Exploring the Energy/Quality Trade-off of Non Volatile Memory in Intermittent Computing

Tesi di Laurea Magistrale in
Computer Science and Engineering - Ingegneria Informatica

Author: **Rei Barjami**

Student ID: 963611
Advisor: Prof. Luca Mottola
Co-advisors: Prof. Antonio Rosario Miele
Academic Year: 2022-23

Abstract

Small-scale IoT devices powered by energy harvested from the environment are gaining popularity due to their ability to eliminate batteries that are bulky, require high maintenance costs, and have a detrimental impact on the environment. However, the energy harvested from the environment is unpredictable, meaning it is not a stable power supply.

This makes the computation in these devices *intermittent*. Intermittent computing is characterized by an alternation between active computing periods and periods where the device is powered off, waiting to harvest enough energy from the environment to resume computing. In intermittent computing systems, frequent energy failures occur, resulting in the loss of data stored in volatile memory that has not been persisted in Non-Volatile Memory(NVM). Unlike ordinary devices, like laptops and smartphones, energy harvesting devices lack the computational and memory resources to equip a full-fledged operating system to manage recovery after an energy failure. To enable these devices to continue computing from the point where energy failed, it is necessary to save the execution state in NVM.

Writes in NVM to persist the state are an overhead. Although these operations do not directly contribute to program execution, they allow the program to appear as if it is executing continuously despite frequent interruptions. STT-MRAM, an emerging NVM technology, can be used as NVM for intermittent systems to lower this overhead since it has better energy efficiency characteristics compared to other NVM technologies like FeRAM, MRAM, and flash memories. Writes in STT-MRAM are stochastic in nature. Specifically, the probability of an STT-MRAM cell switching from one state to another during a write operation depends on the amount of current provided to it. This is an opportunity for us. By piloting the current for writing the cell, we can decrease the energy consumed by writes in STT-MRAM at the cost of introducing some errors in the written data.

In our research, we examine the benefits and drawbacks of using STT-MRAM with approximated writes as the persistent storage for an intermittent computing system. To do this, we build an experimental framework to simulate both approximated and correct executions of various benchmarks. This allows us to calculate the energy saved due to approximated writes and the decrease in the quality of the outputs. By analyzing this data, we can identify when the proposed approximate computing technique is effective, reducing a relevant amount of the NVM writes overhead while maintaining an acceptable quality in the produced output. Further, we explore, for each benchmark, the trade-off between quality and energy consumption. By exploring this trade-off, we determine the optimal approximation level that achieves the best energy efficiency while maintaining an acceptable reduction in the accuracy of the results.

Our results demonstrate that, for example, for an image decoder program, using approximated execution can save up to 50% of the energy used for a correct execution, with a Root Mean Squared Error (RMSE) of 35 compared to the correct execution, indicating that this technique can significantly reduce energy usage while still producing acceptable output quality.

However, the results for a Fast Fourier Transform program are vastly different. Despite applying the strongest level of approximation, the energy consumption is only reduced by 4% compared to a fully correct execution. At the same time, the quality of the output is significantly affected, with a high probability of obtaining results with an Average Relative Error (ARE) greater than 100%.

Keywords: intermittent computing, approximate computing, internet of things, energy efficiency, batteryless systems

Abstract in lingua italiana

Dispositivi di piccole dimensioni alimentati dall'energia raccolta dall'ambiente stanno diventando sempre più popolari grazie alla loro capacità di eliminare le batterie che sono ingombranti, richiedono costi di manutenzione elevati e hanno un impatto negativo sull'ambiente. L'energia raccolta dall'ambiente ha un comportamento imprevedibile, il che significa che non è una fonte di alimentazione stabile. Ciò rende il calcolo in questi dispositivi intermittente.

La computazione intermittente è caratterizzata dall'alternanza tra periodi di calcolo attivo e periodi in cui il dispositivo è spento, in attesa di raccogliere abbastanza energia dall'ambiente per riprendere l'esecuzione. Nei sistemi di computazione intermittente si verificano frequenti interruzioni dovute alla mancanza di energia, con conseguente perdita dei dati memorizzati nella memoria volatile che non sono stati resi persistenti nella memoria non volatile (NVM). A differenza dei dispositivi ordinari, come laptop e smartphone, i dispositivi che utilizzano energia raccolta dall'ambiente non dispongono delle risorse computazionali e di memoria per avere un sistema operativo che gestisce il corretto recupero dopo un'interruzione dovuta alla mancanza di energia. Per consentire la computazione dopo un'interruzione per mancanza di energia è necessario salvare lo stato dell'esecuzione nella NVM.

Le scritture nella NVM per salvare lo stato di esecuzione rappresentano un overhead. Anche se queste operazioni non contribuiscono direttamente all'esecuzione del programma, consentono al programma di sembrare in esecuzione continuamente nonostante le frequenti interruzioni. La STT-MRAM è una tecnologia emergente di NVM, può essere utilizzata come NVM per i sistemi intermittenti per ridurre questo overhead in quanto ha migliori caratteristiche di efficienza energetica rispetto ad altre tecnologie di NVM come FeRAM, MRAM e flash. Le scritture in STT-MRAM sono di natura stocastica. In particolare, la probabilità che una cella STT-MRAM passi da uno stato all'altro durante un'operazione di scrittura dipende dalla quantità di corrente fornita ad essa. Questa è un'opportunità per noi, poiché pilotando la corrente per la scrittura della cella possiamo diminuire l'energia consumata dalle scritture in STT-MRAM introducendo però errori nei dati scritti.

Nella nostra ricerca, esaminiamo i benefici e gli svantaggi dell'utilizzo di STT-MRAM con scritture approssimate come NVM per un sistema di computazione intermittente. Per fare ciò, costruiamo un framework sperimentale per simulare esecuzioni approssimate e corrette di vari benchmark, consentendoci di calcolare l'energia risparmiata grazie alle scritture approssimate e la riduzione della qualità degli output. Analizzando questi dati, possiamo identificare quando la tecnica di calcolo approssimativo proposta è efficace, riducendo una quantità rilevante dell'overhead di scrittura in NVM pur mantenendo una buona qualità nella produzione di output. Infine, esploriamo, per ogni benchmark, il compromesso tra qualità e consumo energetico. Esplorando questo compromesso, determiniamo il livello di approssimazione ottimale che garantisce la migliore efficienza energetica pur mantenendo una riduzione accettabile dell'accuratezza dei risultati.

I nostri risultati dimostrano che, per un programma decodificatore di immagini, l'utilizzo dell'esecuzione approssimata può risparmiare fino al 50% dell'energia utilizzata per un'esecuzione corretta, con un RMSE di soli 35 rispetto all'esecuzione corretta, indicando che questa tecnica può ridurre significativamente l'utilizzo di energia pur producendo una qualità di output accettabile. Tuttavia, i risultati per un programma di Fast Fourier Transform (FFT) sono molto diversi. Nonostante l'applicazione del livello di approssimazione più forte, il consumo di energia è ridotto solo del 4% rispetto a un'esecuzione completamente corretta, mentre la qualità dell'output è significativamente compromessa, con una elevata probabilità di ottenere risultati con un Errore Relativo Medio (ARE) superiore al 100%.

Parole chiave: computazione intermittente, calcolo approssimato, internet delle cose, efficienza energetica, dispositivi senza batteria

Acknowledgements

I want to thank my advisor, Prof. Luca Mottola, for his advice and all the support he gave me throughout my research. Working with him was inspiring, and I am genuinely grateful for the opportunity to work under his supervision.

I would also like to thank my co-advisor, Prof. Antonio Miele, for his invaluable help and contribution to my research. His insightful feedback has been critical in helping me face the challenges of my research.

Then, I want to thank my lab colleagues: Andrea M., Matteo, Andrea P., and Mattia. They made my time in the lab such a wonderful experience.

Finally, I would like to thank my family for their encouragement and support throughout my studies. Their faith in me has been a constant source of inspiration and motivation.

Contents

List of Figures

List of Tables

1 | Introduction

The popularity of the Internet of Things (IoT) has skyrocketed in recent years, leading to an increased demand for affordable and compact embedded devices. However, most of these devices rely on batteries as their primary energy source, which brings several challenges. For example, batteries can be dangerous if not adequately protected, expensive, bulky, require high maintenance costs, and have a detrimental environmental impact.

To overcome these issues, energy-harvesting devices provide a viable alternative as they can utilize energy from the environment without needing batteries. The energy harvested from the environment is unpredictable, meaning it is not a stable power supply. For this reason, these devices suffer from frequent energy failures, which makes computation *intermittent*.

Intermittent computing is characterized by an alternation between active computing periods and periods where the device is powered off, waiting to harvest enough energy from the environment to resume computing. Figure 1.1 illustrates how intermittent computing operations occur in bursts, where the device executes a computation until the energy stored in a small capacitor reaches the turn-off threshold. Upon reaching this threshold, the device powers off and the capacitor starts recharging until it reaches the turn-on threshold; at which point the computation resumes. The time that separates two consecutive execution periods is not constant as it depends on the environmental conditions that determine the availability of energy.

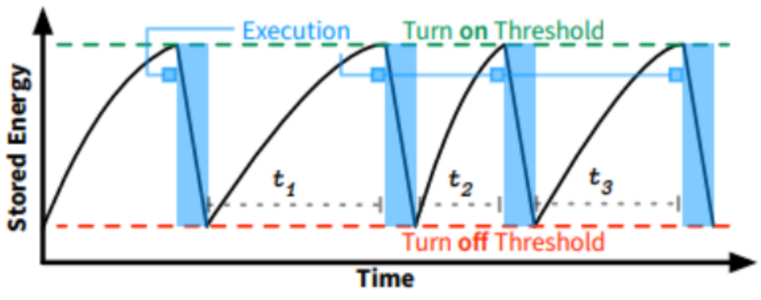


Figure 1.1: An intermittent execution.

Therefore, all the data stored in volatile memory are lost when an energy failure occurs. For ordinary devices, like laptops, this is not a problem since they have an operating system that allows the device to resume working correctly after an energy failure. However, since devices that work with intermittent computing are small and have limited memory and computational power, it is impossible to install an operating system. Therefore, to solve this problem and allow the program to progress in the presence of energy failures, we need to preserve the program's state in Non Volatile Memory (NVM). When the computation restarts, it allows continuing from where it powered off. Many techniques target the problem of making intermittent computing equivalent to continuous computing [2–4]. These techniques can follow two paradigms: checkpoint-based and task-based. Checkpoint-based intermittent computing periodically saves the program's state to NVM at specific places in the code. When the energy fails, the program can resume execution from the last checkpoint with a restore routine. Task-based intermittent computing involves dividing the program into smaller tasks that can be executed independently and saved to NVM individually. Tasks have "Transactional" semantics, so if energy fails during a task execution, the results of the task running during the fail are aborted. When the computation restarts, the execution restarts from this task rather than starting the entire program.

Saving the state in NVM is an overhead since the energy consumed by this operation does not directly contribute to the progress of the program execution. Moreover, NVMs are less energy efficient than volatile memories, so this energy overhead can heavily impact the system's overall energy efficiency. However, saving the state to NVM is essential for intermittent computing, as it enables the program to appear to execute continuously despite being interrupted multiple times.

Energy efficiency is crucial in intermittent computing systems since we want to maximize the work that can be performed during the active computing periods. A possible solution to solve this problem may be using novel NVM technologies such as STT-MRAM since it has better energy efficiency characteristics than other NVM technologies like FeRAM, MRAM, and flash memories [1]. The counterpart is that write operations in STT-MRAM are stochastic [5]. Specifically, the probability of an STT-MRAM cell switching from one state to another during a write operation depends on the amount of current provided to it. Monazzah et al. [6] noted that this characteristic of STT-MRAM can be used for an Approximate Computing (AC) technique. By piloting the current for writing the cell, we can decrease the energy consumed by writes in STT-MRAM at the cost of introducing some errors in the written data. We can exploit this approximation technique for reducing the NVM writes overhead of intermittent computing systems.

AC is a paradigm where a system provides an approximated result instead of a correct one while reducing the computational time and increasing energy efficiency. This paradigm is based on the idea that approximate results could be acceptable if they meet application requirements. For example, many applications in intermittent computing systems can be suitable for approximate computing [7], like FFT, image processing, and so on. However, it is important to note that AC cannot be applied blindly to any program. In some cases, like cryptography algorithms, even a slight loss in accuracy makes the program's output completely useless.

On the one hand, intermittent computing faces the challenge of scarce and unpredictable energy, necessitating high energy efficiency. On the other hand, AC can enhance the energy efficiency of systems. Therefore, applying AC to an intermittent computing system represents an opportunity, as it can address the energy efficiency challenges faced by the former. Notably, other works in literature have explored this idea of applying software-based AC techniques to intermittent computing systems [7, 8]. We explore the opportunities given by AC in intermittent computing systems but with a technique closer to the hardware.

Research Question

We consider a board for an intermittent computing system that uses an STT-MRAM as the persistent storage. We exploit the hardware AC technique proposed by Monazzah et al. [6] to reduce the overhead of writing in STT-MRAM.

The question we aim to answer is: *Can we obtain significant reductions in the overhead of writes in NVM, using STT-MRAM, while still maintaining a good quality in the produced outputs?*

Contribution

We design an approximation technique for intermittent computing systems based on the unique characteristic of stochastic switching of STT-MRAM as explained above. We chose to define different quality levels so that we can adjust the level of approximation.

We define an experimental environment composed of an MSP430-series microcontroller (MCU) that uses STT-MRAM as NVM. We build a framework for generating experimental results. This framework uses widespread simulators, such as NVSim [9] and MSPSim [10]. NVSim simulates the behavior of NVM technologies under various conditions, such as varying temperature, voltage, and current for write/read operations, and outputs the latency, endurance, area, leakage power, and energy consumption of the modeled NVM technology. Using this simulator, we can compute the energy consumption of approximated and correct writes on STT-MRAM. MSPSim is an emulator for the MSP430 microcontroller that accurately simulates the behavior of the MSP430 computing core, RAM, and peripherals. MSPSim allows us to run code and accurately measure performance without the need for physical hardware.

We use a set of heterogeneous benchmarks to evaluate the efficacy of our approximation technique. There is still no standard benchmark selection for evaluating intermittent computing systems. Other works in literature [11] use benchmarks from the MiBench2 suite[12] for evaluating intermittent systems. The benchmarks we select from this suite are FFT, PicoJpeg, and Susan edge detection. FFT calculates the Fast Fourier Transform of an input signal in the time domain. PicoJpeg, on the other hand, takes a JPEG image as input and outputs the decoded image in bitmap format. Susan edge detection is a lightweight edge detection algorithm that takes grayscale images as input. These benchmarks have distinct characteristics. For instance, FFT has a long computational phase and generates relatively short outputs, whereas PicoJpeg has a short computational phase but generates large outputs. We aim to consider various scenarios and highlight how our approximation technique effectiveness can vary depending on the program.

For each benchmark, we select a proper evaluation metric to measure the impact of our approximation technique on the accuracy of the results compared to the correct execution. For example, the evaluation metric for FFT is the Average Relative Error (ARE), for PicoJpeg, it is Root Mean Squared Error (RMSE), while for Susan, we selected precision and recall with a tolerance threshold. Then, we run these benchmarks with different approximation levels and evaluate how approximation impacts the accuracy of the obtained results and how much the energy overhead of writes in NVM decreases.

Finally, we quantitatively explore, for each benchmark, the trade-off between quality and energy consumption. By exploring this trade-off, we determine the optimal approximation level that achieves the best energy efficiency while maintaining an acceptable reduction in the accuracy of the results.

Our evaluation results indicate that our approximation technique can result in substantial energy savings. For instance, when applied to the PicoJpeg benchmark, the energy consumption for running the entire benchmark can be decreased by almost 50% compared to fully accurate execution. while sill preserving a high Quality of Results (QoR), as evidenced by a low RMSE value of only 35. However, the results for FFT are vastly different. Despite applying the strongest level of approximation, the energy consumption is only reduced by 4% compared to fully correct execution. At the same time, the QoR is significantly affected, with a high probability of obtaining results with an ARE greater than 100%.

1.1. Structure

This thesis is structured into five chapters. First, we give an overview of the state of the art of AC and intermittent computing and how these two can be combined. Then, we describe in detail our research question. After this, we describe the methodology we use to produce our experimental results, and finally, we discuss the experimental results we obtained.

Below is a summary of the chapters included in this work:

- **Chapter 2** provides an overview of state of the art in AC and intermittent computing and how these two concepts can be combined. First, we describe the opportunities and challenges presented by AC. Next, we survey a small selection of AC techniques proposed in the literature that are relevant to our work.

  We then discuss the opportunities and challenges of intermittent computing and describe the most important system techniques that support this type of computing. The chapter then emphasizes the importance of NVM in intermittent computing systems and compares some NVM technologies for this purpose.

  Finally, we introduce some techniques from the literature that combine AC with intermittent computing.

- **Chapter 3** articulates the research question addressed in this thesis. Firstly, we outline the context in which we place ourselves and in which the question is considered. Next, we elaborate on the advantages of applying our proposed approach. Subsequently, we discuss the drawbacks that may arise from our approach. Finally, we present the trade-off between energy savings and output quality.

- **Chapter 4** details our methodology to produce quantitative results to answer the research questions. First, we describe the evaluation environment and the design of our approximation technique. We build a framework to generate the experimental results composed of the widely used simulators NVSim and MSPsim. For both these simulators, we describe their main features. Finally, we describe the benchmarks we use to evaluate our technique, along with the error metrics that are specific to each of them.

- **Chapter 5** reports on our experimental results. In this chapter, we quantitatively evaluate our approach's effectiveness, considering the amount of energy we can save by reducing NVM writes energy cost and how much the accuracy of the result produced by the benchmarks decreases.

Our results demonstrate that, despite a slight reduction in the QoR, our approximation technique can lead to a significant reduction of up to 50% in energy consumption for programs like PicoJpeg. However, minor energy savings can significantly deteriorate the QoR in other cases, like for FFT.

- **Chapter 6** concludes the thesis and presents possible future work directions.

2 | Background

In this chapter, we will an introduction to Approximate Computing (AC) and intermittent computing. We present motivations, challenges, and state of the art implementations for both these paradigms. The background knowledge presented in this chapter is key for future reasoning in this thesis work.

2.1. Approximate Computing

With a noticeable increase in the number of applications with onerous resource requirements the computational and storage demands of nowadays systems have exceeded the available resources. As the amount of data managed by data centers increases, passing from 33 zettabytes in 2018 to 64.2 zettabytes in 2020 also electricity consumption is increasing significantly. Electricity consumption of US data centers is expected to increase from 91 billion kWh [13] in 2013 to 140 billion kWh in 2020 [14]. At the same time, we are also facing an enormous increase in the number of low-power devices that are causing an explosion in the amount of new data produced. For these reasons, energy consumption has become a key aspect in the design of computing systems. As performance demands are rising they will soon exceed the resource budget.

A solution for this problem is AC which is based on the idea of allowing selective approximation or occasional violation of the specifications of a task. The key point is that approximation can be applied on some parts of an application where a perfect result is not necessary and an approximate or less-than-optimal result is sufficient, for example applications that involve media processing (audio, video, graphics, and image), recognition, and data mining. In these applications a relaxation on the requirements on results correctness is allowed due to several factors [15]: perceptual limitations determined by the ability of the human brain to ‘fill in’ missing information and filter out high-frequency patterns; redundant input data that cause an algorithm to be lossy and still be adequate and noisy inputs.

2.1.1. Motivations

AC allows to trade-off accuracy in return for increased performance. AC approach can alleviate the scalability bottleneck [16], reducing execution time by early loop termination [17, 18], skipping memory accesses [19], offloading computations to an accelerator [20], improving yield [21]. However, in our work, we focussed on the energy efficiency improvements that can be obtained by AC.

In many scenarios AC can be proactively used for efficiency optimization, to create a tradeoff between results correctness and energy consumption. Approximation is well tolerated on a vast class of applications like programs that contain some non critical portions and errors in these portions do not affect significantly the Quality of Results (QoR). Esmaeilzadeh et al. [22] found that in many benchmarks like fft, imagefill, jmeint, raytrace, 98% of Floating Point operations can be approximated. Rahimi et al. [23] observed that in Image Processing applications a peak signal-to-noise ratio (PSNR) of 30DB is typically considered acceptable. Therefore if program execution is not 100% numerically correct due to few errors during computations the program can still "appear" to execute correctly. AC can be used on this type of applications for example by reducing the energy consumed in storage and memory access by performing a voltage scaling on SRAM supply-voltage or by performing early loop termination or with loop perforation. Another domain that is significantly affected by AC is IoT since devices such as sensors and actuators which interface with the physical world are by their own nature not correct due to the unavoidable presence of noise. AC can be utilized in this domain by simply increasing the already existing approximation level.

2.1.2. Challenges

Some applications are not well suited for approximation, for example in hard real-time systems on which the execution times of processing tasks need to be well known, in cryptographic algorithms or also in error detection algorithms like cyclic redundancy check (CRC). In image processing applications the definition of "good enough" quality is debatable since some people are more capable of analyzing the quality of an image than others. Typical error-resilient image processing algorithms can accept errors of up to 10%, but this same error would be unacceptable for a military system calculating the trajectory of a ballistic projectile. Another challenge is to understand, in the problems that allow an approximation, how large is the margin of error acceptance, so to find what is the maximum applicable level of approximation that does not exceed this margin of error. For this reason, it is important to ensure the quality of approximate computations through output monitoring.

Choosing the correct quality metric is another important step, take for example approximating a compression program with deviation in the output file size as the accuracy metric. Under approximation, the file size of an invalid output – a corrupt file – may even become equal to the file size of the exact output, Akturk et al. [24] proved that by using this quality metric the corrupt output may not be detected.

In conclusion, AC cannot always be applied regardless of the context, but we need to consider if the system is amenable to not fully correct executions.

2.1.3. Techniques

In this section, we will discuss some AC techniques found in the literature. We selected for this section the techniques that are most significant for a proper understanding of this thesis work.

Voltage Scaling

Voltage scaling reduces the energy consumption of hardware while inducting possible errors. For example, by reducing the set voltage in SRAM some energy will be saved when writing in the memory, but this will also increase the possibility of the bit not switching and so errors can be generated.

An important technique of voltage scaling, Near Threshold Computing (NTC), where the supply voltage V_{dd} is approximately equal to the threshold voltage of the transistors V_{th}, was presented by Dreslinsky et al. [25]. In this region the energy savings are almost the same as in subthreshold ($V_{dd}<V_{th}$) but with more favorable performances. At the cost of saving energy by scaling the supply voltage, there is a decrease in the frequency. For example for the Phoenix processor, presented by Seok et al. [26], from a nominal 9.13 MHz and 29.6 pJ/inst we translate to a 9.8$\times$ reduction in energy and a 9.1$\times$ in reduction of frequency. Dreslinsky at al. [25] also proved that in NTC there are more favorable variability characteristics with respect to subthreshold. The variability characteristics considered are process, voltage and temperature variations. Borkar et al. [27] observed that these parameter variations pose a major challenge for the design of high-performance microprocessors.

Another way of performing voltage scaling was presented by Monazzah et al. [6]. In their work, they discussed a hardware/software approach to adjust the tradeoff between energy and quality of write operations in STT-MRAM. STT-MRAMs suffer from a reliability issue, that is the switching is stochastic. During write operation, the applied write signal may be unable to change the value of the STT-MRAM cell, thus leading to a write failure. The lower the write signal voltage the higher the possibility to have a write failure because a cell is not switching value, but also lower energy consumption. Here different quality levels were defined. In each quality level a different level of approximation is performed, by applying different voltage on a write in the memory cell. This approach allows more configurability to approximation, having the possibility to give to critical data a higher quality level, so less write failure possibility, while to non critical data a smaller quality level with higher write failure possibility, but also better energy efficiency. In their work, they demonstrated to achieve up to 34% energy savings over a baseline STT-MRAM, with an acceptable quality of the generated outputs.

Loop Perforation

Loop perforation is an AC technique that works by skipping some iterations of a loop to reduce computational effort, making execution cheaper and faster.

This technique was first presented by Sidiroglou et al. [17]. They noted that application such as: video and audio encoders, Monte Carlo simulations and machine learning algorithms can trade off accuracy in return for increased performance using loop perforation, which consists in transforming loops to execute a subset of their iterations while skipping the rest. It is important to not perform this technique on critical loops, whose perforation causes the computation to produce unacceptable results, crashes, increase its execution time, or execute with memory errors, but only on tunable loops, whose perforation produces acceptably accurate computations with increased efficiency. Sidiroglou and his colleagues show the performance advantage of their technique by evaluating it on applications from the PARSEC benchmark suite, declaring that it delivers performance increases up to a factor of seven, with the capability to still maintain an acceptable computation accuracy changing the results that the application produces by less than 10%.

Inexact or Faulty Hardware

There are many AC techniques that fall in this class of approximation strategies. All these techniques work on new designs for faulty hardware, that can achieve better performance with respect to exact hardware in terms of speed, silicon area, power consumption and critical path length.

Kang and Kahng [28] propose an accuracy-configurable approximate adder, for which the accuracy of results is configurable during runtime. The proposed inexact adder, with bit-width N, is composed by (N/k - 1) sub-adders, each of which is a 2k-bit adder. Sub-adders are used to perform partial summations, to reduce the critical path delay the inexact adder avoids the carry chain, so the output of each sub-adder is incorrect when a carry input should be propagated to the results. In the general implementation, the output result will be correct when there are no errors in all (N/k - 1) sub-adders. Kang and Kahng demonstrate that with smaller k values, the minimum clock period and dynamic power can be reduced, but the probability of having a correct result will be decreased. This type of adder is accuracy configurable since by changing the value of k, the accuracy of the inexact adder can be controlled. The adder proposed by Kang and Kahng achieves approximately 30% power reduction versus the conventional exact adder, by accepting relaxed accuracy requirements.

Precision Scaling

Several AC techniques work on the idea of reducing the bit width of variables to reduce storage requirements and computing effort. For example, if we have a floating point(FP) variable that needs to be stored in double precision (64 bits) to be correct, we can store it in single precision (32 bits). Doing so computation will be faster and more energy efficient, but not exact.

Yeh et al. [16] proposed an application of precision scaling in the domain of physics-based animation. They proposed an architecture with a hierarchical floating-point unit, that leverages dynamic precision reduction, to enable efficient FPU sharing among multiple cores. In their work they show three main benefits of precision reduction: First, some FP computations may turn into trivial operations that do not require the use of an FPU. Second, it provides value locality: Precision reduction improves the locality that exists among similar objects in similar scenes and across iterations during constraint relaxation. Third, it enables the possibility of using smaller, faster, and more energy efficient FPUs to replace full precision FPUs. However, since in their proposed architecture the precision requirement can vary dynamically, the architecture still necessitates occasional access to full precision FPUs.

2.2. Intermittent Computing

In the last decade, we have seen a huge increase in interest in small low power computing devices. These devices are used in a very broad number of domains, such as IoT devices, transportation and logistics, healthcare, smart environment, personal and social domain and so on.

Today, the total number of IoT devices connected globally exceeds 7 billion, and it is going to increase to 22 billion by 2025 [29].

Hester et al. [30] noted that in many domains, a main criticality of these devices is that they run with batteries, this can be a problem, for example, when the devices are placed in dangerous environments (e.g., radioactive areas, pressurized pipes) in which battery recharging or replacement becomes a challenging task. A solution to this problem is provided by advances in energy harvesting technology, by forgetting batteries and surviving on energy harvested from the environment, small intermittently powered devices can monitor objects in hard-to-reach places without the battery maintenance burden.

This does not come for free, we have to take into account that computation in these devices is intermittent, since energy is not always available to harvest and, even when it is available, it needs to wait enough to buffer enough energy needed to compute a useful amount of work. In these devices energy is harvested by the ambient(for eg. solar energy, kinetic energy), however, the energy throughput produced by all these sources is way lower even than the consumption of a small device, which will then fall into many power failures that will unpredictably turn off the device multiple times per second. For this reason, execution in these devices is radically different to traditional battery-powered devices.

Batteryless intermittent computing is different from classic continuous computing since there is no stable power supply. Energy harvested from the environment availability depends on environmental conditions. Powering a device with it causes intermittent execution with an alternation between active computing periods and periods where the device is powered off, waiting to harvest enough energy from the environment to resume computing. For this reason, execution is intermittent. When an energy failure happens, the content of the device's volatile main memory is lost. If partial results have not been saved to the Non Volatile Memory (NVM), results produced so far are lost. Ordinary devices such as laptops and smartphones equip an operating system that manages a recovery after the energy failure. However, the devices used for intermittent computing are small and have limited storing and computing capabilities, for this reason, they cannot install an operating system. For this reason, having an intermittent execution creates the need of saving the work done before an energy failure to not start the entire computation all over again. There are several ways that we can find in literature to solve this problem, but they can be summarized into two paradigms: checkpoint/restore-based solutions and task-based solutions.

2.2.1. Checkpoint/Restore

Two routines are needed to implement this paradigm:

- Checkpoint: before having an energy failure the system saves its state into NVM. When and how to perform the checkpoint, which is an energy-expensive operation, is a critical choice. Many articles in the literature give a solution to this problem.[HarvOS 2017][Hibernus 2015] [MementOS 2011].
- Restore: this routine restores the state saved in NVM from a previous checkpoint.

By combining these two routines, performing a checkpoint before having an energy failure and restoring the saved memory state before resuming execution, the system acts as if it is running in a continuous execution without having to restart all over again after the device is powered off. Checkpoint mechanisms can be static when compiler-inserted or dynamic when dynamically-decided.

2.2.2. Task-Based

In this paradigm, a program is decomposed into various atomic tasks small enough to execute and save their results into NVM in a single execution period. The idea is to create a chain of tasks that communicate with each other through NVM, so the input of a task is the output of a previous one. By doing so, it is possible to create arbitrarily complex applications; [31] and [32] are examples of this approach, but they differ in how this communication chain between tasks is done.

Differently from checkpoint-based solutions, which may present problems in data consistency, in task-based solutions the used data will always be consistent, independently of where an energy failure happens. The task-based approach, however, requires some extra effort from the programmer, who has to deconstruct the program into tasks that execute atomically.

Algorithm 2.1 Checkpoints inconsistencies example.

```
1: Bool cond; {this variable is saved in NVM}
2: cond = false;
3: checkpoint();
4: cond = !cond;
5: if cond == true then
6:     print("executed correctly");
7: else
8:     print("inconsistently executed");
9: end if
```

2.2.3. Checkpoint Inconsistencies

The state of an intermittent execution of a program is considered consistent if it corresponds to the state of the continuous execution of the same problem, otherwise, it is inconsistent. Lucia et al. [33] have demonstrated that intermittent execution, with the use of checkpoints, can lead to state inconsistencies. The main problem is that the volatile state, such as registers stack and global variables, is lost after an energy failure of the device and rolled back to the last check-pointed value after a restore, instead, the NVM is not lost after a power off, and its value may be inconsistent with the volatile state restored by a checkpoint.

Let us focus on algorithm 2.1 to better understand what inconsistency is. Suppose that *cond* is a boolean saved in NVM, at line 2 its value is modified to false, and then a checkpoint is performed saving the volatile state into NVM, line 4 is executed and the value of *cond* is set to true. After that, an energy failure occurs. When the device will harvest enough energy to restart its execution it will first perform a Restore routine that restores only the state of the main memory, not of the NVM, then execution will restart from the line after the checkpoint, so line 6 will be executed. Since *cond* is saved in NVM its value will now be true, so in line 4 its value will be changed to false, and then the else branch will be executed. This execution is different from the one we would obtain by running the problem in a continuous execution, so it is inconsistent, and the produced result is wrong.

Checkpoints are critical operations since they involve NVM, which is slower and more energy-demanding than volatile memory, we should use checkpoints only when strictly required to. They can produce inconsistencies by mixing persistent and non-volatile states, they are also of variable length, depending on how much stack is being utilized during the execution, so it would be optimal to perform a checkpoint only when it would not be too long. For these reasons choosing when to perform a checkpoint is a non-banal problem, and many different solutions have been proposed in the literature[3, 11, 32].

2.2.4. Intermittent Computing Techniques

In this section, we will present some relevant techniques for implementing intermittent computing focusing on checkpoint-based approaches.

MementOS

MementOS is a static checkpointing mechanism [2], it has two parts:

1. A set of program transformation passes that automatically place checkpoints inside the code.
2. A library that contains state checkpointing and recovery routines.

At compile time MementOS modifies programs in two ways: First, it places trigger points. A trigger point is a call to a MementOS function that estimates the available energy. If this energy is below a certain threshold a checkpoint is taken, otherwise, the checkpoint is ignored. This allows MementOS to have a dynamic-like behavior even if it is a static checkpointing mechanism. Second, it wraps the program's main() function with code that allows restoring execution from a previous checkpoint. To suspend execution in time for a checkpoint to complete, MementOS should insert enough trigger points at compile time so that run-time energy trends are effectively sampled, but it should not insert too many measurements, otherwise, their energy cost will be predominant with respect to the execution's cost. MementOS offers three strategies for automatic checkpoint placement:

1. loop-latch: a trigger point is placed at each loop latch (the end of the loop body), resulting in an energy check for each iteration of each loop in the program.
2. MementOS places a trigger point after each function call, resulting in an energy check each time a function returns.
3. In timer-aided mode, MementOS adds to either the loop-latch or function-return mode a hardware timer interrupt that raises a flag at predetermined intervals. Each trigger point then checks the flag and proceeds with an energy check only if the flag is up. The flag is lowered again for the next trigger point.

Besides offering an automatic trigger point insertion mechanism, MementOS offers the possibility to a programmer to manually add trigger points by including a header file and placing function calls in the program. MementOS does not provide the possibility to allocate elements into NVM, and thus it does not include such memory in the checkpoint content.

HarvOS

HarvOS proposed by Bhatti and Mottola [3] is a static checkpoint solution that requires minimal intervention by the developer. This solution operates at compile time and dynamically adapts to varying levels of remaining energy at run-time while capturing the actual program execution through the control-flow graph (CFG) of the program.

In their solution they place calls to trigger functions by looking at the CFG of a program, having different strategies depending on the program constructs. For example, having a different behavior for branching statements as opposed to loops results in a placement of triggers ad hoc for a specific application. At the same time, another goal of this solution is to reduce the size of the checkpoints by placing trigger functions where the size of the allocated memory is reduced. When a trigger function is called, based on the current system state, a checkpoint can be performed. At compile time HarvOS performs four steps:

- Step 1: The worst-case memory usage throughout the code is estimated, this is used to compute the highest energy cost E_{CKPmax} for checkpointing.
- Step 2: Compute the maximum number of cycles C_{MCUmax} that can be executed after the device wakes up with a freshly charged energy buffer supplying energy. $E_{\text{wake-up}}$
- Step 3: This step takes as input the results of the ones before, which can be computed independently one from the other. In this step, C_{use}, the number of useful cycles the MCU can execute in a worst-case scenario, is computed. To do so it is considered that the device starts with energy equal $E_{\text{wake-up}}$ and after this, the device does not receive any additional energy from the environment. Also, it is considered that the device needs to spend E_{CKPmax} before having an energy failure to checkpoint its state. C_{use} is, in practice, the number of cycles the MCU can execute to make progress in the program.
- Step 4: The CFG of the program is computed, to every block in the graph is associated with the number of cycles required to execute it. Then the CFG is split into sub-graphs whose total stretch in the number of cycles is at most $C_{\mathrm{use}}/2$. At least one trigger function must be placed within each sub-graph. The distance in terms of clock cycles between two calls should be lower than C_{use} to have enough energy to reach the next call and perform the checkpoint.

To minimize the checkpoint size, HarvOS identifies in each sub-graph the block corresponding to the minimum size of allocated memory and places a trigger call right at the end of it.

Hibernus

Hibernus is a dynamic checkpoint mechanism proposed by Balsamo et al. [4]. In Hibernus, there are two states: active and hibernating. The hibernating state is a low-power mode(LPM) where the device consumes considerably less energy since any computation is stopped. The system moves between these states when the supply voltage (V_{CC}) passes thresholds. When V_{CC} drops below a low threshold V_{H}, a checkpoint is performed, and the system switches into the hibernating state. The checkpoint may then be restored when the supply voltage rises above a high threshold V_R and the computation resumes.

The Restore routine is performed only if an energy failure occurs while the board is in LPM. If not, there is no need to do so since, in LPM, the volatile memory does not lose its content. To implement Hibernus, the device must be equipped with a voltage comparator that triggers an interrupt when the capacitor reaches a given voltage threshold. Hibernus requires the programmer to put the initialization routine into its code and to specify the voltage thresholds V_R and V_H, which are critical values for the stability of this checkpoint mechanism.

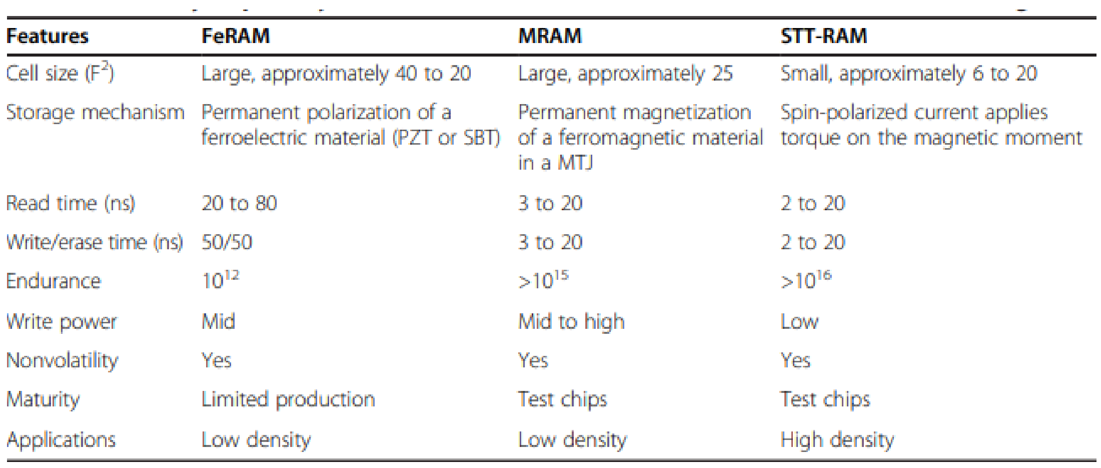
To overcome the problem of finding the correct values, Balsamo et al. proposed Hibernus++ [34], which extends the functionality provided by Hibernus with automatic calibration of such thresholds. Hibernus does not present any data flow inconsistencies even if it does not provide the possibility to allocate elements directly in NVM. This is done thanks to the LPM in which the system is set when the hibernate routine is called. In this state, the CPU is disabled. This means that no further computation is performed until there is enough energy, granting consistency over the elements present in NVM.

2.2.5. The Choice of the NVM in Intermittent Computing

A critical operation for implementing intermittent computing applications is saving the state of the device's main memory, typically SRAM, into NVM. Traditionally, microcontrollers have employed flash memory as the primary NVM technology. However, in flash memory, write/erase operations are energy-intensive, thus making this technology inefficient for frequent checkpointing.

An alternative to flash is given by the many emerging new NVM technologies such as ferroelectric RAM (FRAM), STT-MRAM and MRAM. These technologies have better power and performance characteristics compared to flash memory. It has been demonstrated that it is possible to design microcontrollers that integrate FRAM [35] and STT-MRAM [36, 37]. Moreover, Jayakumar et al. [38] have also shown that by having FRAM as unified memory, it is possible to perform in situ checkpoints, allowing to perform even long-running executions without having to care for energy failures. However, FRAM and STT-MRAM are still inferior to SRAM in both energy consumption and performance, so the best solution is to have hybrids of SRAM and FRAM/ STT-MRAM.

2.2.6. Comparison between NVMs



Features	FeRAM	MRAM	STT-RAM
Cell size (F^2)	Large, approximately 40 to 20	Large, approximately 25	Small, approximately 6 to 20
Storage mechanism	Permanent polarization of a ferroelectric material (PZT or SBT)	Permanent magnetization of a ferromagnetic material in a MTJ	Spin-polarized current applies torque on the magnetic moment
Read time (ns)	20 to 80	3 to 20	2 to 20
Write/erase time (ns)	50/50	3 to 20	2 to 20
Endurance	10^{12}	$>10^{15}$	$>10^{16}$
Write power	Mid	Mid to high	Low
Nonvolatility	Yes	Yes	Yes
Maturity	Limited production	Test chips	Test chips
Applications	Low density	Low density	High density

Figure 2.1: From the work of Meena et al. [1], comparison of different NVM technologies.

Meena et al. [1] compared FeRAM, MRAM and STT-MRAM. Figure 2.1 summarizes their comparison. Considering these technologies' read time and write energy, we can say that they outperform flash memory in both performance and energy consumption. Another critical aspect to consider is the endurance of these memories. Since in intermittent computing, checkpoints may be performed various times in a single second, it is important to have memories that can be overwritten numerous times without their bits dying. All these memories perform well also in this parameter, with at least 10^{12} erase cycle endurance. After evaluating the most important characteristics of these NVMs, we can conclude that they can all be valid alternatives to traditional flash memory.

2.2.7. STT-MRAM

By evaluating Figure 2.1, we noted that STT-MRAM is the most promising among these emerging NVMs substitutes for flash memory, with the best energy consumption and lowest read and erase time.

An STT-MRAM is made by many magnetic tunnel junctions (MTJ) cells that store the data. Each MTJ cell is composed of a barrier oxide layer placed between two ferromagnetic layers. The ferromagnetic layer, whose magnetic orientation is permanently fixed, is called Reference Layer(RL). At the same time, the other one is named Free Layer(FL) since its magnetic orientation can be arbitrarily rotated. In each MTJ cell, values are stored in terms of resistance states. When the magnetizations of RL and FL are parallel, it is in a low resistance state. Instead, when the magnetization of FL and RL are antiparallel to each other, the MTJ is in high resistance state. By applying a bidirectional current to the MTJ, we can change the magnetic orientation of the free layer. If the write current flows from FL to RL, the magnetization state switches to low resistance. Instead, the magnetization state changes to high resistance if the current flows in the opposite direction.

There are some things to consider when dealing with this kind of memory, Devolder et al.[39] noted that STT-MRAMs suffer from a reliability issue, the stochastic switching; during a write operation, the applied signal may be unable to switch the value of the STT-MRAM cell, leading to a write failure. Moreover, Bi and Wu [40] noted that Write failure is an asymmetric phenomenon in STT-MRAMs, with $0 \rightarrow 1$ transition requiring a higher current to make the STT-MRAM cell reliably commute compared to the value needed to do so in a $0 \rightarrow 1$ transition. In many contexts, however, the exact correctness of computations is not required, like in applications resilient to a certain degree of errors like AC.

As stated by Monazzah et al. [41] the probability of an MTJ failing to switch its state can be calculated by:

$$P_{\mathrm{wf}}(t_{\mathrm{w}}) = exp\left(-t_{\mathrm{w}}\frac{2\mu_{\mathrm{B}}p(I_{\mathrm{w}} - I_{\mathrm{C}_0})}{(c + \ln(\Pi^2\frac{\Delta}{4}))(em(1+p^2))}\right) \tag{2.1}$$

Where Δ is the thermal factor, I_{C_0} is MTJ's critical switch current at 0 Kelvin, c the Euler constant, e the magnitude of electron charge, m the magnetic momentum of FL, μ_{B} the Bohr magneton, I_{w} and t_{w} the write pulse width. By controlling the write pulse and current, we can change the probability of the MTJ failing to switch.

For its good characteristics as a substitute for flash memory as a NVM in the context of intermittent computing and its stochastic switching that adapts perfectly to the context of AC, we decided to base this work on a microcontroller using STT-MRAM as a NVM.

2.3. Intermittent AC

Energy consumption is an essential issue in computation in general. Still, in the context of intermittent computing, this aspect is even more significant since the efficacy of intermittent computing systems depends on the amount of energy they have at their disposal. Applications running on transiently powered devices are typically tolerant to approximation. For this reason, it is promising to synergize the domain of AC, which allows for increased energy efficiency of executions with intermittent computing. Thanks to approximation, we can increase the amount of progress computed by an intermittent computing system before it runs out of power.

Ordinary intermittent computing solutions maintain equivalence to continuous executions by creating a persistent state on NVM, enabling computations to advance in the presence of energy failures. Since NVM memory is slower and more energy-demanding than volatile memory, checkpointing in NVM heavily impacts performance causing system throughput to reduce while energy consumption increases. Van Der Woude & Hicks [11] noted that due to checkpointing, the total run-time overhead could reach up to 170% the cost of the application processing.

Ganesan et al. [8] propose a set of approximation techniques for intermittent computing: subword pipelining and subword vectorization. Using these techniques makes it possible to transform processing on energy-harvesting devices from all-or-nothing to as-is computing. The idea is to accept the approximated result achieved before the power went off and then execute the next task when power is restored instead of resuming the computation from a checkpoint that leads to the correct result. They propose to process data at subword granularity rather than word granularity. First, each word is split into subwords. Then subwords are processed in order from the most to the least significant. Processing each subword generates an approximate result. If a energy failure occurs before all subwords are evaluated, and the result of the already processed subwords is acceptable, the remaining subwords get ignored upon a restart, and execution continues with the next operation. They propose:

- Subword pipelining: decompose high-latency instructions into smaller subword operations.
- Subword vectorization: merge low-latency instructions, allowing parallel processing on the most significant subwords of different data elements.

Using these techniques, starting from word-based computing where the entire word must be processed to generate a result, we can obtain an iterative model where a partial approximate result is generated after the first subword gets processed and the results improve with each subsequently processed subword.

3 | Research Question

In this chapter, we introduce the research question we target in this thesis. To do so, in Section 3.1, we first define the base setting for intermittent computing. From this, we derive three new possible settings obtainable by reducing the cost of Non Volatile Memory (NVM) writes using a hardware Approximate Computing (AC) technique. In Section 3.2, we discuss the benefits of applying this technique and how applying approximation to reduce NVM writes energy cost affects the accuracy of the produced results in an intermittent computing system. Finally, in Section 3.3, we provide an overview of the energy/quality trade-offs we consider.

3.1. Settings

Here we introduce the base scenarios in which an intermittent computing system works. By reducing the NVM write energy cost from these base scenarios, we derive three new possible cases, each exploiting the energy saved to gain different benefits.

An intermittently-powered device executes its program in bursts as energy becomes available. These bursts are separated by recharge periods, during which the device waits to harvest from the environment enough energy to perform a meaningful amount of computation. Two approaches exist to ensure that the computation is equivalent to a continuous computation: checkpoint-based and task-based.

Checkpoint-based systems

Intermittent systems that use checkpoint-based approaches, shown in Figure 3.1, preserve the program's state, including its registers, stack, global variables, and code in NVM. This ensures that the state can be restored after an energy failure occurs, allowing the computation to resume from where it stopped in the previous energy burst.

When the system has enough energy to start computing, the program's state is restored from the last checkpoint saved before the energy failure occurred. At this point, the computing phase begins, during which the program progresses in its execution: $T_{\text{cmp,c-b}}$ is the duration of this computing phase. The workload is the amount of progress the

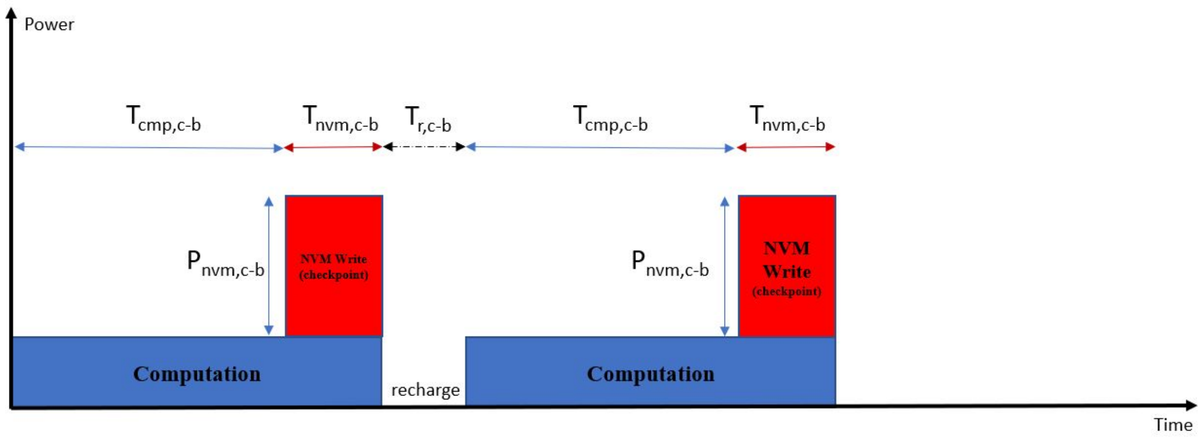


Figure 3.1: Checkpoint-based intermittent computation.

system makes in an energy burst, and it depends on $T_{\text{cmp,c-b}}$. Right before an energy failure occurs, the system executes the checkpointing routine. It takes a certain amount of time $T_{\text{nvm,c-b}}$ to complete, and it has an energy cost of $E_{\text{nvm,c-b}}$. When an energy failure occurs, the device must wait a recharge period in which it recharges its capacitor with energy harvested from the environment. This period is denoted as $T_{\text{r,c-b}}$.

Moreover, it is worth noting that when writing to NVM, the device needs to be running in active mode, so with the use of the components it uses during the computing phase. This is because we consider systems with no Direct Memory Access (DMA) available, which means that the computing unit must be active and involved in the write operation. However, the program is not progressing or making any further computation during this period. The device is solely running to write to NVM, with no other operations being performed. For this reason, while executing the checkpointing routine, energy is consumed without contributing to the actual progress in the program execution. This can be seen as a form of energy waste, but it is necessary to ensure that the computation can be resumed after an energy failure with the correct state.

The value of $E_{\text{nvm,c-b}}$, the energy consumption of NVM writes, so the area in red in Figure 3.1, can be calculated as $T_{\text{nvm,c-b}} \times P_{\text{nvm,c-b}}$. Taking this into account from now on we will always discuss in terms of energy consumption of NVM writes and not power consumption.

Task-based systems

In intermittent systems using task-based approaches, shown in Figure 3.2, the program is decomposed into tasks small enough to execute and save their results into NVM in a single energy burst. Furthermore, the tasks have transactional semantics. This means

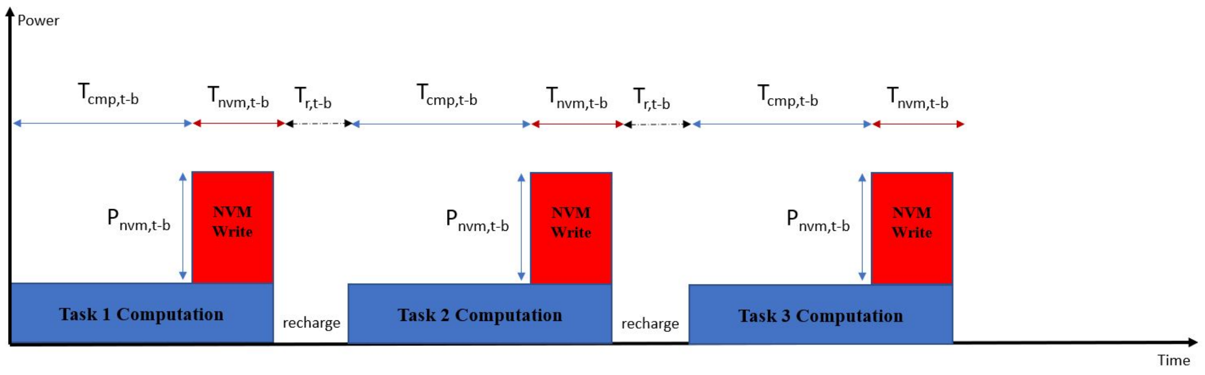


Figure 3.2: Task-Based intermittent computation.

that if we have an energy failure while executing a task that has not completed, all the operations done by that task are aborted. In the next energy burst, the computation will restart from the beginning of this task. With this approach, the idea is to create a chain of tasks that communicate to each other through NVM, so the output of a task can be the input of the next one.

The task-based approach requires more effort from the programmer, who has to deconstruct the program into a chain of tasks. In Figure 3.2, we can see that $T_{\text{cmp,t-b}}$ is the duration of the computational phase of the task. In Figure 3.2, the dimension of the tasks is the same for simplicity. We denote with $E_{\text{nvm,t-b}}$ the energy required to write the task output in NVM. When a power failure occurs, the system must wait for its capacitor to recharge before resuming the computation. We denote this period $T_{\text{r,t-b}}$.

3.2. Questions

Writes in NVM are energy-hungry operations, consuming a relevant amount of the energy stored by the device's capacitor. Van Der Woude et al. [11] noted that checkpointing might add up to 60% run-time overhead with peaks at 170%. This study investigates the potential benefits of using AC to reduce the energy consumed during write operations in NVM. Specifically, the technique we explore consists in adjusting the current applied to the writes in NVM. However, approximating writes in NVM also introduces errors in the data saved. These errors introduced by the approximation may significantly reduce the quality of the generated output.

With this thesis, we aim to answer the following research questions quantitatively:

1. What are the benefits of reducing the energy consumption of NVM writes?
2. How does this reduction impact the Quality of Results (QoR)?

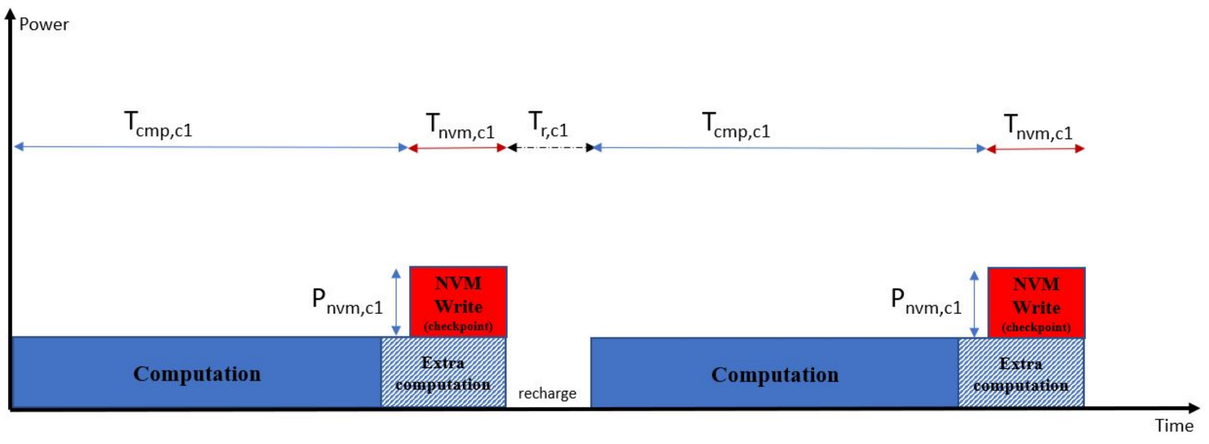


Figure 3.3: Increasing computing phase by reducing NVM write energy cost.

3. Can we obtain significant reductions in the energy consumptions of NVM writes while maintaining an acceptable QoR?

To examine the advantages and the drawbacks that can be obtained by using AC to reduce the cost of writes in NVM, we identify three potential settings:

- **Setting 1**, the capacitor is fixed, and a checkpoint-based approach is used.
- **Setting 2**, the capacitor is fixed, and a task-based approach is used.
- **Setting 3**, the capacitor is not fixed, but the workload for each energy burst is fixed.

By exploring these settings, we aim to highlight the benefits of reducing the cost of NVM writes in intermittent computing systems. For each case, we will also investigate the effects of introducing errors resulting from approximation to provide a comprehensive understanding of the trade-offs between energy efficiency and output QoR.

3.2.1. Benefits

The energy saved by reducing the energy required to perform writes in NVM can be invested to obtain different advantages in an intermittent computing system. Here we explore what these advantages in the settings introduced above are.

Setting 1: Fixed capacitor dimension, checkpoint-based

In the first setting, we have a fixed capacitor, so we cannot modify its dimension, and we are using a checkpoint-based system, where the NVM is used to save the machine's state allowing the device to make progress.

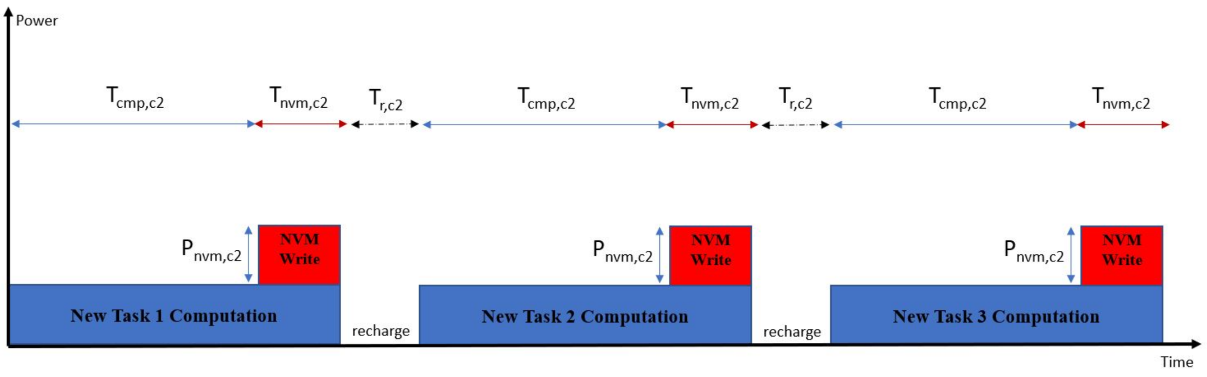


Figure 3.4: Re-organize tasks by having longer computing phase, by reducing NVM write energy cost.

Since the capacitor size cannot be altered, the energy saved by reducing the cost of saving the state in NVM can be invested to extend the computing phase, as shown in Figure 3.3. This, in turn, increases the amount of computation the device can perform during each energy cycle. Furthermore, since the capacitor is fixed, the recharge time is the same as when the checkpointing data is not approximated.

A longer computational phase allows the program to make more progress in a single charge, which reduces the number of energy cycles, and so checkpoints needed to complete the execution of the program. This longer computing phase improves system throughput since energy failures are fewer, and the system's uptime increases. Longer uptime periods also make the system more available to perform jobs and handle requests, increasing the system's availability. So we increase the energy spent on useful things.

For this setting, we can see comparing Figure 3.3 with Figure 3.1 that the length of the computing phase is greater than the length of the computing phase in the regular checkpoint-based model, so $T_{\text{cmp,c1}} > T_{\text{cmp,c-b}}$. This is because the energy for writing the state in NVM $E_{\text{ckp,c1}}$ is smaller than $E_{\text{ckp,c-b}}$, so the energy saved can be invested in the computing phase. Since the capacitor size is fixed, the recharge time remains the same, so $T_{\text{r,c1}} = T_{\text{r,c-b}}$.

Setting 2: Fixed capacitor, task-based systems

Here we examine the setting where the intermittent computing system uses a task-based paradigm and the capacitor size is fixed. In this setting, we can see from Figure 3.4 that the advantage we obtain is the possibility to re-factor the program's decomposition in longer tasks with respect to the original task-based scenario depicted in Figure 3.2.

The programmer does the decomposition in tasks of the program. By reducing the energy

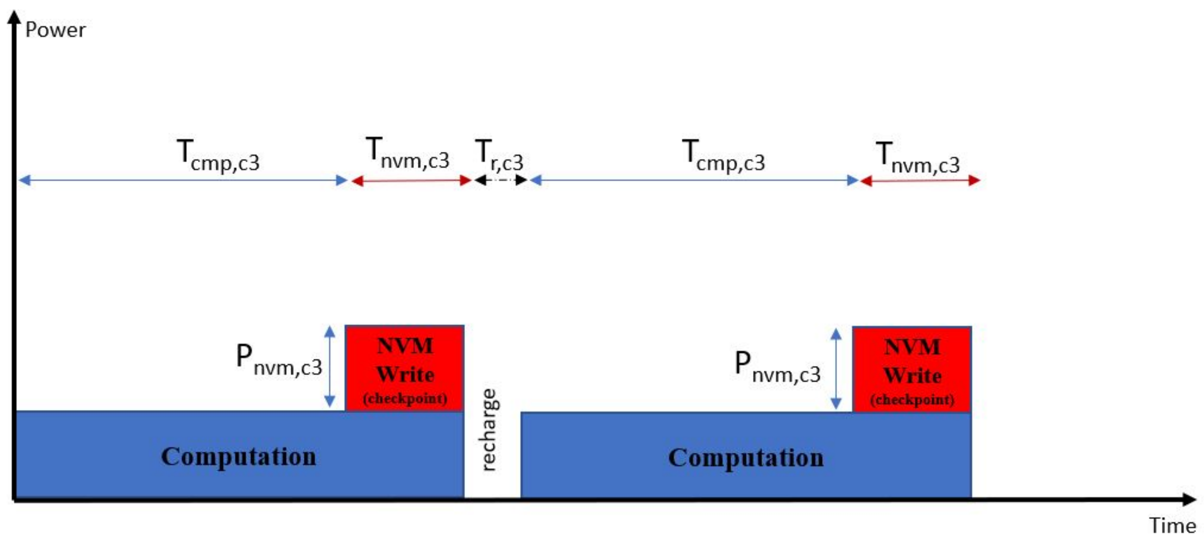


Figure 3.5: Using a smaller capacitor allows for shorter recharge times.

for writing the task output in NVM and investing this saved energy in the computing phase, the programmer can decompose the program into longer tasks. Longer tasks enable the program to be completed in fewer tasks overall [42], which reduces the number of energy failures that occur during the execution of the program. This means the system's uptime increases, increasing availability and throughput, similar to the checkpoint-based setting we discussed earlier.

We can now compare Figure 3.4, where we reduced the energy cost of NVM writes, allowing programmers to re-factor the tasks into longer ones, with the regular task-based model in Figure 3.2. We can see that the execution time of longer tasks $T_{\text{cmp,c2}}$ is higher than $T_{\text{cmp,t-b}}$. The energy consumption of writes of the output in NVM is smaller, so $E_{\text{nvm,c2}} < E_{\text{nvm,t-b}}$. Lastly, since the capacitor size is fixed, the recharge time remains the same; this means $T_{\text{r,c2}} = T_{\text{r,t-b}}$.

Setting 3: Capacitor not fixed, fixed computing workload

In this third setting, we consider an intermittent computing system where the size of the capacitor is not fixed, giving us the flexibility to adjust its dimension. However, the program's progress in a single energy burst is fixed, so we must dimension the capacitor size considering this constraint. We place this constraint since we want to consider a system with the same availability and throughput as the original case. If we reduce the computing phase length, we can use smaller capacitors, but this increases the number of energy failures, decreasing availability and throughput. The benefits of this approach can be applied to both checkpoint-based and task-based intermittent computing systems.

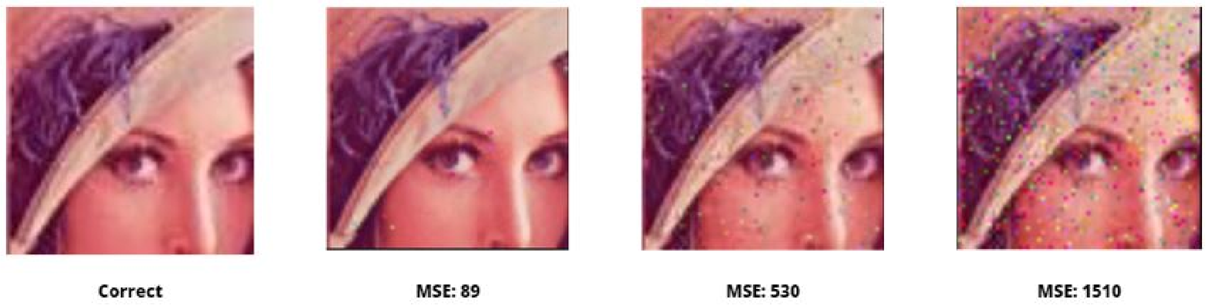


Figure 3.6: Mean square srror increase with higher approximation levels.

In this case, the energy saved by reducing NVM writes energy permits us to use smaller capacitors to perform the same computing work in a single energy cycle as in the original case.

Using smaller capacitors has several advantages. Firstly, they have less leakage power, meaning they lose less energy when unused. Secondly, smaller capacitors reach the operating voltage sooner, which enables them to recharge faster. Faster recharge times translate into reduced downtime, which increases system availability. Thirdly, smaller capacitors occupy smaller areas, making them well-suited for embedded systems where space is often a constraint.

When comparing the checkpoint-based system with reduced NVM write costs shown in Figure 3.5 to the standard checkpoint-based model in Figure 3.1, we can observe that the computing phase remains unchanged, so $T_{\text{cmp,c-b}}$ is equal to $T_{\text{cmp,c3}}$. However, the energy consumed for NVM write operations is reduced: $E_{\text{nvm,c3}}$ is less than $E_{\text{nvm,c-b}}$. As a result, a smaller capacitor can be used, meaning $C_{\text{cap,c3}}$ is smaller than $C_{\text{cap,c-b}}$. This leads to shorter recharge periods: $T_{\text{r,c3}}$ is smaller than $T_{\text{r,c-b}}$. The same observations hold for intermittent computing systems that use a task-based paradigm.

3.2.2. No Free Lunches

Now we discuss the drawbacks of approximating writes in NVM. The same approximation technique was utilized in the settings examined in Section 3.2.1. The energy savings of reducing the cost of NVM writes can be used to obtain different benefits. However, the errors introduced by reducing this energy cost are of the same nature in all the described settings.

Performing approximate writes in NVM reduces the energy cost of writing data, but it comes at the cost of introducing errors in the saved data. These errors can affect the quality of results, making the output unusable. The higher the degree of approximation

	Recharge Time	CP Length	QoR	NVM energy cost
Case 1	$T_{\mathrm{r,c1}} = T_{\mathrm{r,c\text{-}b}}$	$T_{\mathrm{cmp,c1}} > T_{\mathrm{cmp,c\text{-}b}}$	$Q_{\mathrm{c1}} < Q_{\mathrm{corr}}$	$E_{\mathrm{nvm,c1}} < E_{\mathrm{nvm,c\text{-}b}}$
Case 2	$T_{\mathrm{r,c2}} = T_{\mathrm{r,t\text{-}b}}$	$T_{\mathrm{cmp,c2}} > T_{\mathrm{cmp,t\text{-}b}}$	$Q_{\mathrm{c2}} < Q_{\mathrm{corr}}$	$E_{\mathrm{nvm,c2}} < E_{\mathrm{nvm,t\text{-}b}}$
Case 3	$T_{\mathrm{r,c3}} < T_{\mathrm{r,c\text{-}b}}$	$T_{\mathrm{cmp,c3}} = T_{\mathrm{cmp,c\text{-}b}}$	$Q_{\mathrm{c3}} < Q_{\mathrm{corr}}$	$E_{\mathrm{nvm,c3}} < E_{\mathrm{nvm,c\text{-}b}}$

Table 3.1: Summary of advantages and drawbacks of the three cases introduced above, assume in setting 3 we have a checkpoint-based approach.

used, the more errors are introduced, resulting in a greater decrease in the QoR, as shown for an example for an image processing application in Figure 3.6. Not all programs can tolerate approximation, so we must carefully consider the program that the intermittent computing system is meant to run before using approximations in NVM.

In intermittent computing, the checkpoint written in NVM or the output of a task is typically used as input in the next power cycle. This means that errors introduced in the early energy cycles will have a greater impact on the final result quality than those introduced in later energy cycles. Therefore, tuning the level of approximation used in NVM writes is crucial. For example, one approach to reduce the snowball effect caused by errors is to use an increasing level of approximation, starting with a small approximation in the early phases and gradually increasing to stronger approximations in later phases.

The errors that arise from approximating writes in NVM are the same in all the cases discussed earlier. The difference lies in how we invest the energy saved. The QoR in the scenario where NVM writes are carried out correctly without any approximation is denoted as $\mathrm{Q_{corr}}$. The QoR for Setting 1, denoted as $\mathrm{Q_{s1}}$ and for Setting 2, denoted $\mathrm{Q_{s2}}$, will be lower than $\mathrm{Q_{corr}}$. In the same vein, the QoR for Setting 3, denoted $\mathrm{Q_{c3}}$, will also be less than $\mathrm{Q_{corr}}$.

3.3. From Qualitative to Quantitative

In this chapter, our focus has been on qualitatively examining the impact of using AC on reducing the energy cost of NVM writes in intermittent computing systems. Through exploring various settings with different types of advantages, we aimed to gain a better understanding of how AC affects energy consumption in such systems. However, to truly quantify the extent of this impact, we will perform a detailed experimental analysis in the next chapters.

Our aim is to run different benchmarks on a system where writes on NVM are approxi-

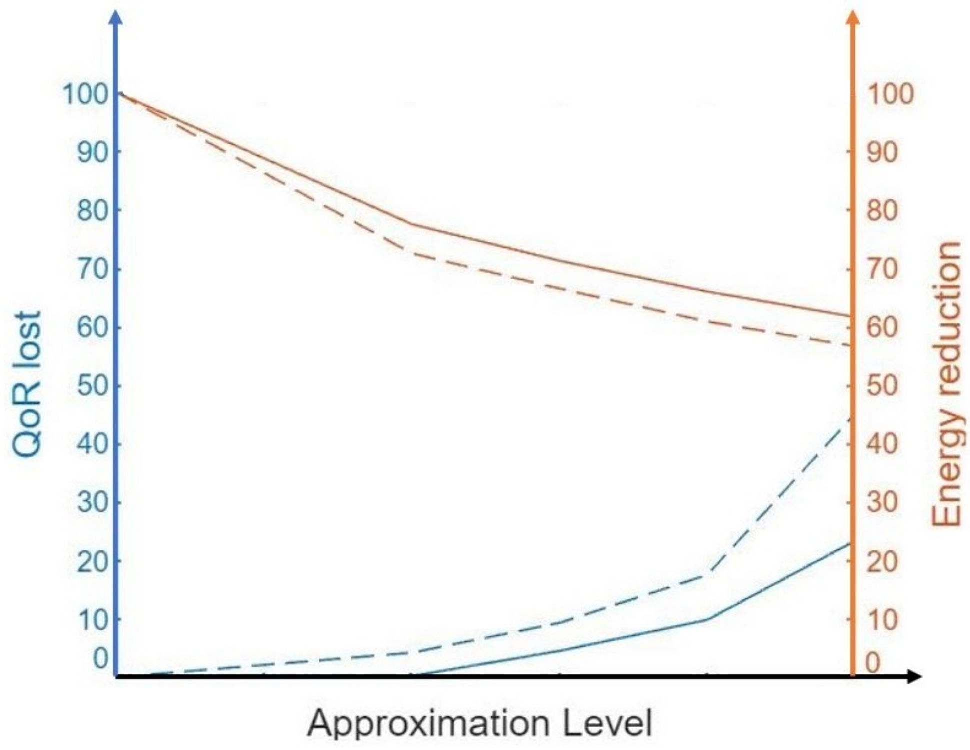


Figure 3.7: Tradeoff error/quality, stronger approximation will increase the energy savings but reduce the QoR, the dotted and the continuous lines refer to two different programs, we can see that different programs have different trends.

mated and assign numerical values to all relevant variables presented in Table 3.1. This enables us to accurately measure the reduction in energy consumption and the degradation in the QoR obtained by using AC in intermittent computing systems. In addition, we also evaluate the tradeoff between errors and energy saved through our experimental analysis exploring different levels of approximation and measuring the corresponding amount of energy saved as well as the magnitude of error introduced.

The *tradeoff between energy and quality* is an important consideration in designing and operating an intermittent computing system. The degree of approximation used in NVM writes directly affects this tradeoff. A higher degree of approximation results in more significant energy savings but also greater errors and a decrease in the quality of results. Conversely, using a lower degree of approximation results in fewer errors and a higher quality of results but with more limited energy savings. *Finding the optimal level of approximation* is crucial in achieving the desired tradeoff between energy and quality of the output.

Generally, the amount of energy saved and the QoR degradation depend on the specific program the system is running and on the device and NVM characteristics. Intuitively, as shown in Figure 3.7, the trends of this energy/quality tradeoff heavily depend on the program the system is running. While for some programs, the amount of energy saved is big even with a small degradation in the QoR, for other programs, an almost imperceptible reduction in the energy consumption may cause the QoR to decrease drastically, making the output unusable. In intermittent computing systems, the tradeoff between energy and quality is further complicated because errors introduced in the early power cycles have a greater impact on the final result quality than errors introduced later. Therefore, it is key to tune the approximation level used in NVM writes carefully to maximize energy saving while still achieving the desired QoR.

In summary, the key aspect we explore in this thesis is understanding the *quality/energy tradeoff and the effectiveness of approximating writes in NVM.* The tradeoff between energy and quality is critical to find the optimal level of approximation to achieve the desired balance between energy savings and quality of results.

4 | Methodology

In this chapter, we describe the methodology used to perform the quantitative evaluation, answering the research questions described in the previous chapter. First, section 4.1 describes the target architecture and the design of our approximation technique to reduce the energy consumption of Non Volatile Memory (NVM) writes. Next, in Section 4.2, we describe the workflow to generate the results that we show in Chapter 5. Lastly, in Section 4.3, we describe the benchmarks we use to evaluate our technique and the quality metrics specific to each of them.

4.1. Target Hardware Platform

In this section, we describe the platform we use to conduct the experimental study we discuss in Chapter 5. We start by describing the NVM we select for our platform and the approximation technique we utilize to reduce the cost of writing into it. Then, we present the selection of the rest of the platform.

4.1.1. STT-MRAMs and Approximating Write Operations

In Chapter 3, we explored the potential benefits of reducing energy consumption in NVM by utilizing approximate writes instead of correct ones. Here, we present in detail the approximation technique we employ, based on the stochastic switching present in STT-MRAM memories. In Section 2.2.7, we note that STT-MRAM performs better in terms of energy consumption and latency than other NVM technologies. Therefore, it is more suitable for embedded systems with strict energy constraints, which is the focus of our thesis.

As discussed in section 2.2.8, STT-MRAM is susceptible to stochastic switching. This is when the applied signal during a write operation cannot switch the value of the STT-MRAM cell, leading to a write failure. We must apply a pulse of a specific width and current to the STT-MRAM cell to achieve a successful write operation. The higher the current and width of the pulse, the greater the probability of successfully switching the

Quality	WER	Set Current (μA)
Q0	10^{-8}	1153
Q1	10^{-6}	865
Q2	10^{-5}	769
Q3	10^{-4}	673
Q4	10^{-3}	577

Table 4.1: Relationship between WER and set current for a 32nm STT-MRAM.

STT-MRAM cell. However, to achieve almost 100% accuracy, we need to use a high current for write operations. In our case, we do not always require fully accurate writes, so we can introduce errors and use smaller currents, thereby reducing energy consumption significantly. This creates a knob between the energy consumption and the Write Error Rate (WER) of writes in STT-MRAM.

To better understand the relationship between the current and the WER in a 32nm STT-MRAM cells, Monazzah et al. [6] conduct several Monte Carlo simulations, by varying the write current while pulse width was fixed at 10nS. Based on this relationship, we define different quality levels, each with a specific WER and the corresponding current required to achieve it. The baseline quality level, Q0, represents the correct case with a WER of 10^{-8}. Quality levels progress from Q1 to Q4, where Q4 provides the best energy efficiency but with a higher WER of 10^{-3}. Table 4.1 summarizes our defined quality levels, their WER, and their set current.

STT-MRAMs have the useful property of allowing actuation at the block granularity, which means that the memory can be divided into sections, and a specific quality level can be assigned to each section. This feature allows for deciding which quality level to give to each saved data and determining in which section of the STT-MRAM the data will be written. STT-MRAM can be used as the only NVM due to this property. Critical data that cannot be approximated, such as counters in a loop, can be saved in a section where writes are performed correctly. In contrast, data that can be approximated can be saved in a different section where energy consumption is more efficient.

In summary, the technique we use for approximating writes in NVM is based on the concept of stochastic switching of STT-MRAMs. We are able to exploit the knob between write energy and quality offered by STT-MRAM to define different quality levels with varying WERs and energy consumption.

4.1.2. MCU

After selecting STT-MRAM as NVM, in this section, we describe the rest of the platform we use to perform our evaluations.

The platform we select is composed of a Texas Instruments board that utilizes the MSP430 microcontroller (MCU) family. Specifically, we chose three different MCUs, namely the MSP430L092 [43], MSP430G2x5 3[44], and a newly-designed MCU presented by Singhal et al. [45], we call this MSP430Singhal from now on, all based on the MSP430 CPU. Although all these MCUs have the same RISC16 architecture, they differ in energy consumption, amount of RAM and ROM, and clock. Table 4.2 summarizes the difference between these MCUs. These MCUs are commonly used in IoT devices and have limited energy consumption, making them ideal for an energy harvesting scenario. We chose multiple MCUs with different power consumptions to emphasize that the proposed approximation technique's benefits are strictly related to the overall system's energy consumption and not just the NVM's energy consumption. Even if we save energy in writing to the NVM, the overall energy consumption will still be high if the CPU consumes a lot of energy. Therefore, we chose to highlight the importance of considering the system's overall energy consumption.

Figure 4.1 summarizes the platform we use for our evaluation. It is the standard MSP430 MCU architecture, but instead of the FeRam or the flash memory, we placed an STT-MRAM instead as NVM.

	MSP430Singhal	**MSP430L092**	**MSP430G2x53**
Power supply (V)	3	1.3	2.2
Frequency (MHz)	16	5	16
Active Power $(\mu W/MHz)$	28.3	58.5	506
Sleep power (μW)	0.32	8	3.74
Main Memory	$8kB$ SRAM	$2kB$ SRAM	$4kB$ SRAM

Table 4.2: Comparison between the selected types of MCUs.

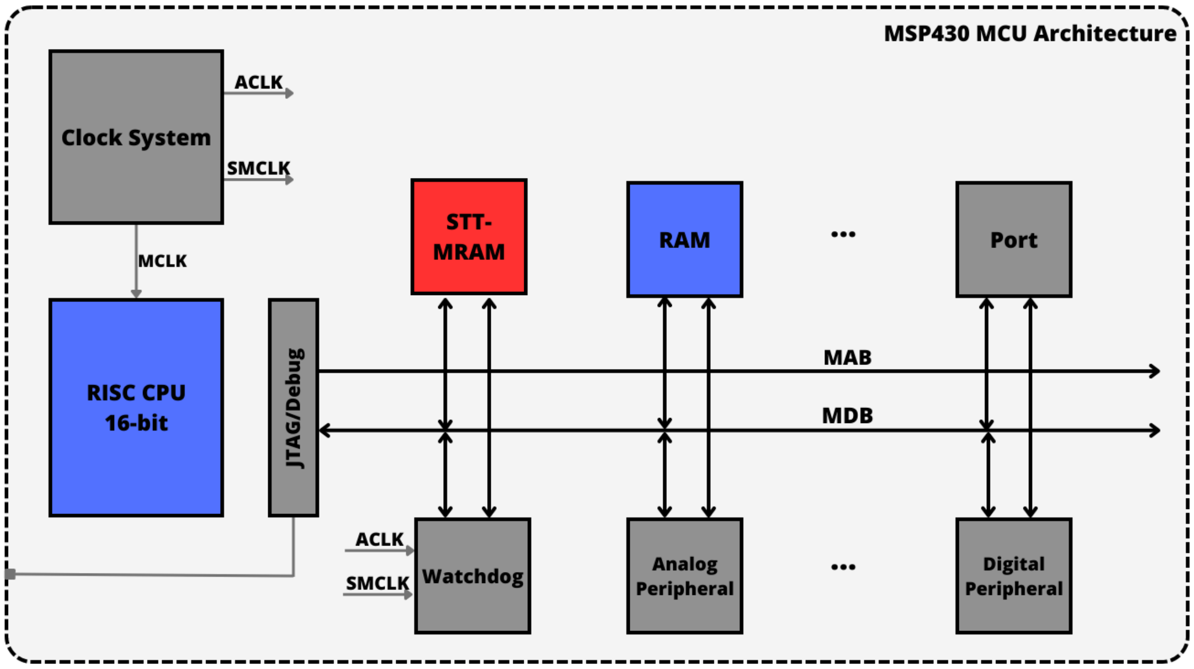


Figure 4.1: The MSP430 with STT-MRAM as NVM architecture.

4.2. Evaluation Process

In this section, we describe the evaluation process we utilize to compute the effectiveness of utilizing the approximation technique we describe above to reduce the *total energy consumption* of a program running in the platform described in Section 4.1. To properly describe the evaluation process, we need first to introduce two simulators we utilize: MSPSim and NVSim, which we will describe in Sections 4.2.1 and 4.2.2, respectively. After this, in Section 4.2.3, we describe the whole workflow, from the inputs to the final evaluation outputs.

4.2.1. MSPSim

MSP430 MCUs are widely used in various applications, including building and home automation [46], factory control and automation, and health and medical [47] applications. However, software development for MSP430 can be complex, as it often involves real-time interfacing with sensors, actuators, and other peripherals. For this reason, testing and debugging become challenging in the development process.

MSPSim is an emulator for the MSP430 MCU that accurately simulates the behavior of the MSP430 core, RAM, and peripherals, allowing us to execute and debug programs in a simulated environment without needing the physical hardware. In addition, the emulator supports various MSP430 device families. One of the most relevant features of MSPSim is its ability to simulate the cycles of MSP430's core. MSPSim emulates the timing and behavior of the core, including its instruction set, registers, and memory operations. MSPSim allows us to run our code and accurately measure performance as we would on the physical hardware.

MSPSim does not emulate any NVM. We modify MSPSim to allow us to consider a part of the address space as it is STT-MRAM. This portion of the address space is further divided into sections, with each section assigned an approximation level. The approximation level determines the probability of errors occurring when values are written to that section. Higher approximation levels result in more errors. In this way, we can simulate the approximation technique for STT-MRAMs described in Section 4.1.1. When running a program using MSPSim, we can choose which data needs to be saved in the simulated STT-MRAM portion of the address space and in which section with what approximation level the data should be saved.

4.2.2. NVSim

NVSim is a simulator that allows us to evaluate the cost and performance characteristics of various NVM technologies, such as flash memory, phase-change memory, and STT-MRAM.

NVSim simulates the behavior of NVM technologies under different conditions, including varying temperature, voltage, and current for write/read operations. NVSim outputs the latency, endurance, area, leakage power, and energy consumption of the modelled NVM technology, which are essential for evaluating the design of NVM technologies. In particular, we rely on NVSim to estimate the energy consumption of NVM technologies.

We utilize NVSim to characterize our STT-MRAM. To fully understand the behavior of the memory array, it is necessary to accurately characterize the electrical properties of each cell, such as the current required for set and reset operations and the voltage needed for reading the cell value. This information can vary depending on the cell's specific design and can significantly impact performance, reliability, and energy consumption.

4.2.3. Workflow

This evaluation workflow, represented in Figure 4.2, consists of two parts that converge to determine the optimal approximation level for a given program.

On one side, we provide the source code (.c file) of the program as input to the MSP430-

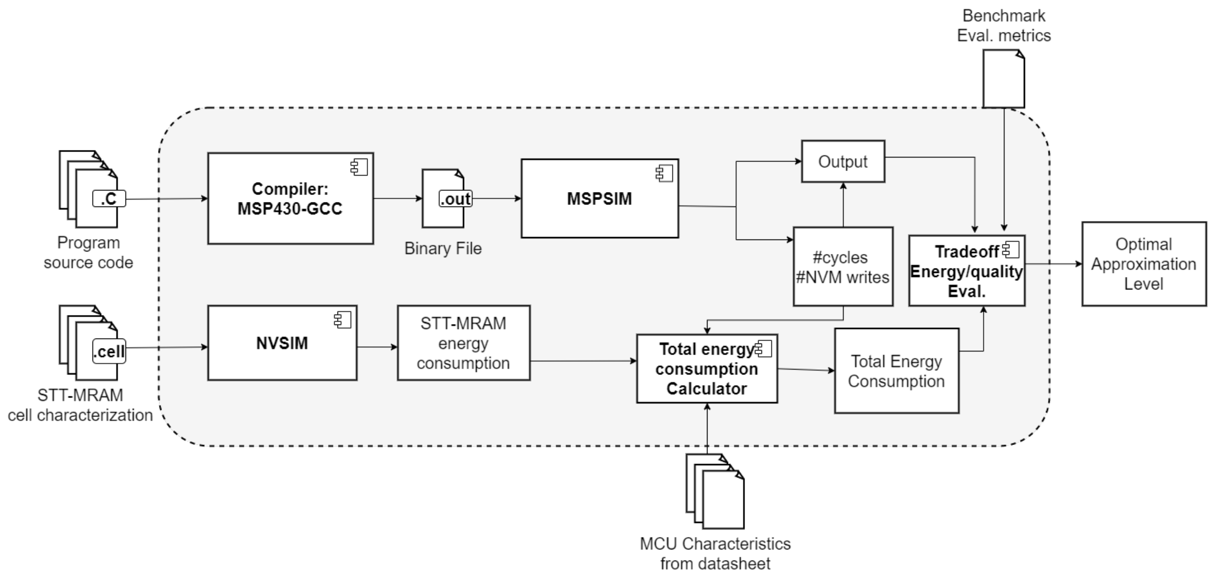


Figure 4.2: The evaluation pipeline, from the source code and STT-MRAM cell definition to the energy/quality tradeoff.

GCC compiler, which generates an executable file (.out file). We then feed this .out file to the MSPSIM simulator, which runs the program with different levels of approximation. For each program, we produce two outputs:

1. A file containing the number of instruction executions it takes to complete the execution of the program, as well as the number of NVM accesses the program performs.
2. A file containing the program's output. The output changes if we change the approximation level since a different amount of errors will be introduced.

On the other side of the process, we have characterizations, one for each approximation level, of the STT-MRAM cell in .cell files, which we provide as an input to NVSim. This software calculates and outputs the energy consumption of the cell at different approximation levels.

To determine the total energy consumed by a program for accessing the NVM, we combine the energy consumption values obtained from NVSim with the number of NVM accesses calculated earlier.

Additionally, the total energy consumed to run the program includes not only the energy consumed for accessing the NVM but also the energy consumed by the MCU to execute all instructions. This total energy consumption can be calculated using the number of instruction executions determined earlier and the energy consumption of running the

MCU in active mode, as shown in Table 4.2.

By comparing the energy consumption of each approximation level with the quality of the program output, we can determine the tradeoff between Quality of Results (QoR) and energy savings. The quality metric used to compute this difference in QoR is application specific. This tradeoff allows us to find the optimal approximation level for the given program. It also allows us to evaluate the effectiveness of our approach.

4.3. Benchmarks

There is no standard benchmark selection for evaluating intermittent computing systems, which have strict constraints on energy, memory, and computation resources. Other works in literature [11] use benchmarks from the MiBench2 suite [12] for evaluating intermittent systems. Mibench2 is a suite of benchmarks designed for embedded systems and covers a wide range of application domains such as automotive, consumer electronics, network, security, telecommunications, and office.

For our evaluation, we also select benchmarks from the MiBench2 suite. We chose a heterogeneous selection of benchmarks to cover various aspects relevant to the evaluation. Specifically, we considered the following aspects for each benchmark:

- Whether the benchmark is amenable to approximation or it requires 100% accurate computing.
- Whether the benchmark is a pipeline of subtasks or a single task. Some benchmarks are intrinsically built as a sequence of subtasks, with the output of each subtask becoming the input of the next one. With this type of benchmark, we can force the output of each subtask to be written in NVM instead of in volatile memory. This allows us to simulate a task-based intermittent computing system.

In the approximated execution, some errors are introduced in the written data. This leads to a decrease in the QoR compared to the output of the correct execution. To evaluate this decrease in the QoR, we need to compare the output obtained from the correct execution with the one obtained from the approximated execution.

We select a specific quality metric for each benchmark to quantify this difference in QoR. The choice of the quality metric is specific to the benchmark. By comparing the output obtained from the approximated execution with that obtained from the correct execution using the selected quality metric, we can evaluate the impact of the introduced errors on the QoR of the benchmark's output.

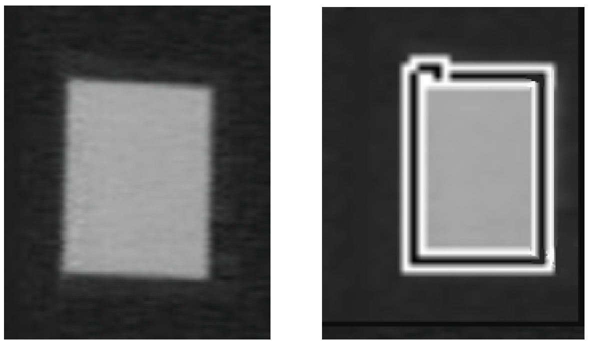

(a) Input (b) Output

Figure 4.3: Susan input and output, execution without approximation.

4.3.1. Susan Edge Detection

Susan is an image recognition and processing algorithm that allows the recognition of corners and edges on images. It detects the edges in an image based on the differences in pixel intensity between a central pixel and the one surrounding it. To do so, it places a circular mask of pixels centered around each pixel in the image. At this point, for each mask, it compares the intensity of the pixels in the mask with the central one. The pixels that have similar intensity values are part of the same segment.

Susan edge detection is structured as a pipeline of sub-tasks, where the output of a sub-task is fed as input of a successive one. So errors will affect the final QoR differently depending on the stage they are introduced. This program is of particular interest to us since it will allow us to show the errors snowball effect we discussed in Section 3.2.2.

Susan takes as input images in grayscale pgm format. For our environment with limited availability of RAM, the images need to be resized to fit our memory constraints. Therefore, its output is always a grayscale pgm image, where the identified edges are represented as black pixels surrounded by white ones. In Figure 4.3, we can see an example of the input and output of the Susan edge detection algorithm.

Quality Metric

In Susan Edge detection, we evaluate the quality of the approximated results by comparing them with the correct execution results. The evaluation metrics used are precision and recall, calculated with a tolerance value δ. For example, suppose a pixel detected in the approximated execution is not in the same position as in the correct execution. In that case, we can still consider it correctly detected if it is very close to the correct position. The δ value represents the maximum distance from the correct position we allow to consider a pixel correctly detected. A larger δ value means we allow more deviation from the correct position and still consider the pixel correctly detected. In contrast, a smaller δ value requires the detected pixel to be very close to the correct position to be considered as correctly detected.

We can so define:

- True positives (TP): pixels that were correctly detected as part of an edge both in the approximated execution and in the correct execution, or around the pixel with a distance less than or equal to δ.
- False positives (FP): pixels that are detected as part of the edge in the approximated execution but were not detected as part of the edge in the correct execution or around the pixel plus the δ.
- False negatives (FN): pixels that were part of the edge in the correct execution but were not detected as part of the edge in the approximated execution or around the pixel plus the δ.
- True negatives (TN): pixels that were not part of the edge in the correct execution and were also not detected as part of the edge in the approximated execution or around the pixel plus the δ.

At this point we calculate precision and recall with these formulas:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{4.1}$$

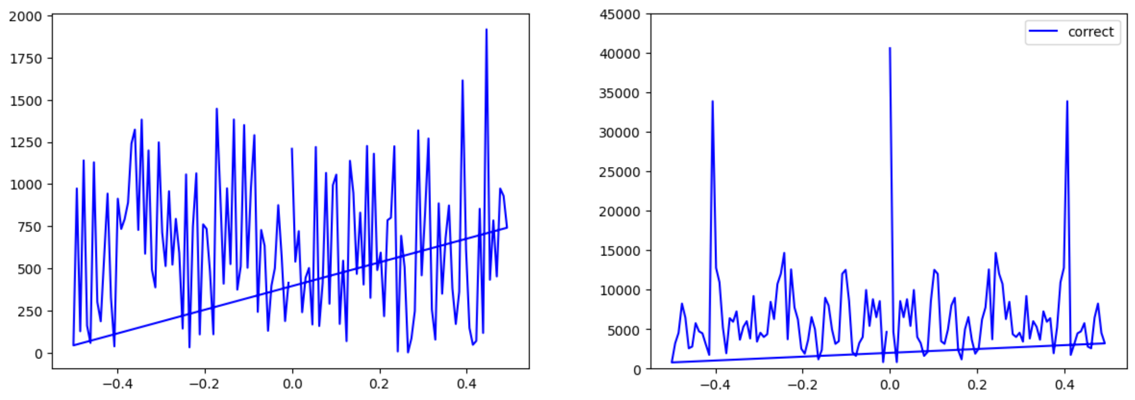
$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{4.2}$$

4.3.2. FFT

The second benchmark we consider is Fast Fourier Transform (FFT). This benchmark computes the discrete Fourier Transform of a sequence of complex numbers. Discrete Fourier Transform computation is widely used in many domains, like signal processing, communications, digital recording, and sampling.

The input this benchmark expects is a sequence of complex numbers representing the signal in its original domain, such as time or space, and outputs a sequence of the same length of complex numbers in the frequency domain. An example is provided in Figure 4.4.

This benchmark can be seen as a single task since once an output is generated, it is not



(a) Random input (b) Output

Figure 4.4: FFT input and output, execution without approximation.

used as input by successive computations, so it does not affect the rest of the execution.

Quality Metric

The evaluation metric we selected to evaluate the performance of an approximated execution FFT with respect to a fully correct one is Average Relative Error (ARE). This metric is the most widely used in literature for the FFT benchmark [48].

$$\text{AvgRelativeError} = \frac{1}{n}\sum_{i=0}^{n-1}\left|\frac{x[i]^{correct} - x[i]^{approx}}{x[i]^{correct}}\right| \tag{4.3}$$

With n being the length of the input sequence, $x[i]^{correct}$ is the ith element of the output sequence from the correct execution of FFT, and $x[i]^{approx}$ is the ith element of the output sequence from the approximated execution of FFT.

4.3.3. PicoJPEG

PicoJPEG is a benchmark from MiBench2. It decompresses an image in the format of JPEG and outputs the decoded image in bitmap format. PicoJPEG is a lightweight image-decoding algorithm, which makes it well-suited for intermittent computing systems that have limited computing capabilities.

We selected this benchmark for evaluating our approach since it produces a very large output. For example, for a 64x64 image, the bytes we will write in NVM are 3x64x64. So we can evaluate the potential energy savings in a program that makes intense use of NVM.

Quality Metric

Being PicoJPEG a program that produces an image as output, we decide to use Root Mean Squared Error (RMSE) as the evaluation metric to compare the approximated execution with the correct one as suggested in other works in literature [48]. RMSE calculates the average root-mean-square of the pixel differences of the precise and approximate outputs.

The formula for the computing of RMSE is:

$$\text{RMSE} = \sqrt{\sum_{n=0}^{width*height} \frac{(CorrImage[n] - ApprImage[n])^2}{(width * height)}} \tag{4.4}$$

With $CorrImage[i]$ being the ith pixel of the flattened image computed with a fully correct execution and $ApprImage[j]$ being the jth pixel of the flattened image computed with an approximation execution.

4.3.4. Only NVM Writes Micro-Benchmark

This benchmark is not a part of the MiBench2 suite. It is a simple benchmark that solely writes an array of values in NVM. This benchmark is important because it is the same as a checkpointing routine where the system only writes in NVM and performs no other operations. This benchmark represents the corner case where there is no additional workload except for the one required to write in NVM. By evaluating the energy savings achieved with this benchmark, we can measure how much potential for energy savings we have with a given platform. This is a best-case scenario for evaluating the effectiveness of our approximation technique since the energy savings for the other benchmarks that also have an additional workload will surely be smaller than the ones of this micro-benchmark.

In this chapter, we presented the framework, the tools, the workflow, and the benchmark we used for generating the experimental results we show in the next chapter.

5 | Experimental Evaluation

This chapter presents the experimental results obtained using the methodology described in the previous chapter. In Section 5.1, we describe the parameters that determine the effectiveness of our Approximate Computing (AC) technique and the results on the energy consumption of STT-MRAM at different approximation levels. Section 5.2 shows the energy savings we can obtain with our AC technique for various benchmarks. In Section 5.3, we show the tradeoff between energy and quality for each benchmark.

5.1. System Parameters

To evaluate the efficacy of our AC technique, we need to consider various parameters. Consider for simplicity a single energy burst, as shown in Figure 5.1. With our AC technique, we can reduce $\mathrm{P}_{\mathrm{nvm}}$, thus reducing the energy consumed by Non Volatile Memory (NVM) writes, calculated with $E_{\mathrm{nvm}} = P_{\mathrm{nvm}} \times T_{\mathrm{nvm}}$. We can then call $\mathrm{E}_{\mathrm{computation}}$ the energy consumed for the computation in that burst. It is calculated as $E_{\mathrm{computation}} = T_{\mathrm{computation}} \times P_{\mathrm{computation}}$. Let us call the total amount of energy computed in a burst $E_{\mathrm{total}} = E_{\mathrm{nvm}} + E_{\mathrm{computation}}$.

The performance of our approach depends on the impact of E_{nvm} on E_{total}. If E_{nvm} has

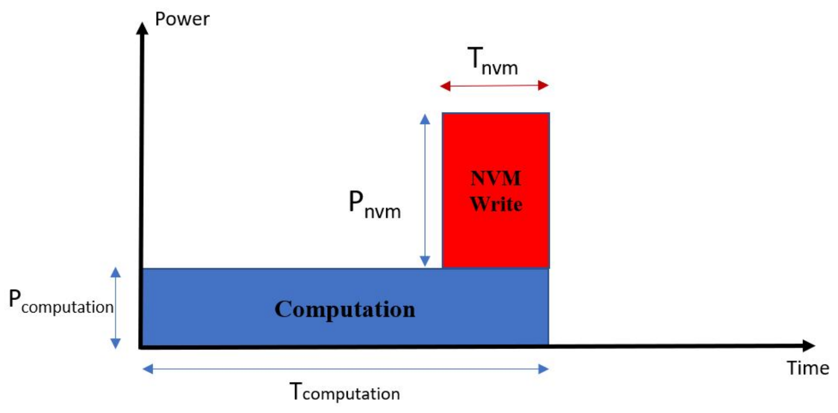


Figure 5.1: The variables to consider for evaluating the performance of our approach.

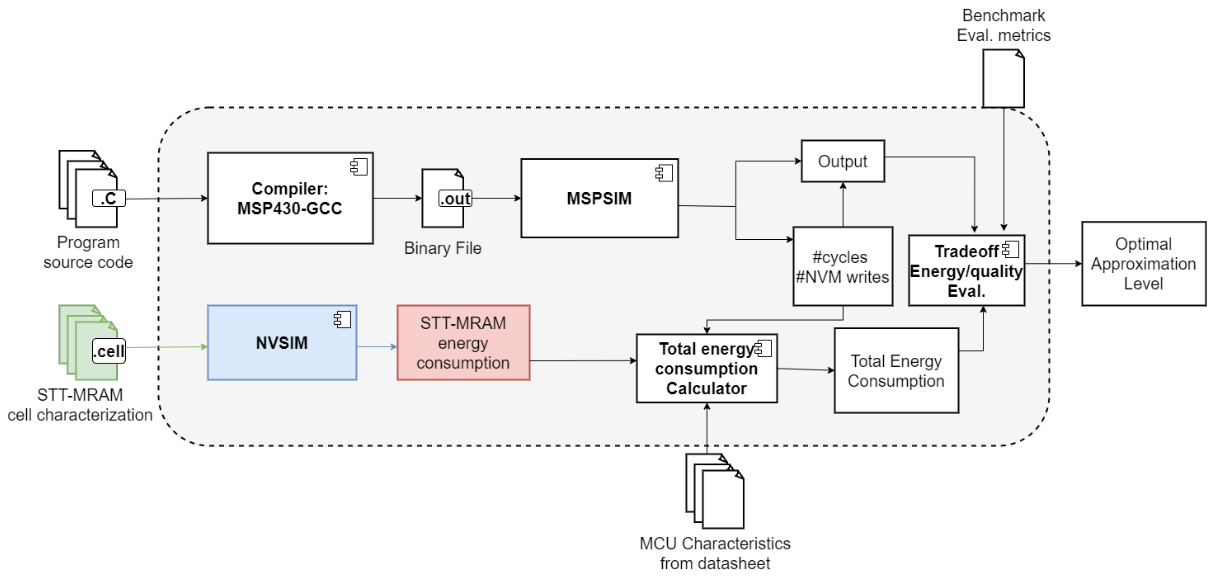

Benchmark Eval. metrics
.C
Program source code
Compiler: MSP430-GCC
.out
Binary File
MSPSIM
Output
#cycles #NVM writes
Tradeoff Energy/quality Eval.
Optimal Approximation Level
.cell
STT-MRAM cell characterization
NVSIM
STT-MRAM energy consumption
Total energy consumption Calculator
Total Energy Consumption
MCU Characteristics from datasheet

Figure 5.2: Evaluating process, highlight on how we compute STT-MRAM energy consumption. Green is input, blu intermediary steps, red output.

a negligible impact on E_{total}, i.e., $E_{\text{nvm}} << E_{\text{computation}}$, then minimizing E_{nvm} results in minimal energy savings.

Now, let us discuss where the values of these variables come from:

- $\mathbf{P_{computation}}$ depends solely on the microcontroller (MCU). In Table 4.2, we show the values of $\text{P}_{\text{computation}}$ for the MCUs we consider in our evaluation.
- $\mathbf{P_{nvm}}$ depends on the characteristics of the STT-MRAM. This is the value that we can reduce using our AC technique.
- $\mathbf{T_{nvm}}$ depends on the benchmark and the amount of NVM utilized by the benchmark.
- $\mathbf{T_{computation}}$ depends on the benchmark and the amount of computation required to calculate the benchmark output.

STT-MRAM Characterization

We now present the experimental results on the energy consumed when writing a bit in STT-MRAM and how much this value can be reduced with our AC technique. These results are key for computing the overhead of writing in NVM and how much we can shrink this value. The results we obtain here are the heights of the NVM write area in Figure 5.1 for different approximation levels.

We define five quality levels, with Q0 being the baseline representing the quality that

Quality	WER	Set Current (μA)	Write Energy per bit (pJ)	% Energy consumption normalized to Q0
Q0	10^{-8}	1153	167	100%
Q1	10^{-6}	865	94	56.3%
Q2	10^{-5}	769	74	44.3%
Q3	10^{-4}	673	57	34.1%
Q4	10^{-3}	577	43	25.7%

Table 5.1: WER-energy relationship for a 32nm STT-MRAM. Energy consumption are reported for a single bit.

ensures nearly 100% correct writes, with a Write Error Rate (WER) of 10^{-8}, meaning one bit switch failure every 10^{8} bit switches. Q1 is the first approximated level, with a WER of 10^{-6}, and Q4 is the most approximated level, with a WER of 10^{-3}.

We compute the energy usage for writing a bit in an STT-MRAM for each quality level. The WER of STT-MRAM cells depends on the value of the set current. To determine the set current for each quality level, we referred to a study by Monazzah et al. [6] that presented the relationship between the set current and the WER in STT-MRAM memories. In Figure 5.2, we highlight the specific part of our evaluation process that enables us to calculate the energy consumption for writing in STT-MRAM. To perform this calculation, we require ".cell" files that contain the characterization of an STT-MRAM cell, such as the set current, cell form factor, read voltage, and others. We created a unique ".cell" file for each quality level, in which we adjusted the set current based on this WER and set current relationship. We then feed these ".cell" files to NVSim which outputs the energy consumption values for the write operation of the STT-MRAM for each quality level.

Table 5.1 summarizes the energy consumption results, showing that even with the first level of approximation, the energy saved is high, being 56.3% of the energy consumed with Q0. The energy consumption decreases further with stronger approximation, with Q4 consuming only 25.7% of the energy of Q0. It is important to point out that the decrease in energy consumption does not follow a linear pattern, with the reduction being more significant in the initial levels of approximation. However, this reduction becomes less pronounced with the strongest quality levels.

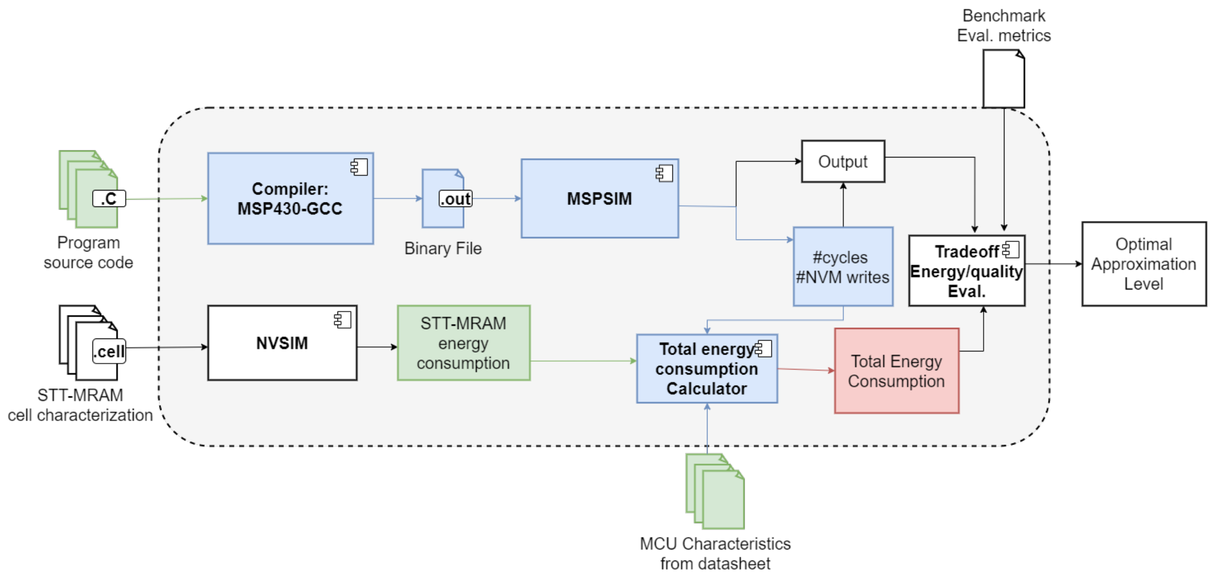

Benchmark Eval. metrics
.C
Program source code
Compiler: MSP430-GCC
.out
Binary File
MSPSIM
Output
#cycles #NVM writes
Tradeoff Energy/quality Eval.
Optimal Approximation Level
.cell
STT-MRAM cell characterization
NVSIM
STT-MRAM energy consumption
Total energy consumption Calculator
Total Energy Consumption
MCU Characteristics from datasheet

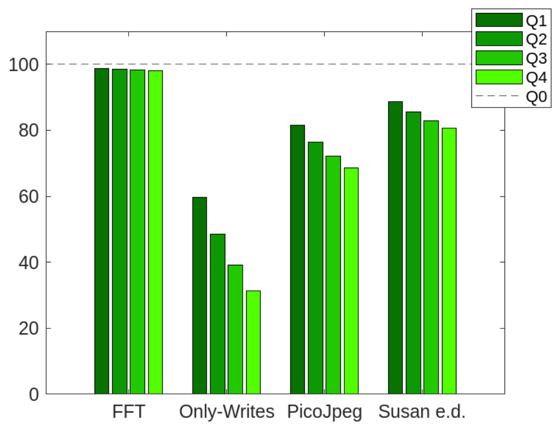
Figure 5.3: Evaluating process, highlight on how we compute the total energy consumption. Green is input, blu intermediary steps, red output.

5.2. Total Energy Consumption

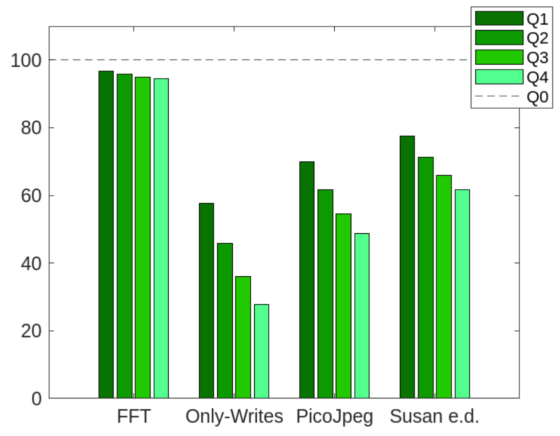
In this section, we show the energy savings obtainable using our AC technique, so how much we can reduce E_{total}. We then explain what the effectiveness of the proposed AC technique depends on.

Figure 5.3 highlights the portion of our evaluation process to compute the total energy consumed by every benchmark with our AC technique at different quality levels. We show that the input this portion takes are the source codes of the programs, the energy consumptions of the STT-MRAM at different quality levels we computed in the previous section, and the MCUs characteristics of energy consumption we show in Table 4.2. The produced output is the total energy consumed for running the program. We call this value E_{total}. This is a big portion of the evaluation process; for this reason we first show the results that we obtain in terms of energy savings thanks to our AC technique. After this, we explain how the intermediary values affect these results.

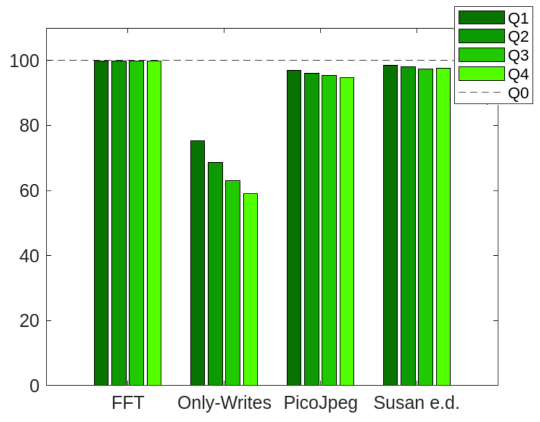
Figure 5.3 displays the total energy consumed for each benchmark, normalized to the total energy consumed by the benchmark executed with quality level Q0. We show that the reduction in the energy consumed varies dramatically between benchmarks and MCUs. The energy savings are significant for almost all the benchmarks for a very energy efficient MCU like the MSP430Singhal. However, for the MSP430G, which has more demanding energy requirements, the energy saved is almost always negligible, regardless of the bench-



(a) MSP430L



(b) MSP430Singhal



(c) MSP430G

Figure 5.4: E_{total} with different levels of approximation normalized to E_{total} with Q0, in different MCUs for various benchmarks.

mark or the approximation level.

We now focus on the MSP430Singhal, as it provides the best results for our approximation technique. For the Only-writes microbenchmark, which does nothing but write an array of values in NVM and represents the corner case for our approximation evaluation, the energy reduction with the maximum approximation level Q4 is significant, being it only 30% the energy consumed with Q0. Looking at Figure 5.3, we can see that the PicoJpeg yields the highest energy savings among all the other benchmarks from MiBench2. With an approximation level of Q4, the energy consumed is only half of the energy consumed when running the benchmark with Q0. However, in FFT, the energy consumption reduction is almost negligible, regardless of the quality level. In Susan edge detection, we observe significant energy savings, with the energy consumed with the Q4 quality level being only 0.62 times the energy consumed in the correct execution.

Benchmark	Active cycles	NVM writes (bit)
Only-writes	8×10^3	32×10^3
FFT	48×10^6	420×10^3
Pico-jpeg	3638×10^3	848×10^3
Susan Edge	9500×10^3	1107×10^3

Table 5.2: Average number of active cycles and NVM writes to complete each benchmark.

Looking at the results, we note that energy consumption does not decrease linearly while increasing the approximation level. So, energy consumption decreases more significantly from Q1 to Q2 than from Q3 to Q4. For this reason, pushing the approximation level too far could be less effective.

As we explained in Section 5.1, the effectiveness of our AC technique depends on how much E_{nvm} impacts on E_{total}. We define $E_{ratio} = (\frac{E_{nvm}}{E_{total}}) \times 100$, this is the effectiveness index that represents the effectiveness of our AC technique. We compute this index for FFT, PicoJpeg, Susan edge detection and Only-writes. E_{ratio} also depends on the MCU utilized to run the benchmark, so we will show results for all the MCUs described in Table 4.2.

It is important to note that the number of active cycles it requires the MCUs to complete the benchmark, and the number of accesses in NVM, do not change if we run the benchmark with approximate execution or with a correct execution nor it changes for different MCUs. This is because all the considered MCUs have the same core and instruction set architecture, and the inputs we use for each benchmark are the same. Moreover, we run all the benchmarks with the MCUs set at the same clock. So, while $P_{computation}$ changes for different MCUs, $T_{computation}$ and T_{nvm} are independent from the MCU.

We run each benchmark hundreds of times, every time with a different input. We count the number of active cycles and NVM writes it takes to complete each run and then do an average of these values for each benchmark. This allows us to find, for each benchmark, the average number of active cycles it takes to complete it and the average number of NVM writes it performs. Our approximation technique may be used at any point in time, but we cannot know the state of the memory at that moment. To simulate a scenario "a regime", before each run of the benchmarks, we randomly initialize the simulator's memory. If we do not do so, MSPSim will initialize all the memory with zeros, this is not a realistic case, and it could affect the obtained results. The results we get are summarized in Table 5.2.

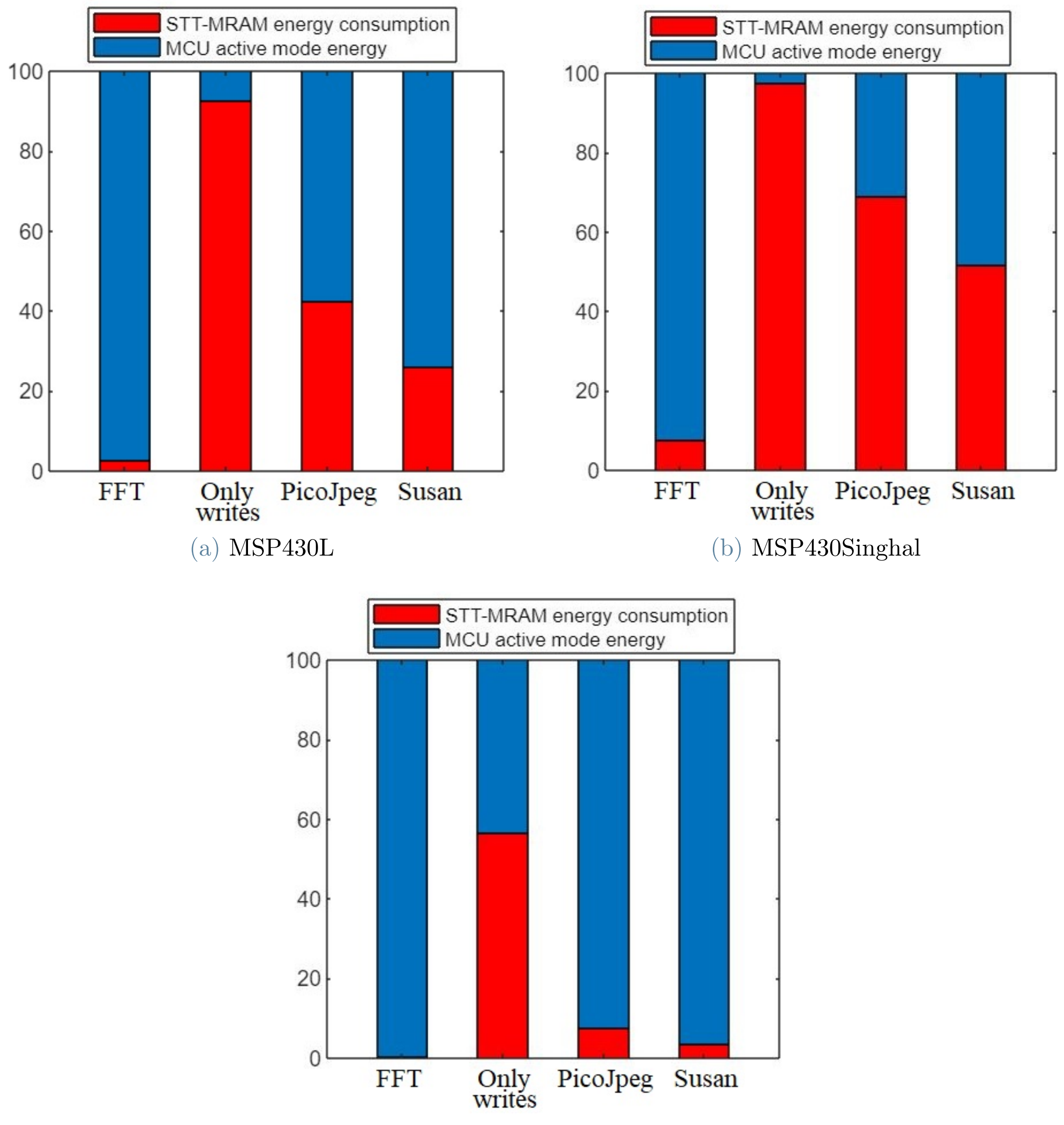


Figure 5.5: E_{nvm} impact on E_{total} for different benchmarks in various MCUs.

We now combine these values with the active energy power for each considered MCU with the energy cost to perform a single write in STT-MRAM with the baseline Q0 calculated in Section 5.1. This procedure allows us to find, in the case of STT-MRAM writes performed without approximation, the value of $E_{\text{nvm}} = \#NVMwrites \times E_{\text{Q0}}$, with E_{Q0} being the energy to write a bit in STT-MRAM with quality level Q0. We can also compute $E_{\text{computation}}$ by knowing the number of active cycles shown in Table 5.2 and the active power of the MCU shown in Table 4.2. Now it is easy to compute the index of effectiveness of our AC technique: $E_{\text{ratio}} = \frac{E_{\text{nvm}}}{E_{\text{nvm}}+E_{\text{computation}}}$. Figure 5.5 shows how much the writes in NVM impact the total energy consumption for running each benchmark.

To evaluate the effectiveness of our approach, we first analyze how E_{ratio} changes when different MCUs are used. As expected, the effectiveness of our approach is heavily influenced by the active mode energy consumption of the MCU being used. To illustrate this, we compared two MCUs: MSP430Singhal in Figure 5.5b and MSP430G in Figure 5.5c. The first is the most energy efficient, while MSP430G is the least efficient. We found that for the PicoJpeg benchmark, the E_{ratio} was 69% for MSP430Singhal and only 7% for MSP430G, resulting in a difference of almost 10x. For the Only-writes micro-benchmark, where the $T_{\text{computation}}$ part of the execution is the smallest possible, the E_{ratio} was 97% for MSP430Singhal and 56% for MSP430G, a difference of 2x. We conclude that for MCUs with energy consumption in active mode similar to that of the MSP430G, our AC technique does not give the possibility for significant energy savings. At the same time, for MSP430Singhal and MSP430L, depending on the benchmark, we can save a considerable amount of energy.

We now focus on MSP430Singhal, which benefits the most from our AC technique. As we expect, the benchmarks that involve image processing offer more opportunities for our approach. PicoJpeg, produces big outputs to be saved in NVM while having a relatively small computation amount. The energy consumed by writes in NVM accounts for 69% the total energy consumption of running the benchmark. For this reason, the amount of energy we can save by reducing the cost of writes in NVM is significant.

Susan edge detection is the second most promising program for our AC technique, with $E_{\text{ratio}} = 51\%$. The outputs of Susan are way smaller than the ones of PicoJpeg, being its results grayscale images versus rgb images of PicoJpeg, so almost one-third the dimension. However, their E_{ratio} is not that different because Susan is intrinsically a benchmark composed of more subtasks. While in PicoJpeg, the approximation is made only at the end when the result is written in NVM, in Susan, writes in NVM are also performed as intermediate results, where a write in NVM in a subtask of Susan is then used as an input in a following subtask, as we explained in 4.3.1. However, for this reason, the behavior of approximated Susan's results follow a more complex pattern than that of PicoJpeg. We discuss this in detail in Section 5.3.2.

At last, we show that FFT is the benchmark that gives fewer opportunities for our AC technique, with writes in NVM accounting for only the 7% of the total energy consumed to run the benchmark. This is because its output is pretty small, while the computational part that cannot be approximated is relatively long. This makes this type of program not convenient for applying our approximation technique.

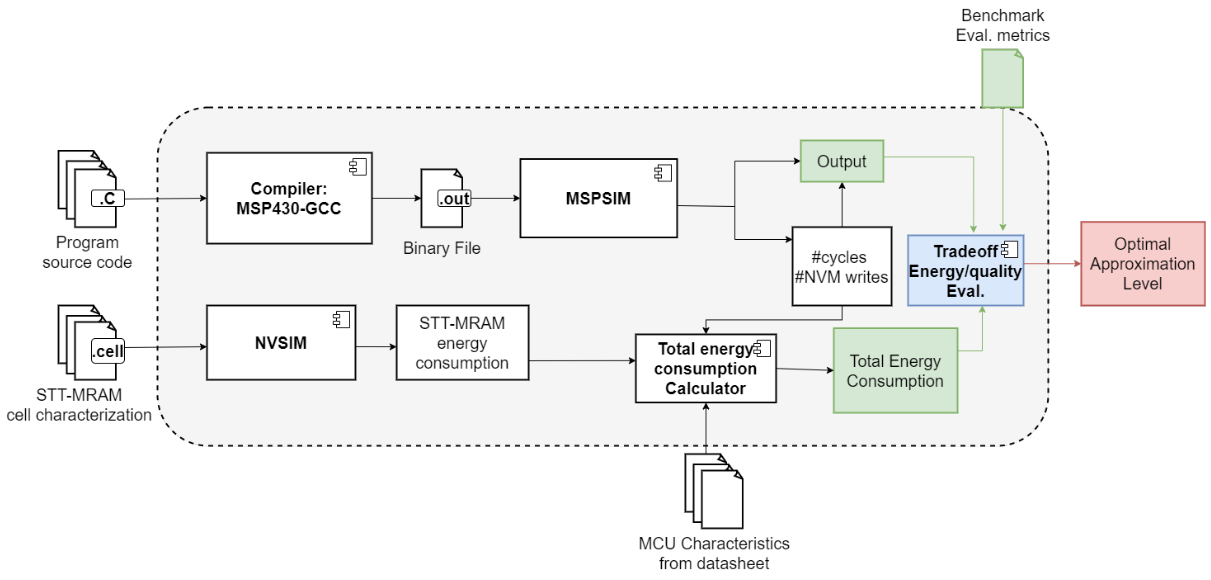

Benchmark Eval. metrics
.C
Program source code
Compiler: MSP430-GCC
.out
Binary File
MSPSIM
Output
#cycles #NVM writes
Tradeoff Energy/quality Eval.
Optimal Approximation Level
.cell
STT-MRAM cell characterization
NVSIM
STT-MRAM energy consumption
Total energy consumption Calculator
Total Energy Consumption
MCU Characteristics from datasheet

Figure 5.6: Evaluation process, highlight how we compute the tradeoffs. Green is input, blu intermediary steps, red output.

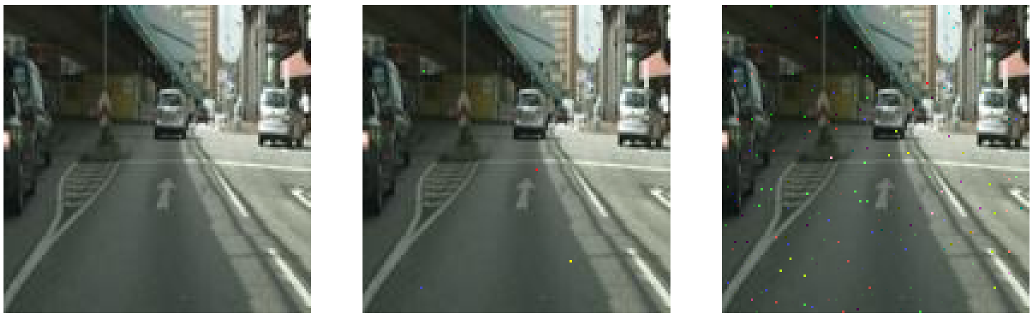
In the next section, our evaluation will include the deterioration of the Quality of Results (QoR) for each benchmark. This will help us determine the level of approximation that achieves a significant reduction in energy consumption without severely affecting the accuracy of the output.

5.3. Quality/Energy Trade-off

Until now, we only evaluated the potential energy savings of our AC technique, but we did not consider the degradation of the results it causes. In this section, we study the trade-off of energy saved vs QoR for each of the presented benchmarks. Figure 5.6 highlights the part of the evaluation process that allows us to define this relationship. The inputs for this evaluation are the results of the benchmarks, run with hundreds of different inputs at different levels of approximation, and the average E_{total} of each benchmark for each level of approximation. For each benchmark, we describe how the resulting QoR decreases and if we recognize some pattern in the nature of this quality degradation.

5.3.1. PicoJpeg

In Figure 5.7, we show how our AC technique affects the outputs of PicoJpeg. This benchmark can be seen as a single task, where the approximation happens only at the end when writing the result in NVM. This can be seen as an approximate saving of an image.



(a) Q0 (b) Q1 (c) Q4

Figure 5.7: PicoJpeg output at various levels of approximation.

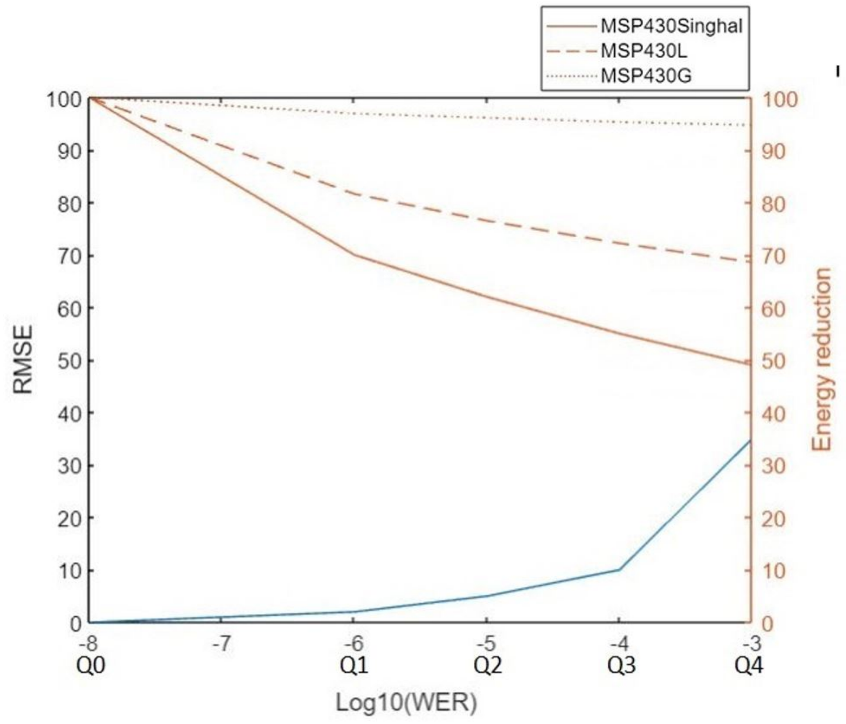


Figure 5.8: PicoJeg trade-off.

When saving the image in STT-MRAM, some pixels at random positions will be incorrect when a bit fails to switch. The only difference when increasing the approximation level and the number of bit switch failures is that we introduce more wrong pixels in the image. There is no correlation in the position of failed pixels, but they are spread randomly in the image. We call this scheme of errors "Salt & Pepper". Similar results are obtained by Zeinali et al.- [49] that also used STT-MRAM as memory to save images.

Figure 5.8 demonstrates this benchmark's energy/quality trade-off, showing how the Root Mean Squared Error (RMSE) compared to the result of a correct execution increases as

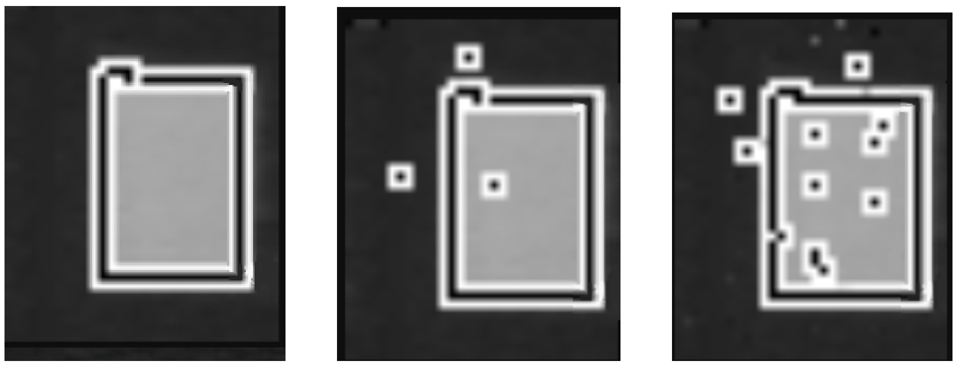

(a) Q0 (b) Q1 (c) Q4

Figure 5.9: Susan edge detection output at various levels of approximation.

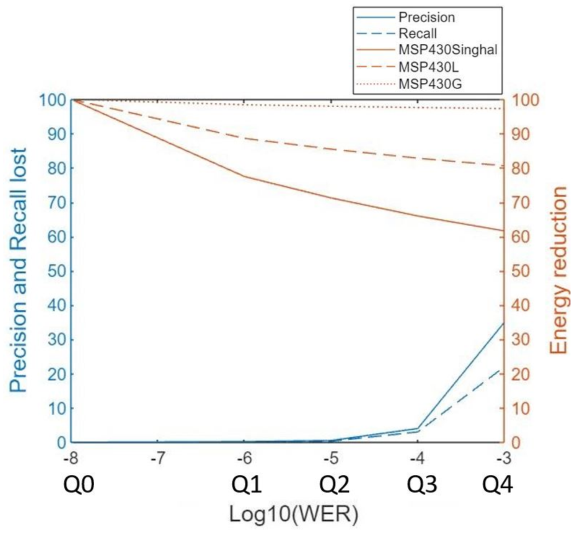
the approximation level changes. The RMSE almost linearly increases from Q1 to Q3 and experiences a significant increase when moving to Q4, with the RMSE increasing from 9.8 at Q3 to 35 at Q4. On the other hand, the decrease in energy consumption, shown on the right, is more prominent for the first levels of approximation and mitigates in the last ones.

In summary, we observe that for Q3, the RMSE is only 9.8. With that approximation level, we can achieve an energy reduction of 45% and 27.8% of the total energy consumed by the benchmark with an MSP430Singhal and MSP430L MCU, respectively. At the cost of a small degradation in quality, we can achieve significant energy savings when running this program.

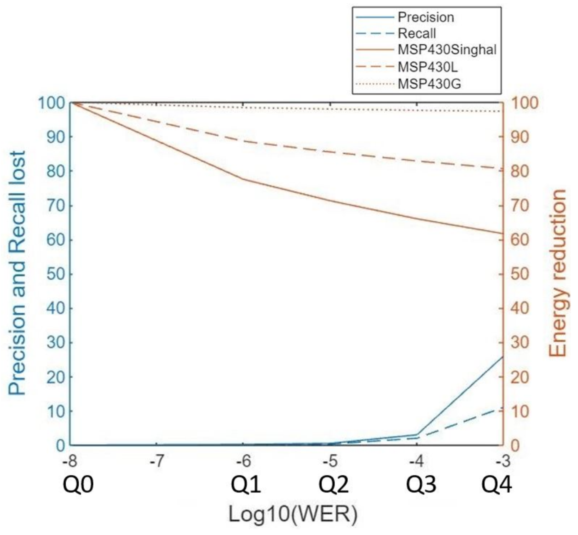
5.3.2. Susan Edge Detection

In Figure 5.9, we present an example of the results of the Susan edge detection benchmark with varying levels of approximation. Although this is also an image-processing algorithm, the errors introduced in the results are not limited to Salt & Pepper errors. The Susan algorithm consists of a pipeline of tasks, and errors introduced in the initial stages of the pipeline may impact the subsequent computations, leading to more complex errors. This observation is significant as it is equivalent to a task-based intermittent computing. The Susan edge detection benchmark is divided into various well-defined stages. Every stage ends with a write in NVM. This value is then used as input for the next stage.

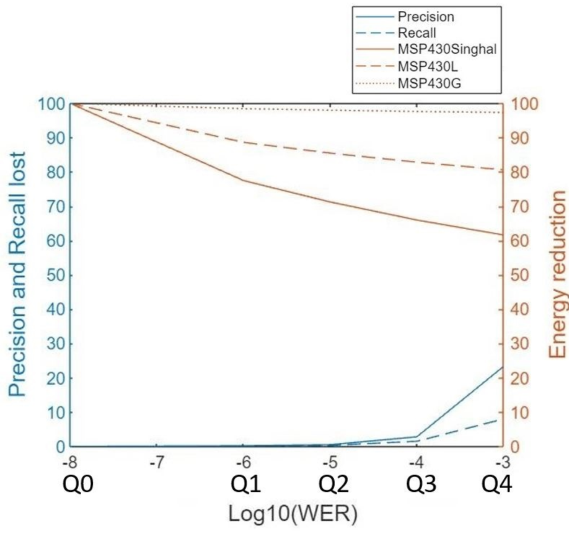
In the final stage, when the image is saved in NVM, we still introduce some Salt & Pepper errors, similar to the PicoJpeg benchmark. However, these errors are hardly noticeable in grayscale images. As the approximation level increases, the number of incorrectly detected edges increases, and we observe a trend of false positives being introduced.



(a) $\delta = 0$



(b) $\delta = 1$



(c) $\delta = 2$

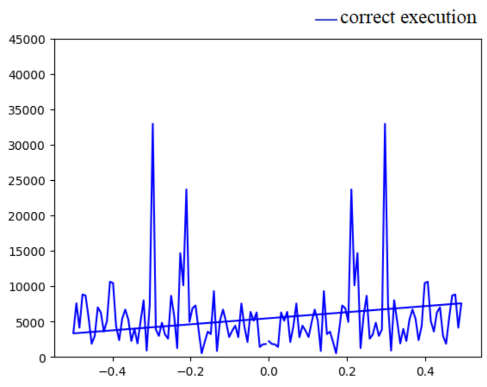
Figure 5.10: Susan edge detection trade-off. Precision and recall shown for different values of tolerance δ

However, true positives are generally detected even with high levels of approximation. These observations are further supported by the results on precision and recall we show in Figure 5.10.

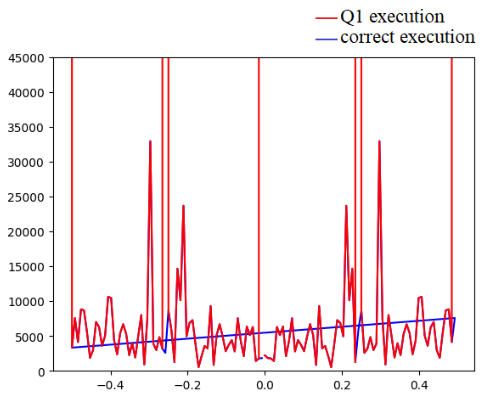
In Figure 5.10, we compare the decrease in the QoR of the approximated Susan outputs with the energy savings we can obtain with the approximation. We plot this tradeoff for $\delta = 0$ in Figure 5.10a, $\delta = 1$ in Figure 5.10b and $\delta = 2$ in Figure 5.10c. As we explain in 4.3.1, the delta is the tolerance, so we consider a pixel correctly detected if it is in the same position as the correct execution result or around the position as the correct result with a distance less than or equal to δ pixels. Increasing the tolerance, the precision and the recall values are higher.

We observe that also for this benchmark, the decrease in precision and recall is almost linear from Q0 to Q3, and it shows a jump for Q4. Q3 seems the most promising approximation level. We can note that with Q3 considering $\delta = 1$, we have a decrease in precision of only 4% and a decrease in recall of 2%, while reducing the energy consumption, in an MSP430Singhal MCU, of about 34% the original energy consumption of the benchmark.

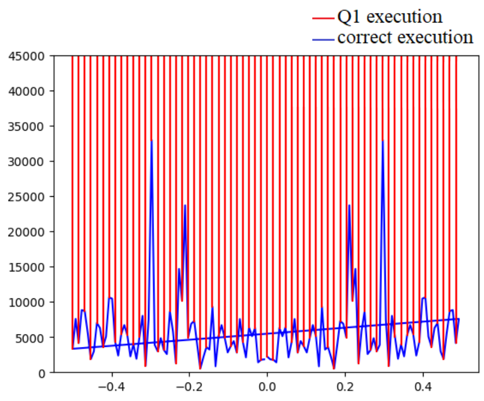
We can note also that moving from $\delta = 0$ to $\delta = 1$, the increase in the precision and recall for the approximated execution is bigger than when moving from $\delta = 1$ to $\delta = 2$. This indicates that the miss-detected pixels are often located near the actual edges. Hence, even if the approximated output is not entirely accurate, it still provides relevant information about the edges present in the image.



(a) Q0



(b) Q1



(c) Q4

Figure 5.11: FFT output at various levels of approximation.

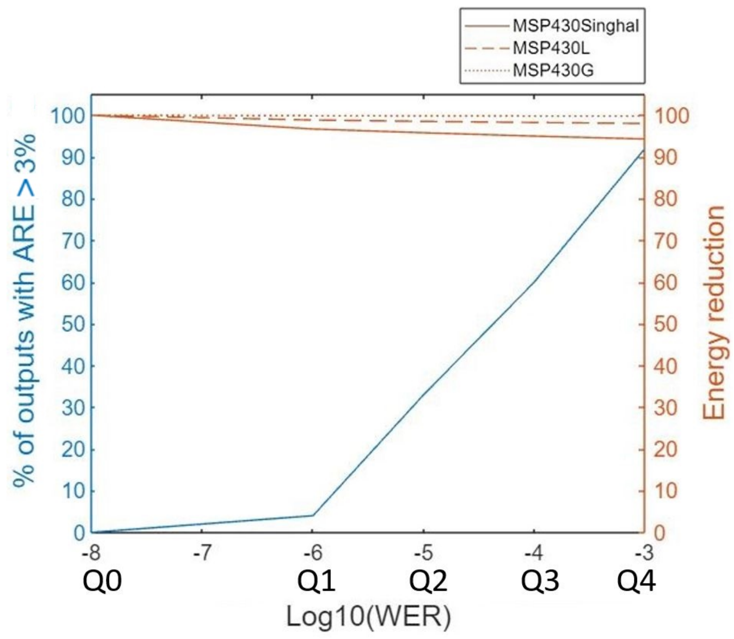


Figure 5.12: FFT trade-off.

5.3.3. FFT

In this section, we evaluate the trade-off for the Fast FFT benchmark. Figure 5.11 shows an example of the output at different levels of approximation. We observed that the obtained result follows the correct curve, but it introduces various spikes, and the more we increase the approximation level, the more spikes are introduced in the result. These spikes have a very high value, causing the Average Relative Error (ARE) to increase drastically. Since this benchmark can be seen as a single task, an introduced error in the final result in NVM does not affect the rest of the execution. For this reason, the type of errors we introduce here is again "Salt & Pepper", similar to the PicoJpeg benchmark, since the output signal manifests sparsely occurring spikes that have no relation one with the other. However, while this type of error in an image processing algorithm is not a big problem, in FFT they cause the output to be almost useless. For example, an image where 10% of randomly spread wrong pixels is still usable, but for a signal having 10% of the values at random spikes, the output becomes useless, even if in the rest of the output, the approximated curve follows the correct one. From this, we conclude that our AC technique is unsuitable for this type of program.

Figure 5.12 shows how energy consumption decreases with different approximation levels. In the graph, we evaluate the percentage of outputs with an ARE greater than 3%, which is a significant amount of error. We found that for Q1, only 1% of the obtained results have an ARE bigger than 3%, but this percentage increases to 35% at Q2. An FFT that produces a non-usable output for one-third of the times is useless. Thus, we can say that the maximum approximation level we can apply is Q1. Furthermore, we observed that even at the strongest approximation level, the energy savings for this benchmark are very low, with only 4.5% using the MSP430Singhal. In contrast, for the other MCUs, the energy saved is negligible. This observation also reinforces our conclusion that no benefits can be obtained by applying our AC technique for systems running this type of program.

In this chapter, we performed the experimental evaluation. we have shown the trade-offs for various benchmarks, considering various MCUs. In the next chapter, we draw conclusions with respect to the experimental evaluation carried out in this chapter and provide indications for future work.

6 | Conclusion and Future Works

Energy harvesting devices spare the use of bulky, hazardous, and definitely not-eco-friendly batteries. The drawback of this is that energy harvested from the environment is inconsistent, so it cannot be considered a stable power supply. For this reason, energy harvesting devices suffer frequent energy failures making computation intermittent.

To make the computation act as if it was continuous, these devices need to preserve their state across energy failures. This is possible by saving the state in Non Volatile Memory (NVM) before an energy failure occurs. Intermittent devices have scarce energy resources, and writes in NVM are energy-hungry operations needed to make the computation look continuous, so they represent an overhead.

We explored the use of STT-MRAM as the NVM for an intermittent computing system. By piloting the write current, we were able to decrease the energy consumed by writes in STT-MRAM at the cost of introducing some errors in the written data. This allowed us to reduce the NVM write overhead while accepting a degradation in the Quality of Results (QoR).

We reported that energy savings with this approximation technique might be significant. Reducing the cost of writes in STT-MRAM allowed us to save almost 50% of the total energy consumed to run a program while still producing results with a relatively small degradation in the QoR. Unfortunately, the results are not always so exciting. In other cases, an approximation that slightly reduced energy consumption caused a dramatic decrease in the QoR. We defined and quantified a trade-off between energy savings and QoR degradation and explored this tradeoff for various benchmarks. This exploration allowed us to understand when our approach is more effective.

In Chapter 3, we described our research question. We identified where the energy savings that we can obtain could be invested, so what are the obtainable benefits. We found that the benefits are increased availability and throughput of the system and the possibility of having smaller capacitors. At the same time, we identified that the drawback we have to accept is a decrease in the accuracy of the produced results.

In Chapter 4, we described the hardware of our target platform, composed of an MSP430 microcontroller that uses STT-MRAM as NVM. We then defined the framework we utilized to generate our experimental results. Lastly, we described our selection of benchmarks. By selecting benchmarks with different characteristics, we have the possibility to cover various aspects for our experimental evaluation.

In Chapter 5, we reported the experimental results we obtained with our framework, allowing us to evaluate the effectiveness of our approach quantitatively. In this chapter, we explored the tradeoff between energy savings and degradation of the QoR for each benchmark. We explained the reason why some programs are more suited for this approximation technique than others. Moreover, we reported the reason why in some programs, the errors followed more complex patterns than in others.

For example for an image decoder benchmark, using our approximation technique, we saved up to 50% of the energy used for a correct execution, with minimal degradation in the QoR. However, the results for a Fast Fourier Transform benchmark are vastly different. The energy consumption can be reduced at most of 4% compared to a fully correct execution. At the same time, the quality of the output decreases drastically, making almost all the outputs produced useless.

The benchmark we evaluated were either equivalent to a single task, so to be executed in a single energy burst or a pipeline of tasks, the same as a task-based intermittent system. We did not perform an experimental evaluation on a dynamic checkpoint-based intermittent computing system where the whole program state is persisted in NVM in points we cannot know at compile time. Exploring the use of our approximation technique in this type of intermittent system could give even better results since checkpoint-based systems have a more intensive use of NVM compared to task-based and is a possible direction for future work.

Moreover, in Chapter 3, we explained that checkpoints written in NVM or the output of tasks are typically used as input in the next power cycle. This means that errors introduced in the early energy cycles have a greater impact on the QoR than those introduced in later energy cycles. Implementing the possibility of changing at runtime the approximation level so to minimize this snowball effect can be an interesting direction to enhance the effectiveness of our approximation technique.

In Chapter 5, we explored the energy/quality tradeoff for each benchmark, identifying the "optimal" approximation level as the one that provides significant energy savings while not decreasing the QoR significantly. However, it is not always possible to quantify what constitutes a "significant decrease in the QoR" in cases like image processing. For example, consider a system upstream that uses the output produced by the Susan edge detection program. In some cases, it may be tolerable to have some edges not correctly detected, while in other cases, even a small error could cause the system to malfunction. Incorporating an upstream system that uses the approximated outputs of an intermittent computing system as an input into our framework could be material for future work. This would allow us to explore the trade-off we defined and better determine the optimal approximation level.

In conclusion, one of the major challenges in intermittent computing is reducing the overhead of preserving the system's state across energy failures. This problem has been the focus of many studies in the literature. In our work, we propose a hardware approximate technique based on adjusting the write current of STT-MRAM, which takes advantage of the stochastic switching property of these memories. Our results show that it is possible to achieve significant reductions in the overhead of preserving the system's state by sacrificing the correctness of the results.

Bibliography

[1] Jagan Singh Meena, Simon Min Sze, Umesh Chand, and Tseung-Yuen Tseng. Overview of emerging nonvolatile memory technologies. *Nanoscale research letters*, 9:1–33, 2014.

[2] Benjamin Ransford, Jacob Sorber, and Kevin Fu. Mementos: System support for long-running computation on rfid-scale devices. In *Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems*, pages 159–170, 2011.

[3] Naveed Anwar Bhatti and Luca Mottola. Harvos: Efficient code instrumentation for transiently-powered embedded sensing. In *Proceedings of the 16th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 209–219, 2017.

[4] 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.

[5] Yiming Huai et al. Spin-transfer torque mram (stt-mram): Challenges and prospects. *AAPPS bulletin*, 18(6):33–40, 2008.

[6] Amir Mahdi Hosseini Monazzah, Amir M Rahmani, Antonio Miele, and Nikil Dutt. Cast: content-aware stt-mram cache write management for different levels of approximation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(12):4385–4398, 2020.

[7] Fulvio Bambusi, Francesco Cerizzi, Yamin Lee, and Luca Mottola. The case for approximate intermittent computing. In *2022 21st ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pages 463–476, 2022. doi: 10.1109/IPSN54338.2022.00044.

[8] 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.

[9] 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.

[10] Joakim Eriksson, Adam Dunkels, Niclas Finne, Fredrik Osterlind, and Thiemo Voigt. Mspsim–an extensible simulator for msp430-equipped sensor boards. In *Proceedings of the European Conference on Wireless Sensor Networks (EWSN), Poster/Demo session*, volume 118, 2007.

[11] Joel Van Der Woude and Matthew Hicks. Intermittent computation without hardware support or programmer intervention. In *OSDI*, pages 17–32, 2016.

[12] MiBench2. MiBench2: A benchmark suite for micro-architectural research. `https://github.com/impedimentToProgress/MiBench2`, 2019.

[13] Sparsh Mittal. Power management techniques for data centers: A survey. *arXiv preprint arXiv:1404.6681*, 2014.

[14] Anthesis NRDC. Data center efficiency assessment, 2014.

[15] Rangharajan Venkatesan, Amit Agarwal, Kaushik Roy, and Anand Raghunathan. Macaco: Modeling and analysis of circuits for approximate computing. In *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 667–673. IEEE, 2011.

[16] Thomas Yeh, Petros Faloutsos, Milos Ercegovac, Sanjay Patel, and Glenn Reinman. The art of deception: Adaptive precision reduction for area efficient physics acceleration. In *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*, pages 394–406. IEEE, 2007.

[17] Stelios Sidiroglou-Douskos, Sasa Misailovic, Henry Hoffmann, and Martin Rinard. Managing performance vs. accuracy trade-offs with loop perforation. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pages 124–134, 2011.

[18] Vinay Kumar Chippa, Debabrata Mohapatra, Kaushik Roy, Srimat T Chakradhar, and Anand Raghunathan. Scalable effort hardware design. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(9):2004–2016, 2014.

[19] Amir Yazdanbakhsh, Gennady Pekhimenko, Bradley Thwaites, Hadi Esmaeilzadeh, Onur Mutlu, and Todd C Mowry. Rfvp: Rollback-free value prediction with safe-to-approximate loads. *ACM Transactions on Architecture and Code Optimization (TACO)*, 12(4):1–26, 2016.

[20] Thierry Moreau, Jacob Nelson, Adrian Sampson, Hadi Esmaeilzadeh, and Luis Ceze. Approximate computing on programmable socs via neural acceleration. *Technical report*, 2014.

[21] Shrikanth Ganapathy, Georgios Karakonstantis, Adam Teman, and Andreas Burg. Mitigating the impact of faults in unreliable memories for error-resilient applications. In *Proceedings of the 52nd Annual Design Automation Conference*, pages 1–6, 2015.

[22] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. Architecture support for disciplined approximate programming. In *Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems*, pages 301–312, 2012.

[23] Abbas Rahimi, Amirali Ghofrani, Kwang-Ting Cheng, Luca Benini, and Rajesh K Gupta. Approximate associative memristive memory for energy-efficient gpus. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1497–1502. IEEE, 2015.

[24] Ismail Akturk, Karen Khatamifard, and Ulya R Karpuzcu. On quantification of accuracy loss in approximate computing. In *Workshop on duplicating, deconstructing and debunking (WDDD)*, volume 15, page 28, 2015.

[25] Ronald G Dreslinski, Michael Wieckowski, David Blaauw, Dennis Sylvester, and Trevor Mudge. Near-threshold computing: Reclaiming moore's law through energy efficient integrated circuits. *Proceedings of the IEEE*, 98(2):253–266, 2010.

[26] Mingoo Seok, Scott Hanson, Yu-Shiang Lin, Zhiyoong Foo, Daeyeon Kim, Yoon-myung 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.

[27] Shekhar Borkar, Tanay Karnik, Siva Narendra, Jim Tschanz, Ali Keshavarzi, and Vivek De. Parameter variations and impact on circuits and microarchitecture. In *Proceedings of the 40th annual Design Automation Conference*, pages 338–342, 2003.

[28] Andrew B Kahng and Seokhyeong Kang. Accuracy-configurable adder for approxi-

mate arithmetic designs. In *Proceedings of the 49th annual design automation conference*, pages 820–825, 2012.

[29] Knud Lasse Lueth. State of the iot 2018: Number of iot devices now at 7b – market accelerating, 2018. URL `https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/`.

[30] Josiah Hester, Nicole Tobias, Amir Rahmati, Lanny Sitanayah, Daniel Holcomb, Kevin Fu, Wayne P Burleson, and Jacob Sorber. Persistent clocks for batteryless sensing devices. *ACM Transactions on Embedded Computing Systems (TECS)*, 15 (4):1–28, 2016.

[31] Kiwan Maeng, Alexei Colin, and Brandon Lucia. Alpaca: Intermittent execution without checkpoints. *Proceedings of the ACM on Programming Languages*, 1 (OOPSLA):1–30, 2017.

[32] Alexei Colin and Brandon Lucia. Chain: tasks and channels for reliable intermittent programs. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 514–530, 2016.

[33] Brandon Lucia and Benjamin Ransford. A simpler, safer programming and execution model for intermittent systems. *ACM SIGPLAN Notices*, 50(6):575–585, 2015.

[34] 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.

[35] Sudhanshu Khanna, Steven C Bartling, Michael Clinton, Scott Summerfelt, John A Rodriguez, and Hugh P McAdams. An fram-based nonvolatile logic mcu soc exhibiting 100% digital state retention at vdd= 0 v achieving zero leakage with $<$ 400-ns wakeup time for ulp applications. *IEEE Journal of Solid-State Circuits*, 49(1):95–106, 2013.

[36] Yong Kyu Lee, Yoonjong Song, JooChan Kim, SeChung Oh, Byoung-Jae Bae, SangHumn Lee, JungHyuk Lee, UngHwan Pi, Boyoung Seo, Hyunsung Jung, et al. Embedded stt-mram in 28-nm fdsoi logic process for industrial mcu/iot application. In *2018 IEEE Symposium on VLSI Technology*, pages 181–182. IEEE, 2018.

[37] Masanori Natsui, Daisuke Suzuki, Akira Tamakoshi, Toshinari Watanabe, Hiroaki

Honjo, Hiroki Koike, Takashi Nasuno, Yitao Ma, Takaho Tanigawa, Yasuo Noguchi, et al. A 47.14-\mu\text {W} 200-mhz mos/mtj-hybrid nonvolatile microcontroller unit embedding stt-mram and fpga for iot applications. *IEEE Journal of Solid-State Circuits*, 54(11):2991–3004, 2019.

[38] Hrishikesh Jayakumar, Arnab Raha, Woo Suk Lee, and Vijay Raghunathan. Quick-recall: A hw/sw approach for computing across power cycles in transiently powered computers. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 12(1):1–19, 2015.

[39] T Devolder, J Hayakawa, K Ito, H Takahashi, S Ikeda, P Crozat, N Zerounian, Joo-Von Kim, C Chappert, and H Ohno. Single-shot time-resolved measurements of nanosecond-scale spin-transfer induced switching: Stochastic versus deterministic aspects. *Physical review letters*, 100(5):057206, 2008.

[40] Xiuyuan Bi, Zhenyu Sun, Hai Li, and Wenqing Wu. Probabilistic design methodology to improve run-time stability and performance of stt-ram caches. In *Proceedings of the International Conference on Computer-Aided Design*, pages 88–94, 2012.

[41] Amir Mahdi Hosseini Monazzah, Majid Shoushtari, Seyed Ghassem Miremadi, Amir M Rahmani, and Nikil Dutt. Quark: Quality-configurable approximate stt-mram cache by fine-grained tuning of reliability-energy knobs. In *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6. IEEE, 2017.

[42] 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.

[43] MSP430L092 Ultra-Low-Power MCU. `https://www.ti.com/lit/ds/symlink/msp430l092.pdf`, 2021. [Accessed: Apr. 4, 2023].

[44] Texas Instruments. Msp430g2553 mixed signal microcontroller, 2011. URL `https://www.ti.com/lit/ds/symlink/msp430g2553.pdf`.

[45] Vipul Kumar Singhal, Vinod Menezes, Srinivasa Chakravarthy, and Mahesh Mehendale. 8.3 a 10.5a/mhz at 16mhz single-cycle non-volatile memory access microcontroller with full state retention at 108na in a 90nm process. In *2015 IEEE International Solid-State Circuits Conference - (ISSCC) Digest of Technical Papers*, pages 1–3, 2015. doi: 10.1109/ISSCC.2015.7062969.

[46] Fredrik Osterlind, Erik Pramsten, Daniel Roberthson, Joakim Eriksson, Niclas Finne,

and Thiemo Voigt. Integrating building automation systems and wireless sensor networks. In *2007 IEEE Conference on Emerging Technologies and Factory Automation (EFTA 2007)*, pages 1376–1379, 2007. doi: 10.1109/EFTA.2007.4416941.

[47] Zimu Li, Guodong Feng, Fenghe Liu, Jia Q Dong, Ridha Kamoua, and Wendy Tang. Wireless health monitoring system. In *2010 IEEE Long Island Systems, Applications and Technology Conference*, pages 1–4, 2010. doi: 10.1109/LISAT.2010.5478274.

[48] Amir Yazdanbakhsh, Divya Mahajan, Hadi Esmaeilzadeh, and Pejman Lotfi-Kamran. Axbench: A multiplatform benchmark suite for approximate computing. *IEEE Design Test*, 34(2):60–68, 2017. doi: 10.1109/MDAT.2016.2630270.

[49] Behzad Zeinali, Dimitrios Karsinos, and Farshad Moradi. Progressive scaled stt-ram for approximate computing in multimedia applications. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 65(7):938–942, 2018. doi: 10.1109/TCSII.2017.2738844.