctions with different free layer structures. *Physical Review B*, 73(13), 2006.

[398] Masoud Zabihi, Zamshed Iqbal Chowdhury, Zhengyang Zhao, Ulya R Karpuzcu, Jian-Ping Wang, and Sachin S Sapatnekar. In-memory processing on the spintronic cram: From hardware design to application mapping. *IEEE Transactions on Computers*, 68(8):1159–1173, 2018.

[399] Masoud Zabihi and et.al. In-memory processing on the spintronic cram: From hardware design to application mapping. *IEEE Transactions on Computers*, 68(8), 2018.

[400] Masoud Zabihi, Arvind K Sharma, Meghna G Mankalale, Zamshed Iqbal Chowdhury, Zhengyang Zhao, Salonik Resch, Ulya R Karpuzcu, Jian-Ping Wang, and Sachin S Sapatnekar. Analyzing the effects of interconnect parasitics in the stt cram in-memory computational platform. *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, 6(1):71–79, 2020.

[401] Masoud Zabihi, Zhengyang Zhao, DC Mahendra, Zamshed I Chowdhury, Salonik Resch, Thomas Peterson, Ulya R Karpuzcu, Jian-Ping Wang, and Sachin S Sapatnekar. Using spin-hall mtjs to build an energy-efficient in-memory computation platform. In *20th International Symposium on Quality Electronic Design (ISQED)*, pages 52–57. IEEE, 2019.

[402] Stephanie A Zajac, Amanda N Bozovich, Bernard G Rax, Joe Davila, Duc Nguyen, Wilson P Parker, Aaron J Kenna, Steven S McClure, Jason L Thomas, Kelly W Stanford, et al. Updated compendium of total ionizing dose (tid) test results for the europa clipper mission. In *2020 IEEE Radiation Effects Data Workshop (in conjunction with 2020 NSREC)*, pages 1–4. IEEE.

[403] Jintao Zhang and Naveen Verma. An in-memory-computing dnn achieving 700 tops/w and 6 tops/mm 2 in 130-nm cmos. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(2):358–366, 2019.

[404] Yang Zhang et al. Vertically integrated zno-based 1d1r structure for resistive switching. *Journal of Physics D: Applied Physics*, 46(14):145101, 2013.

[405] Quming Zhou, Mihir R Choudhury, and Kartik Mohanram. Tunable transient filters for soft error rate reduction in combinational circuits. In *2008 13th European Test Symposium*, pages 179–184. IEEE, 2008.

[406] Quming Zhou and Kartik Mohanram. Gate sizing to radiation harden combinational logic. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(1):155–166, 2005.

[407] Alwin Zulehner, Alexandru Paler, and Robert Wille. An efficient methodology for mapping quantum circuits to the ibm qx architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(7):1226–1236, 2018.