Applying memoization as an approximate computing method for transiently powered systems

Perju, D.Ș.

Award date:
2019

Awarding institution:
Royal Institute of Technology

DEGREE PROJECT IN INFORMATION AND COMMUNICATION TECHNOLOGY,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, *SWEDEN 2019*

Applying Memoization as an Approximate Computing Method for Transiently Powered Systems

DRAGOS-STEFAN PERJU

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Applying Memoization as an Approximate Computing Method for Transiently Powered Systems

DRAGOȘ-ȘTEFAN PERJU

Master in ICT Innovation
Date: Wednesday 7th August, 2019
Supervisor: Yu Yang
Examiner: Ahmed Hemani
Industrial supervisor: Naveed Bhatti
School of Information and Communication Technology
Host company: RISE Research Institiutes of Sweden
Swedish title: Tillämpa Memoization i en Ungefärlig Beräkningsmetod för Transientdrivna System

Abstract

Internet of Things (IoT) is becoming a more and more prevailing technology, as it not only makes the routine of our life easier, but it also helps industry and enteprise become more efficient. The high potential of IoT can also help support our own population on Earth, through precision agriculture, smart transportation, smart city and so on. It is therefore important that IoT is made scalable in a sustainable manner, in order to secure our own future as well.

The current work is concerning transiently powered systems (TPS), which are embedded systems that use energy harvesting as their only power source. In their basic form, TPS suffer frequent reboots due to unreliable availability of energy from the environment. Initially, the throughput of such systems are therefore lower than their battery-enabled counterparts. To improve this, TPS involve checkpointing of RAM and processor state to non-volatile memory, as to keep computation progress saved throughout power loss intervals.

The aim of this project is to lower the number of checkpoints necessary during an application run on a TPS in the generic case, by using approximate computing. The energy need of TPS is lowered using approximations, meaning more results are coming through when the system is working between power loss periods. For this study, the memoization technique is implemented in the form of a hash table. The Kalman filter is taken as the testing application of choice, to run on the Microchip SAM-L11 embedded platform.

The memoization technique manages to yield an improvement for the Kalman application considered, versus the initial baseline version of the program. A user is allowed to ”balance” between more energy savings but more inaccurate results or the opposite, by adjusting a ”quality knob” variable epsilon ϵ. For example, for an epsilon $\epsilon = 0.7$, the improvement is of 32% fewer checkpoints needed than for the baseline version, with the output deviating by 42% on average and 71% at its maximum point.

The proof of concept has been made, being that approximate computing can indeed improve the throughput of TPS and make them more feasiable. It is pointed out however that only one single application type was tested, with a certain input trace. The hash table method implemented can behave differently depending on what application and/or data it is working with. It is therefore suggested that a pre-analysis of the specific dataset and application can be done at design time, in order to check feasiability of applying approximations for the certain case considered.

Keywords: memoization, approximate computing, intermittent computing, energy harvesting, embedded systems, internet of things

Sammanfattning

Internet of Things (IoT) håller på att bli en mer och mer utbredd teknik, eftersom det inte bara underlättar rutiner i vårt liv, utan det hjälper också industrin och företag att bli effektivare. Den höga potentialen med IoT kan också hjälpa till att ge stöd åt vår egen befolkning på jorden, genom precisionslantbruk, smart transport, smarta städer och mer. Det är därför viktigt att IoT görs skalbart på ett hållbart sätt för att säkra vår egen framtid.

Det nuvarande arbetet handlar om transientdrivna system (TPS), vilket är inbäddade system som använder energiskörning som sin enda kraftkälla. I sin grundform har TPS ofta återställningar på grund av opålitlig tillgång till energi från miljön. Ursprungligen är därför sådana systems genomströmning lägre än deras batteriaktiverade motsvarigheter. För att förbättra detta använder TPS kontrollpunkter i RAM och processortillstånd till icke-flyktigt minne, för att hålla beräkningsförloppet sparat under strömförlustintervaller.

Syftet med detta projekt är att sänka antalet kontrollpunkter som krävs under en applikationskörning på en TPS i ett generiskt fall, genom att använda ungefärlig datorberäkning. Energibehovet för TPS sänks med ungefärliga belopp, vilket innebär att fler resultat kommer när systemet arbetar mellan strömförlustperioder. För denna studie implementeras memoiseringstekniken i form av en hashtabell. Kalman-filtret tas som testapplikation för att köra på Microchip SAM-L11 inbäddad plattform.

Memoization-tekniken lyckas ge en förbättring för Kalman-applikationen som beaktades, jämfört med den ursprungliga baslinjeversionen av programmet. En användare får ”balansera” mellan mer energibesparingar men mer felaktiga resultat eller motsatsen genom att justera en ”kvalitetsrat”-variabel epsilon ϵ. Till exempel, för en epsilon $\epsilon = 0.7$, är förbättringen 32% färre kontrollpunkter som behövs än för baslinjeversionen, med en utdata avvikelse med 42% i genomsnitt och 71% vid sin högsta punkt.

Beviset på konceptet har gjorts, att ungefärlig databeräkning verkligen kan förbättra genomströmning av TPS och göra dem mer genomförbara. Det påpekas dock att endast en enda applikationstyp testades, med ett visst inmatningsspår. Den implementerade hashtabellmetoden kan bete sig annorlunda beroende på vilken applikation och/eller data den arbetar med. Det föreslås därför att en föranalys av det specifika datasättet och applikationen kan göras vid designtidpunkten för att kontrollera genomförbarheten av att tillämpa ungefärliga belopp för det aktuella fallet.

Nyckelord: memoization, approximativ databehandling, intermittent databehandling, energi skördar, inbäddade system, sakernas internet

List of Figures

List of Tables

Acronyms

A Amperes

BLE Bluetooth Low-Energy

C Coulombs

DLL Dynamic link library
DMA Direct memory access

CSV Comma-separated values

EB Exabytes
EU European Union

F Farads
FPU Floating point unit
FRAM Ferroelectric random-access memory

GB Gigabytes
GPIO General-purpose input/output
GPR General purpose registers
GWh Gigawatt-hour

IDE Integrated development environment
IoT Internet of Things

J Joules

kWh Kilowatt-hour

LR Link register

MJ Mega Joules

OS Operating System

PC Program counter (register)

RAM Random-access memory

SIMD Single-instruction Multiple Data
SRAM Static random-access memory

TI Texas Instruments
TPS Transiently Powered Systems

USB Universal Serial Bus

ZB Zettabytes

Contents

Chapter 1

Introduction

1.1 Motivation

It is known that Internet of Things (IoT) is a widely used technology platform that serves us well today. It can be said that it has revolutionized the way people gather information, do work, or even interact with the world [1]. It has brought many improvements to industries such as agriculture [2], healthcare [3], manufacturing [4, 5] and transportation [6], as well as others. Being a popular platform with a wide range of applications, the number of IoT devices has been growing significantly each year, ever since the term was coined in 1999 [7]. As per Ericsson [8] and Gartner's [9] reports, there were between 8 and 11 billion connected[1, 2] IoT devices in 2018. What's more, in 2024, the number of connected IoT devices is estimated to reach 22.3 billion (Ericsson's estimate [8]). There are other reports that give estimations beyond 2024, which however seem unrealistic: Cisco predicts that in 2050 there will be as much as 500 billion IoT connected devices [12]. According to Nordrum [13], estimations too far ahead regarding the population of IoT devices have been too optimistic before. It can be closer to the truth to expect a number of between 100 and 200 billion IoT devices in 2050, extrapolating Ericsson's approximation of 17% on average annual growth in the number of devices [8].

[1] Gartner's estimation is noted as the number of *connected* devices, not distinguishing between the different types of connectivity. Ericsson categorizes IoT devices in its report as being either *wide-area* or *short-range* devices. *Wide-area* devices are internet-connected devices, which includes both *cellular*-connected devices and *non-cellular*-connected devices (being wireless and wired devices, as mentioned in previous reports). As for *short-range* devices, they imply connectivity using ZigBee, Bluetooth, 6LoWPAN, and other similar protocols [10, 11].

[2] The numbers don't include PCs, laptops, tablets or smartphones.

Regardless, the mentioned numbers show that IoT is increasing in popularity, which should bring a source of enthusiasm. We start to depend more and more on such a technology, and for good reason. IoT is not just an improvement in terms of consolidating certain work areas using technology, but it can also guarantee us a more sustainable future as our own population increases around the globe [14]. Therefore, it is important to minimize flaws in such a rich technological ecosystem that, by extension, can sustain our demographic development.

1.1.1 Main issue: IoT's global "energy drain"

The current thesis would like to single out one important issue of IoT that is emerging along with its increasing popularity: the impact that such a high number of "things" can have over energy consumption at a global scale – a concept also known as the *energy drain of IoT* [15]. This is arguably the most important defect that the rapid expansion of "things" can have on the world, especially in today's current context of global climate change [16]. For example, the European Union's (EU) 2030 energy strategy involves a "40% cut in greenhouse gas emissions compared to 1990 levels" and "at least 27% share of renewable energy consumption" [17]. Improving energy consumption of IoT devices can certainly help achieve such sustainability goals.

Internet-connected "things" also inherently need the "cloud", since data that is collected by embedded devices is to be processed further by servers, where it is converted into "value" [7]. As such, data centers also increase in size relatively to IoT's popularity and therefore contribute to the energy budget needed for IoT [18, 19]. For this thesis, however, energy consumption of the "cloud" is considered as a separate problem and is therefore to be disregarded for the present discussion. The current work would however like to focus on the power requirements that essentially appear as a consequence of an IoT device being powered on, such as:

1. The obvious energy consumption that happens simply from *the processing and use of sensors* that an embedded system is undergoing

2. The cost of *wireless communication* needed for such systems: As IoT is everywhere, wireless communication makes sense for such devices to be more mobile. However, *short-range* and, more so, *wide-area* communication (cellular communication) can prove to be costly in terms of energy consumption [20] that should be accounted for.

3. Lastly, the cost of *manufacturing batteries* for wireless IoT: Batteries that are necessary for the mobility of IoT devices do not only incur a financial cost to the user purchasing them. Alongside, there is an ecological impact done by the manufacturing process of batteries. This makes them undesirable in the long run, for a greener future, if alternatives are indeed possible to power up embedded devices.

An estimation of IoT's energy consumption at a large scale

The following section paint a picture of how much energy IoT consumes at a global scale, through some calculations conducted.

Table 1.1: Electrical current intensity (mA) values of Texas Instruments (TI) SensorTag C2650. Source: Lea [21, p. 66]

Functionality	**Current draw**
Stand-by mode	0.24 mA
Running with all sensors disabled	0.33 mA
LEDs	ignored (off)
All sensors on at 100 ms/sample data rate and broadcasting bluetooth low-energy (BLE)	5.5 mA
- Temperature sensor	0.84 mA
- Light sensor	0.56 mA
- Accelerometer and gyros	4.68 mA
- Barometric sensor	0.5 mA

The example of Texas Instruments's (TI) SensorTag C2650 has been given, being a low-powered embedded system that can be used as an IoT device, of which its electrical current draw values for various operation modes have been listed in table 1.1. Considering the most power-intensive mode of operation listed, being that all sensors are sampling and the system is running at 5.5 mA, and knowing that the number of IoT devices was around 11 billion in 2018 and is growing up to be 22.3 billion in 2024, as it was already mentioned [8], Bhide's calculation can be carried out to obtain numbers for IoT's energy consumption at a large scale. For 2018, the calculation would be $11\ billion\ devices \times (5.5\ mA\ /\ 1000) \times (8\ 760\ hours\ in\ a\ year) \times (3.3\ V)$ and would result in around 1 749 gigawatt-hours (GWh) [22]. For 2024, by repeating the same calculation it can be concluded that IoT can consume at least 3 546 GWh in that year[3].

[3]The numbers for both 2018 and 2024 regarding the energy consumed are noted as being

To have a comparison, EU's net electricity generation was 3.1 million GWh in 2016 [23]. No report about EU's electricity production in the year 2018 was released to this day, however we can speculate that the number for 2018 will be between the value for 2016 and 2008's 3.216 million GWh value, which is the peak of electricity production in the last 3 decades. As the tendency of EU's electricity production is to stagnate [23], assuming the 3.1 million GWh value of 2016 as well for 2018 seems reasonable. Having in regard Ericsson's report [8], Europe accounts for only about 25% share of the 11 billion population of IoT devices in 2018 worldwide and goes down to approximately 20% of the 22.3 billion devices approximation in 2024[4]. Table 1.2 shows the actual values from the percentages mentioned, being the IoT devices' population and power consumption after narrowing down such numbers from worldwide to EU level only.

Table 1.2: Population and power consumption of IoT devices worldwide and in Europe only. Source: Ericsson [8]

	2018	**2024**
Worldwide: Number of IoT devices	11 billion	22.3 billion
Worldwide: Power consumption of IoT devices	1 749 GWh	3 546 GWh
In Europe: Number of IoT devices	2.75 billion (25% of 11 billion)	4.46 billion (20% of 22.3 billion)
In Europe: Power consumption of IoT devices	437.25 GWh	709.2 GWh

Given the above values, it can be concluded that IoT devices have only an 0.013% share of the electricity consumption that is happening in Europe, caused only by the processing happening on the systems and their use of their peripherals alone, all under a low-powered architecture. However, with the future of IoT being wireless, that means that such an embedded system does not only leave an energy use fingerprint only from its own resource utilization alone but also indirectly, due to the fact that it needs batteries as a power source

the minimums due to the fact that the calculations are based on an ultra-low power embedded platform. Indeed, other IoT devices exist and are used today, that can consume more energy than the system taken as example. If they were considered, it would raise the energy numbers but would have also made the calculations harder.

[4] As approximated from the graphs of the report, as there are no specific ratios mentioned in the text.

and it communicates through wireless transmissions.

Battery manufacturing considerations. Romare and Dahllöf [24] state in their study that, on average, 500 mega-joules (MJ) are required to produce 1 kilowatt-hour (kWh) worth of lithium-ion battery energy. For the TI SensorTag C2650 which it was taken as an example, for it to be able to work at 5.5 mA constantly for a year, it needs 48 180 mAh worth of batteries (equivalent to six AA batteries to be replaced every 3 months, assuming them having 2000-2100 mAh each). 2.75 billion IoT devices, as it was approximated to be in Europe, have consumed 437.25 GWh in year 2018, as it was noted in table 1.2. This means that producing a year worth of lithium-ion batteries for so many devices would cost 218.625 billion MJ, which converts to 60 730 GWh worth of electricity used only for manufacturing batteries (1 billion MJ being equivalent to 277.78 GWh). This ups the IoT's overall electricity consumption in Europe from 0.013% to 1.97%. However, it is to be highlighted that this calculation does not take into account all the aspects regarding use of batteries for IoT, the necessity of different applications and so on. For example, rechargable batteries can be used and, indirectly so, can attenuate their implied energy costs for manufacturing, if reused multiple times.

Wireless communication considerations. Furthermore, the IoT technology of today implies wireless connection to the Internet. Therefore, for the purpose of the calculation conducted, only mobile data transfers are considered, and the energy consumption of low-range wireless connections (e.g. Bluetooth, ZigBee, etc.) are disregarded. In the context of this thesis, the embedded system that is used for experiments generates about 250 kilobytes of data for every 3000 accelerometer coordinates approximately. A usual accelerometer can sample up to 128 coordinates every second [25]. Arbitrarily considering that about 1000 coordinates would be needed for every transmission of information over the internet, that means that a packet of information would be sent every 7.8 seconds. As such, it can be decided that such a packet would be as big as 50 accelerometer coordinates in size (we are not sending all 1000 coordinates, but digested information), meaning that about 4.2 kilobytes every ~10 seconds would be transmitted. That means that an IoT device would transmit around 12.63 gigabytes (GB) of data every year and, consequentially, we can estimate therefore that for example, in 2018 in Europe, the value of how much IoT would have communicated through wireless connectivity could be as much as 34.73 exabytes (EB) or 0.03473 zettabytes (ZB) of data[5].

For comparison, Cisco says that in 2018, "machines and things" would

[5]Of course, on the rough assumption that all IoT devices worldwide would communicate only accelerometer data.

have communicated between each other as much as 400 ZB of data around the world [26]. One study in Finland [27] evaluated openly accessible information from phone carriers and found that approximately 0.2 kWh is used when a single gigabyte of information is transported through mobile data infrastructure. This leads to an estimation that the electricity consumption value in Europe, in year 2018, due to IoT devices, could be equal to as much as 6 947 GWh. This raises the overall electricity use of such systems in Europe from 1.97% to 2.19%.

Conclusion. Of course, the above calculation was exaggerated in some aspects, while some numbers were obtained from estimations, however the end aim was to illustrate a point. There are various applications in IoT that have different requirements in architecture (either low-powered or otherwise), in energy (smaller or bigger batteries) and regarding data transmission (more or less frequent data being sent, of various packet sizes). Regardless, the main issue was raised, being that the energy needs of IoT have to be lowered, in order to forge a path of a more safer and sustainable future, of such a technology and of the society overall. The calculation was mostly focused on year 2018, but in the future we expect IoT to grow almost exponentially [8] and, because of that, its energy consumption should show growth in the same trend as well. A rampant growth in energy consumption of IoT is to be expected, especially because some future technologies are having trouble scaling down their energy consumption as the IoT trend grows. To extend the latter, we can give the case of 4G, which consumes 23 times more than 3G [28], which is 15 times more energy consuming than WiFi [29]. Due to this, 5G's overall energy consumption is still a debate. On one hand, each base station will have even more hardware implemented in comparison to 4G networks [30], on the other hand its feature set is expected to decrease the energy consumption on average over the whole network [30]. To decrease energy consumption of mobile networks, 3G and 2G networks should also be phased out, as they consume "approx. 70% of the busy hour power when [the network is] at the low-load state" [27].

In spite of that, newer technology of mobile networks and battery sources can not compensate the use of energy if IoT devices themselves do not change their behavior in regard to their resource utilization and power consumption. The trend in Europe is to use less electricity [23] while IoT popularity is growing, meaning that if IoT stays the same, it will occupy more of the energy budget for EU than before. For example, in 2024 based on the same calculation, IoT in EU would consume 3.55% of the electricity provided. If we would follow Cisco's prediction of 500 billion IoT devices in 2050 [12] and

consider 15% of them to be in the EU, then IoT would consume as much as 60% of EU's electricity budget in 2050, assuming that electricity production remains the same further into the future as well. However, as stated before, reports that look far in the future regarding IoT can overestimate. Nevertheless, we should aim for a greener technology and new alternatives need to be implemented, not only at the providing end (e.g. by implementing power grids that supply energy using renewable solutions), but also at the consumer's end through various hardware and software alternatives that do not take only speed and processing efficiency in mind, but also better energy consumption that can sacrifice the former for a better future.

1.1.2 Energy harvesting & checkpointing

In order to design the IoT devices of tomorrow, innovation should be brought in the form of energy harvesting for every node. Decoupling the system from the battery requirement and switching over to energy harvesting as a standalone power source for every system makes such embedded systems more independent and mobile, while also bringing tremendous benefits due to lower maintenance costs (as there are no more batteries to change) and the obvious greener energy benefits (as in, not using a central power grid to charge batteries). This is an important next step for IoT as a technology as it makes it scale up safely. Energy harvesting would be the physical modification while, on the software side, *checkpointing* would be the solution to support the whole ensemble.

As the battery disappears and is replaced with a capacitor, the electric charge available for an embedded system employing energy harvesting is smaller from the start. The energy taken from the environment is not always available, and therefore power outages are to be expected. This causes programs on such system to restart from the beginning when power comes back up, overall slowing down computational progression, compared to an IoT node that is powered constantly by battery. The solution, that checkpointing as a technique provides, triggers the system to save its state when energy is detected to be low, and restore it after a reboot when power is regained. As such, programs will not have to start their computation from the beginning anymore with this solution.

1.1.3 Transiently powered systems

Devices that benefit from energy harvesting as their main power source are named *transiently powered systems* (TPS) or *intermittent computing systems*. As mentioned, they suffer from frequent reboots as energy from the environment is not always available, but comes in bursts. However, energy harvesting benefits embedded systems as they increase their mobility and decrease their need for maintenance (e.g. batteries are not needed to be changed anymore). The current research aims to increase the throughput of transiently powered systems, by using consolidating software techniques such as checkpointing. Through this technological progress, they can become more feasible and therefore a viable replacement of their battery-powered counter parts. Transiently powered systems are a progress of the same concept that is already applied in the real world, such as whole power grids that employ energy harvesting, as well as standalone energy harvesting for cars [31] [32]. Decoupling more devices from the main power grid, especially such numerous ones as it is in the IoT ecosystem, is a good idea – however applying such a technology at a lower scale however becomes more difficult. This work contributes by exploring the applicability of approximate computing as another consolidating software framework for such systems.

1.1.4 Approximate computing

As energy harvesting and checkpointing is employed for IoT devices, it means that the implemented application need not to squabble more energy than it needs to at every step of the computation in order to prevent quick discharges of the capacitor. *Approximate computing* is an optimization that can further help in this regard, yielding an even lower energy consumption, meaning less checkpointing activity happening and therefore fewer pauses in computational progression overall. As it is detailed in the section *2 Background* later, checkpointing by itself involves overhead from writing and reading the states of the system, which consumes energy and is therefore preferred to be avoided for it to happen too often.

Not all applications concerning IoT can tolerate inaccuracies in results or sequences of power outage that are inherent with such solutions, which therefore can not employ approximate computing or even consider energy harvesting for their systems, as the only power source for them. However, there are many embedded systems that have the uncomplicated duty of simply collecting and forwarding data from sensors, which would benefit the entire world as demonstrated, if they could sometimes skip computation and transmission of

collected data if considered to not always be required. It would save energy of the system itself and of the mobile data network as well. Alternatively, energy can also be saved by indeed computing the values needed, but with a margin of error. All in all, approximations do not have the main purpose of lowering energy needs by processing less, but are more important for keeping a forward progress regarding computation, in order to make energy harvesting for IoT devices feasible, as opposed to using batteries as the primary power source.

1.2 Thesis Statement

The purpose of this thesis is to develop and experiment with an approximate computing technique, being *memoization*, as a quality knob to reduce energy usage of a transiently powered system, while maintaining precision of results within tolerable limits. Considering that transiently powered systems usually also employ checkpointing and restoration techniques, the goal of this project is then to lower the number of checkpoint and restoration cycles that need to happen on average throughout an application run of such a system. This can happen only by lowering the overall energy consumption done, which is relative to the number of power loss events. Decreasing the energy requirements is to be done by applying approximate computing. If a lower number of checkpoints happen throughout the runtime of an application, then less energy is wasted for the overhead that is needed for checkpointing. This also automatically means that there are fewer power loss events and, so, TPS get closer to the throughput that their battery-powered counterparts can achieve, only also including the greener energy benefits.

Boundaries. This work attempts to successfully implement a proof of concept where an approximate computing technique can work in the present, on current age embedded systems. Some literature explored in chapter 2, *Background* explore the next step directly, implying significant changes over today's embedded systems, that can prove costly or with some drawbacks, implying risks for users and enterprises. In this sense, this work does not allow significant modifications of the platform of choice. It adopts the idea that the problem of transiently powered systems can be applied on present systems and, more so, software instead of hardware can be synthesized to help such a process. Therefore, no invasive hardware techniques are considered in this work. However, a component attachment (being the Ferroelectric RAM / FRAM) is indeed studied as merely an addon, in order to show the significant performance that it can entail for reference. More so, the development board is not a TPS in itself and, as such, no real-life measurements were done in this sense.

Instead, a mathematical model helped to obtain an evaluation of how such a system could do in the restricted energy circumstances intermittent computing entails, being the theme of this work. The component attachment was also studied through the same paradigm.

1.3 Thesis Structure

This thesis is comprised of 5 chapters. Chapter 1, *Introduction* – the current one – is the chapter that informs and gives the motivation of transiently powered systems. The complementary software mechanism of checkpointing is also described, and approximate computing – being the main technique applied by this work – is initially described as well. A calculation about IoT's energy consumption at a large scale is conducted, in order to point out the benefit that can be achieved through the present project.

Chapter 2, *Background*, provides the necessary documentation about important concepts already mentioned in the introduction – being TPS, checkpointing, approximate computing – as well as the different approximation methods that can be applied, including memoization – which is selected method of choice. More so, it discusses the application of choice that was tested on in this thesis, being the Kalman filter. The chapter also introduces literature discussing different implementations of approximate computing applied to various TPS scenarios. Such work is correlated with the requirements wanted in the current work (building on top of the previous **boundaries** paragraph above), triggering the need of a new implementation.

Chapter 3, *Method and implementation*, is the chapter that provides detail regarding this research's aims, as well as the outline of the research processes – being a quantitative one through experiments. It also discusses the implementation of choice, being a hash table, which is a form of memoization.

Chapter 4, *Experimentation and evaluation*, details the experiment setup, being the Atmel Microchip SAM-L11 embedded platform. The experiment process, as well as the procedure of canvasing of the different experiment parameters (epsilon ϵ, hash table size, iterations), are detailed. The metrics obtained are shown in the form of plots and then discussed.

Chapter 5, *Conclusion*, finally portrays the study's successful outcomes, as well as possible the future work that can continue this research.

Chapter 2

Background

This section will go through the following topics: *transiently powered systems*, being the main hardware platform that this work is focusing on, *checkpointing*, which is the key software technique that complements the operation of such devices, as well *approximate computing*, which is the concept adopted by this work that further improves transiently powered systems as a technology. All in all, the reader will get a better overview of this new, ambitious type of embedded systems that aims to have a greener energy source, as well as the research and theory revolving around this technology – of TPS, checkpointing and approximate computing.

2.1 Transiently Powered Systems

2.1.1 Description

Also known as *intermittent computing systems* [33, 34], *transiently powered systems* [35] are a class of embedded systems that do not use batteries as a power source, but instead rely on harvested energy from the environment (e.g. energy sources such as light/solar, radio frequency, vibration/piezo, magnetic and thermoelectric). Energy from the environment is not reliable – being not always available – therefore resulting in frequent restarts of the system. Restarts of the system means that computation progress will be often lost and the system would have to re-run its whole program from scratch after reboot every time, following the power loss event. This leads to duplicated effort by wasting time reaching to the same point in computation as before losing power. In comparison, embedded systems powered by reliable power sources (such as batteries) do not have such a problem from the start.

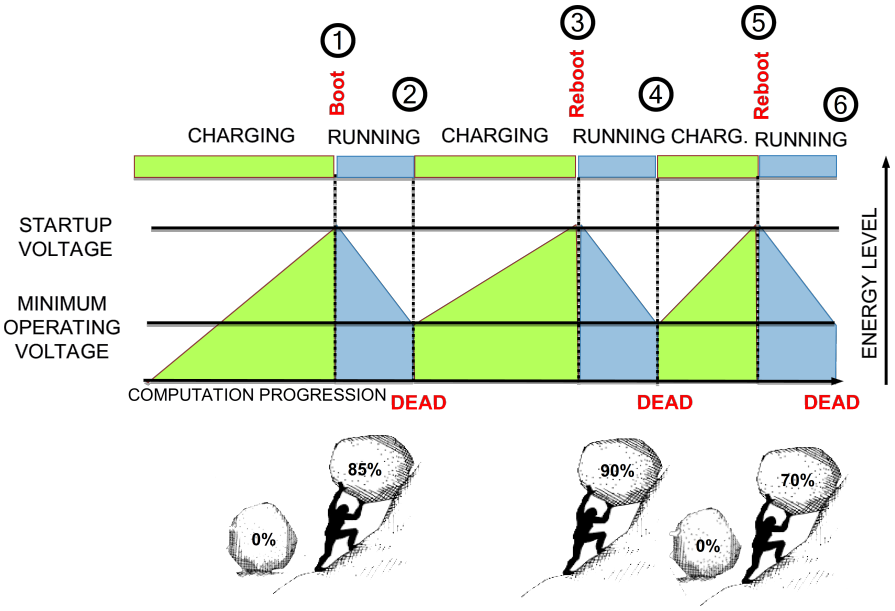

Figure 2.1: Typical timeline of computation happening on a transiently powered system, without any additional software mechanisms enabled (no checkpointing, no approximation)

In figure 2.1, a slice of the lifecycle of a software application that is running on an intermittent computing system is drawn. Studying the figure, it can be seen that the system has intervals in which it is powered off and the considered software application is not running and thus not making any progress, being the areas highlighted in green. During this time, the energy harvesting mechanism charges the capacitor of the system back to the *startup voltage* threshold, being the energy level required for the system to boot-up from scratch. Afterwards, the system is then able to run again, having the intervals in which it is operating normally highlighted in blue. The undergoing computation consumes the collected energy in the capacitor[1], up until the moment when the operating voltage given by the capacitor goes lower than the *minimum operating voltage*, when the system needs to power off again. The points (1), (3) and (5) depict the moments in time when the system has to (re)boot from scratch, while points (2), (4) and (6) depict points when the system needs to power-off due to insufficient energy in the capacitor to work with.

More so, as the system is running figure 2.1 depicts a system without a

[1]The capacitor can, in the meantime, still charge through energy harvesting but at a faster rate than the computation depleting the collected energy.

checkpointing mechanism installed, the progress of the computation[2] is often lost and has to be restarted from the very beginning (from 0%) at the points (2), (4) and (6). This is depicted in figure 2.1 by the silhouette of Sisyphus, an iconic Greek king whose myth discusses a similar fate of always having the boulder fall regardless of the attempts to push it further up the hill[3] (attempting but never reaching 100%).

Overall, the time that a transiently powered system is actively running its installed software application, as illustrated in figure 2.1, is not enough. Regular embedded systems that are properly powered run their software application with no interruptions 99% of the time – occasional downtime being caused due to rare major power failures but not due to general inconsistency of energy available, which is an issue only in the context transiently powered systems. The field of study revolving around such systems attempts to overcome such a setback, to increase the ratio of time the system is actively running, versus it being disabled and recharging its capacitor. The aim is to raise the throughput closer to that of their battery-enabled counterparts and thus then becoming a good candidate to replace them in the future, due to the greener energy approach. One course of action to bring improvement for transiently powered systems is to prioritize throughput while leaving other factors at an disadvantage, such as a decreased accuracy of results, which is the approach this thesis is targeting.

2.1.2 Checkpointing

As the focus is to make intermittent computing systems more reliable, techniques such as *checkpointing* [35] are being employed. *Checkpointing* has the result of maintaining progress of computation amid reboots. Such a mechanism has the positive effect of maintaining a healthy level of throughput, unlike before when the results were barely coming through (computation barely reaching 100% as depicted in 2.1). Checkpointing works by saving the state of the system, storing all or just parts of the RAM memory to a non-volatile memory (such as flash), as well as saving relevant processor register values (in the case of this thesis: the general purpose registers *GPRs*, the program counter *PC* and the link register *LR* are saved alongside RAM).

[2] One example of computation can be a lengthy calculation of a set of instructions considered together to be an *atomic* operation (being a set of instructions which can not be split for a checkpointing procedure to happen in-between them). Such an *atomic* operation can be a single iteration of a *for* loop, for example.

[3] Credits for this analogy go to the industrial supervisor of this work.

This forces developers of such applications, that are dedicating their work to such energy-constrained platforms, to have to consider many factors. For example, the more the application will occupy random-access memory (RAM) during its runtime, the bigger the time and energy overhead will be to checkpoint the relevant data structures to non-volatile memory. The basic approach for using checkpointing involves programmers manually marking points in code where checkpointing is allowed to happen as the program runs [36]. It is however important to prevent the case where an an application checkpoints mid-way of a computation step and, when it is restored afterwards, goes into an unusable state. Checkpointing should also happen in carefully selected locations of the code, the involved considerations being that it should not happen too often as to waste time and energy, as well as that it should happen at "smart" locations in the code – for example **not** where the application uses a lot of RAM memory than then needs to be checkpointed. All of such matters can become default implications in regard to embedded systems programming in the future, as developers will need to consciously implement proper energy management in software, as energy efficiency becomes more of importance to society.

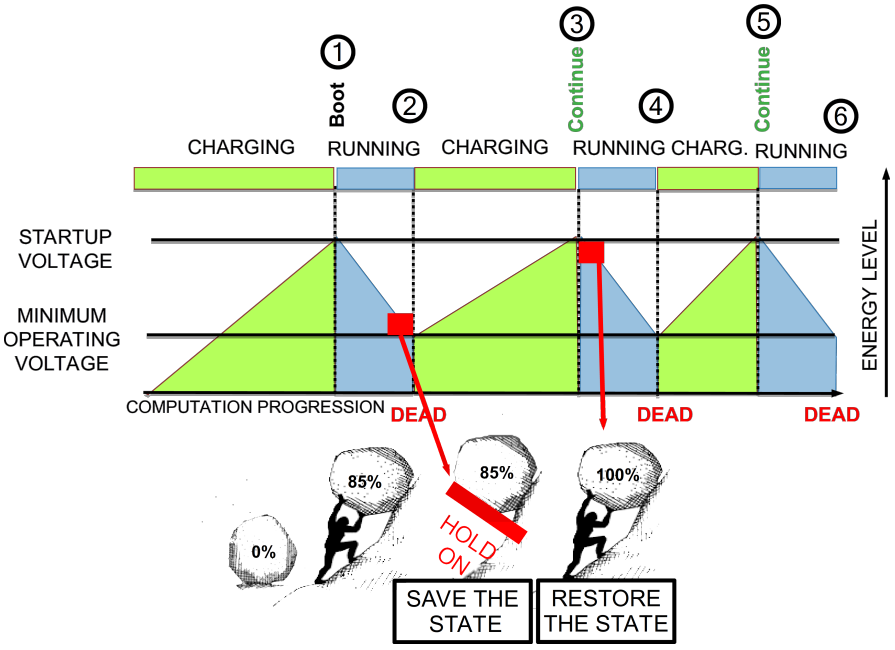

Figure 2.2: Timeline of computation happening on a transiently powered system with checkpointing enabled only

Referring back to the figure 2.1 that was in the previous section, now up-

dated in figure 2.2 to include the checkpointing mechanism, it can be seen that the overall time the system runs for is still the same. However, the computation progress during the times when the system is powered off is now kept, due to checkpointing being employed. At point (2), the system saves the state to a volatile memory. At point (3), the system *restores* the configuration of the system before the power loss back into the RAM memory. Although not depicted, this also happens at point (4) and (6) in the figure.

There are various ways of applying checkpointing that is documented in current research. One basic form was mentioned before, being the manual placement of points in code where checkpointing is allowed to happen, a method which can be beneficial for manageable amounts of code. The HarvOS [37] method talks about this. Other mechanisms automatically place points in code where the voltage is compared to a set threshold and checkpointing is done only after this check to see if the system is running low on energy (e.g. when voltage is lower than the set threshold). MementOS [38] and Hibernus [39] depict such automatic procedures for applying checkpointing.

Some checkpointing techniques rely on hardware support, such as Ratchet [40] and Clank [41]. Besides the switch from battery to energy harvesting, such checkpointing techniques involve extra changes such as using **only** non-volatile memory, even for the medium that is holding the RAM. This means that the copying procedure of checkpointing would be entirely skipped, as data used by the application would be preserved implicitly by the non-volatile medium as power is lost. Such major changes brought to the usual embedded systems, however, are disconsidered by this work, as this makes transiently powered systems harder to adopt by enterprise and individuals, as they involve a higher financial cost for a technology that is not a direct upgrade of the former version. For example, non-volatile systems have a slower execution, while bringing higher financial costs. A non-volatile memory acting as a RAM for the final implementation of this project is modeled however, and experimental results are shown in the *4 Experimentation and evaluation* chapter, revealing promising results. It is again important, still, to regard the aforementioned drawbacks and take the intermediary step that involves the lesser risk of converting current embedded systems with transiently powered systems with the RAM still on a volatile medium, before then taking the bigger step to a more distruptive technology of transiently powered systems with full non-volatile memory structure.

All-in-all, as noted, more and more checkpointing mechanisms are being developed in favor for transiently powered systems in various ways, all with their merit in advancing the paradigm needed to push forward such systems to

be the default choice of the future. in regard to the present thesis, a work that has peaked interest regarding checkpointing is of Bhatti and Mottola [42]. The mentioned work talks about various ways to checkpoint the `Stack`, `Heap` and `Data+BSS` memory segments that are usually specific to a software embedded application, present in the RAM and are therefore necessary to be copied in flash before power loss happens. However, what is to be remarked in the mentioned paper are the simple, smart and various ways checkpointing can be approached, which can help the reader further understand checkpointing and can offer relevant techniques to apply for the current project as well:

1. **Split**: If any memory layout diagram of a software application is studied, it can be seen that the `Stack` segment sits on one side of the address space of the RAM and `Heap` and `Data+BSS` segments are placed at the other side. Naturally, it means that the RAM will always have free memory in-between (in the middle of the RAM), that can be tracked and excluded from checkpointing, instead of the whole RAM to be checkpointed as a whole, wasting energy and time as useless values from the free space area are copied also. This is what the ”split” method mentioned by Bhatti and Mottola [42] employs, which is important for the case in this thesis as well. To be specific, the energy graphs considering both checkpointing and approximate computing, that are obtained in the *4 Experimentation and evaluation*, are considering only the useful memory being copied during checkpoints as well.

2. **Heap tracker**: Sometimes, dynamic memory allocation is triggered during the runtime of the embedded application (e.g. by using the *malloc* function in C/C++ programming language), meaning that the heap memory is used. This is irrelevant to our case, as the application considered in this thesis does not use calls for dynamic memory allocation. In the case of Bhatti and Mottola [42], where dynamic memory allocation is used, the *heap memory* area in RAM usually becomes non-contiguous after some time, as the memory suffers through a sequence of allocations and deallocations, causing memory fragmentation. Most of the time, heap memory allocation algorithms easily know the location and size of each memory chunk that has been allocated. The ”heap tracker” method can therefore, utilizing the same data structures of allocation algorithms, easily know how to checkpoint only the relevant memory that is indeed allocated instead of copying the whole heap into non-volatile memory and wasting effort on copying useless bytes from unallocated memory areas as well. Similarly, this can be used for the

hash table structure. Since the first time the application is run, the hash table takes some time to populate all its initially empty locations, as the application goes through its iterations, and so not all of the contiguous memory space used for the array needs to be checkpointed into memory from the beginning. In a similar fashion to the ”heap tracker”, the hash table can track such unused locations in the array and exclude them from being copied, since it is always important to keep the size of the data that will be checkpointed as small as possible, to reduce the number of writes to non-volatile memory. Granted, the hash table has the tendency to fill itself up quickly and therefore can make such a tracking feature useless as the application runs for a longer period of time. However, this method, inspired by the ”heap tracker”, keeps the hash table optimal at all times at a minimum cost of computation (few checks to see if the ”changed” flag was toggled versus a number of useless non-volatile memory writes). Therefore, the checkpoint is always kept at a minimum size, which can help even for some corner cases where, for example, the input data does not ”match” the hashing function, meaning that some locations in the array can remain empty for a longer period of time. More so, a mechanism that marks some locations as to not be copied during the next checkpoint based on how old they are (e.g. a number of iterations passed and the respective values in the array were not used) can also be employed. This makes the previously explained method still useful even when the application runs for a longer period of time, naturally filling up the hash table as it is supposed to. This also indirectly keeps the data fresh and keeps the system error-free, by discarding old data so that it is not reused way later when the application is in a much different state, where the old result might be irrelevant.

3. **Copy-if-changed**: More so, if copying only the occupied sectors makes sense, incrementally updating the same checkpoint over the first ever version stored in non-volatile memory can be more efficient as well. This is what the ”copy-if-changed” method is about, and the hash table used in the current work can easily imitate such an approach. ”Copy-if-changed” implies comparing the current RAM content with the previous checkpoint already done, to decide which parts are new and to be copied. The hash table can easily mark data that was changed from the last checkpointed version of the array, in order to do the same thing but even in a faster way. A bigger hash table helps the system make better approximations, but otherwise has the disadvantage of making

> the checkpoints bigger and therefore more costly energy-wise, as more flash writing operations are needed for the bigger array. By checkpointing only newly updated values, a bigger hash table's disadvantage can be diminished, as essentially a smaller version of it is checkpointed, just as if we would use a smaller hash table from the start, which would be copied as a whole by a more basic checkpointing mechanism. This is also discussed in the future work section of the *5 Conclusion* chapter.

Bhatti and Mottola [42] notes however that there is always an extra but fixed overhead most of the time when the checkpointing mechanism wants to selectively copy the data to non-volatile memory using a ”smart” strategy. ”Heap tracker” needs a data structure to track the occupied vs. unoccupied heap – a structure that, once introduced, is needed to be read beforehand from the flash memory and checkpointed to it afterwards as well, in addition to the usual data that is also to be copied. The ”copy-if-changed” method needs to compare data from the flash memory with the data from RAM, which means extra reading operations from flash memory are necessary before checkpointing. The authors note in their study, in regard to flash memory, that reading from it is a faster and less energy consuming operation than writing to it, and so the aforementioned methods therefore try to exchange many writing operations for reading ones, in order to save energy and make such methods feasible.

However, they conclude that such ”smarter” approaches for the checkpointing mechanism have a favorable effect only when the application operates with a relatively small quantity of data between checkpoints. For example, in regard to the ”heap tracker” method, if there are a certain number of allocated blocks that, together with the additional data structure used, make up a bigger size of data to be checkpointed that is bigger than the size of the heap itself, then the advantage of the ”heap tracker” method is nullified. The additional data structure used becomes the unnecessary overhead that can be dropped in favor to changing the checkpointing system back to its basic form, if the application often has a big number of allocations most of the time. Considering the ”copy-if-changed” strategy, if only a small set of data is being changed since the last checkpoint, then the read operations needed for the comparison procedure will successfully counter-balance the writing operations that is otherwise done by basic checkpointing, which are now skipped. In opposite to that, if the RAM content is significantly different than the last checkpoint, then the read operations only become an additional energy overhead as most of the memory needs to be copied anyway. As it was for the ”heap tracker”, by profiling the application beforehand it can be understood if the ”copy-if-changed” mechanism will bring an overall positive contribution or it should be discarded for

the respective case.

To summarize this section, the checkpointing software mechanism was presented. Various approaches of how it can be implemented, along with its advantages and disadvantages, were detailed. As such, a larger outline, as well as some specifics, were all included in this section to give a rich picture to the reader on the matter of checkpointing. Hints to the implementation done in this work, which will be explained more completely in later chapters, were done in order to show how checkpointing is important and can be connected to any additional software mechanisms meant for the transiently powered systems.

2.2 Approximate computing methods

Approximate computing is a ”necessary evil” that is to be employed in cases where energy improvement is more desired than accuracy of data, such as it is the case for intermittently powered systems. In the end, the data of a lower precision should still be able to be synthesized into relevant information that is ”good enough” in quality (for example, alerts should still work at the frequency wanted, even if data is not measured as often). The quality threshold is something to be decided by the user. Approximate computing is not an obligatory setback with the advent of transiently computing systems, but it simply provokes developers of such a platform to engage in the thinking required for such a technology to achieve a higher standard. Sometimes lowering precision of a certain data acquisition process can prove to have a significant boost in terms of energy, while having little impact on the quality of information that is being derived from that data. It is simply that the data acquisition process has not been looked upon by the developer, because it can take time to find the best way to do it, or because there are no big restrictions to incentivize efficiency at this level of processing, or otherwise due to the fact that there are no ready-made tools for it. The first and last motives are what drive this thesis.

More so, some domains of work even allow approximate computing to happen. In this category, the example of glucose monitoring can be given. Ganesan, Miguel, and Jerger [36] mentions that international standards regarding glucose monitoring allow it to have an error range of ± 20% when doing calculations on numbers. Possibly, only *some* work domains can be tackled with approximate computing, while others implicate safety risks that an embedded application should not be open to or cause. However, most fields have not explored the possibility of raising efficiency through lowering the otherwise perfect quality of their data utilized, disputably for the reason that they never considered such a paradigm.

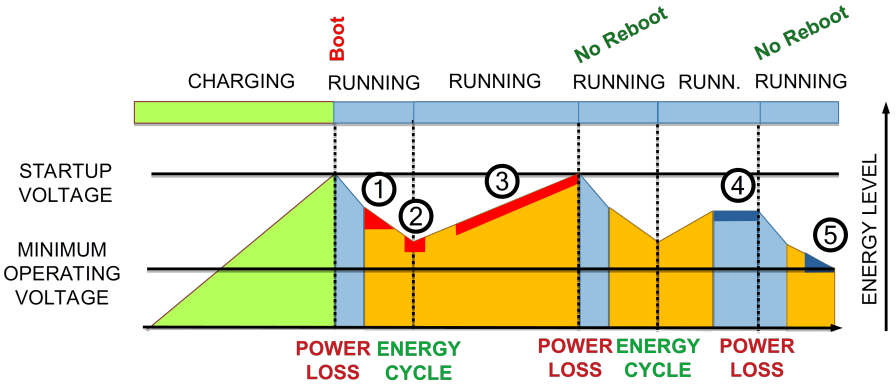

Figure 2.3: Timeline of computation happening on a transiently powered system with checkpointing and approximations enabled

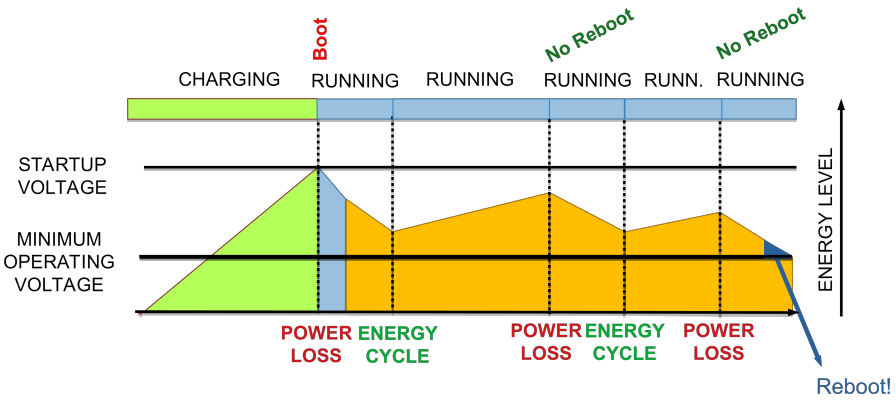

Figure 2.4: Timeline of computation happening on a transiently powered system with checkpointing and approximation enabled. The bad case when approximations can not keep with with energy depletion is considered

Looking at figure 2.3, the same example can be seen, of an application running on an intermittent computing system with checkpointing enabled, as shown previously in section *2.1.2 Checkpointing* (figure 2.2). This time, approximations are enabled to compensate and reduce the number of checkpoints that are happening. Figure 2.3 manages this well enough to prevent any reboots and keeps the overall energy trend consistent (not lowering down), being an idealistic case. Figure 2.4 shows a more realistic case, where the system eventually reboots, although this happens much later than when only just checkpointing was working as the standalone mechanism in the previous figure 2.2.

In figure 2.3, at point (1) we see that approximations are activated when energy is running low. Of course, it could be the case that approximations are always activated, being applied for the whole runtime of the application, but approximate computing can also run in a ”smart”, advanced way, which this figure is portraying. As such, in this instance the approximations are activated dynamically. At (2), the usual checkpointing procedure does not happen as it did in figure 2.2, where the energy was low, this time being prevented due to approximations. At (3), we can see how approximations are kept enabled until the system returns to maximum desired energy (being *startup* voltage). At (4), full accuracy might be requested and then approximations could be disabled – which is also an advanced feature.

The figure 2.3 shows an ideal case, where the approximations manage to lower the power consumption of the system enough to allow the harvester to refill the capacitor's energy entirely at least one time (during point (3)). The energy harvesting mechanisms and the approximate computing method therefore successfully complement each other enough to skip any reboots, and so only manual requests or conditions from the user to disable approximations, like at point (4)),manages to cause a reboot, like at point (5). As mentioned, figure 2.4 shows a more realistic case, where approximations do not manage to lower the power consumption enough for the capacitor to ever be able to fully refill itself while the system is online. The system has to suffer through reboots even if approximations are enabled – reboots like the one happening at the end of the figure – and then has to wait for a complete recharge of the capacitor while it is offline and the application is not running. In the figure 2.4, a complete charge while the system is offline only happens at the start of it, highlighted in green, where the capacitor is needed to be fully charged before a bootup of the system can happen. One full recharge with an offline system would also happen again after the end of the figure.

2.2.1 Overview of different methods

Mittal [43] offers a thorough taxonomy on approximate computing techniques. This section will only list and discuss the ones that were considered to be implemented as the main method for this thesis.

Precision scaling is "reducing the bit-width of input or intermediate operands, to reduce storage and computing requirements" [43]. Regarding this method, Mittal [43] point to the study of Yeh et al. [44] that argues that precision scaling would: (1) reduce some floating point operations to "trivial" ones (such as multiplication by one as the operand loses digits and becomes the number one itself), (2) help other techniques such as memoization to work more efficiently, as more values that were close to one another before, now become the same value and (3) work with smaller hardware than it is currently offered by today's systems, such as a smaller floating point unit (FPU), bringing obvious energy and cost benefits. All of these advantages can be obtained through the simple modifications that precision scaling entails. One example of such a modification can be changing all `double` values to `float` for example. After such modifications however, testing is required to see if the outcome is not significantly changed.

Loop perforation describes the simple principle of skipping loop iterations in order to reduce the overall computation done. Mittal [43] point out to the work of Sidiroglou-Douskos et al. [45], which gives the example of a search algorithm. A search algorithm can reduce its search space by skipping checking some values overall, therefore reducing its execution time. It can then return a value that is "good enough" for the conditions implied by the search.

Data sampling is not extensively discussed in the taxonomy of Mittal [43]. It is defined as the skipping of the entire input data that is to be processed, as to save computing power and therefore energy. Data sampling is applicable to the setup built for this thesis, as its input – being an accelerometer and a gyroscope – brings data which can be considered skippable. Possibly not all of the accelerometer's values or the gyroscope's are always needed to detect important events. It can be difficult for some applications to consider this, however, as it is important for some systems in their respective field of work to measure at their maximum acquisition precision.

2.2.2 Memoization

Memoization is an umbrella term for programming methods that work to make computation faster by reusing results of operations already done [43]. Origi-

nally, it implies a method where the values saved from an already done computation are offered instantly again as output, only if the set of inputs that are occuring are the *same* as the ones that computed the results in the first place.

Factorial is a known mathematical notion that can be given as an example of a function that can greatly benefit from memoization. Some values that were already calculated for a specific factorial index given can be saved, such that if a greater index is requested to be computed, then the process can go faster by starting from the already known values. More so, memoization is so powerful that it is an important part of the core of certain programming languages – such as Haskell [46], a functional programming language.

Moving on to approximate computing, memoization works within this paradigm by reusing results for sets of inputs received that are only *similar* values, and not necessarily equal, to the ones that obtained the results in the first place [43]. The inputs received do not necessarily have a pre-calculated output specifically for them, but are close enough to the ones whose result is known and can be given instead.

Memoization is a machine-independent, cross-platform strategy of approximate computing. Unlike the precision scaling method, for example, it is an optimization that works during run-time of the application, instead of compile-time. It obtains more speed for the function that it helps to perform better, in exchange for storage space, as it requires memory to store pre-calculated values.

In the general sense, memoization works on functions that are referentially transparent [47]. If a function has such a trait, it can be directly switched, in the source code of a program for example, with the values it would normally compute. However, as mentioned by [43], the approximate computing version of memoization has to only return *similar* values that the function would normally give out. This is something that will be observed in this thesis, as the application on which approximate computing is tested on, being the Kalman filter, is a non-linear, stateful function, while the hash table that works under the principle of memoization is usually dedicated for linear functions.

2.3 The Kalman filter

For this thesis, the *Kalman filter* has been chosen as a result of a search for an application that is general enough, popularly used in embedded system applications, while also being an intensive application, enough for approximate computing to have a chance to make an impact.

The *Kalman filter* is an optimal estimation algorithm [48], which has many

applications, such as (1) estimating a system state when it can not be measured directly and (2) estimating a system state from various measurement sources, when such inputs have noise [48], with the aim of obtaining the desired single measurement that is absent from said noise. Some examples of the Kalman filter being used are:

1. It can be used to improve the position on the map of a robot, through correlating the position via GPS with the movements the robot has done so far. As such, a more precise position of the robot can be obtained [49]. For example, this is particularly useful for remotely controlling big vehicles used in quarries, where accidents due to position imprecision is unwanted

2. A navigation system can still place the position of a car on a map, while it is passing through a tunnel and has lost its GPS signal. This entails using the measured acceleration of the car, to substitute the lack of signal from the GPS while being in the tunnel [48]

3. The fuel combustion process on a spacecraft can be dangerous if it bypasses temperature limits during a outer space mission, possibly leaving the vehicle stranded as it damages the mechanical parts of the vehicle. In order to detect such a danger, a temperature sensor is needed but however can not be placed directly inside the combustion chamber where the measurement is wanted to be done. Being an extremely high temperature environment that can melt the sensor, a Kalman filter helps in this situation, by obtaining the internal combustion chamber temperature by correlating the temperature from a sensor placed outside, on a cooler surface and a safer distance that is still near the combustion chamber [48]

The intention of this work was to use an already implemented Kalman filter. As such, already public code [50] was found and adapted from the Arduino version that it was dedicated to, in order to work with the experimentation platform used by this thesis (described in section *4.1.2 Experiment platform: SAM-L11*). For the Kalman filter, both an accelerometer and gyroscope traces were needed. The source of the data set is mentioned in the *4.1.3 Experiment data* section, in the *4 Experimentation and evaluation* chapter. The Kalman filter, in this instance, works as a fusion filter of both peripherals, to compute a more reliable set of position coordinates than with just one set of measurements.

2.4 Literature on approximate computing methods applied to transiently powered systems

Numerous approximate computing methods were considered to be adopted and experimented with for this thesis, instead of implementing an own method. However, the applicability of each of the studies found was difficult for the requirements considered in this thesis, much so that it impeded going forward with any one of them. The studies that have been of interest for this thesis have however been shortly described below and some traits of them have been listed in the following paragraphs.

Lee et al. [51] give, in their work, an approximate computing solution in the context of machine learning. The paper describes splitting the learning algorithm in independent, separate actions that can be treated as individual tasks. Then, the authors implement a decision algorithm for a scheduler that can arrange the tasks to be executed efficiently, with respect to the energy remaining for the system to run at any time.

Gobieski, Beckmann, and Lucia [33] implement SONIC, TAILS and GENESIS, being different approximate computing methods in the form of programming techniques implemented at different levels (e.g. compiler, run-time) that complement each other. SONIC is a programming API that offers a few features, such as (1) *task tiling*, which partitions longer loops' iterations into tasks that are possible to fully run for the entirety of the given energy buffer (e.g. a capacitor cycle) and (2) *loop continuation* which, by keeping the index and the data used by a running loop directly in non-volatile memory, can continue the loop after a reboot from a voltage loss almost instantly afterwards[4]. TAILS is a runtime adaptive binding of hardware parameters, meaning that it configures different hardware components, that are usually used by an embedded system, to use energy efficiently at different moments in computation. Gobieski, Beckmann, and Lucia mention that TAILS fully works in the case of configuring the direct memory access (DMA) for optimal block data movement, as well as for configuring the Texas Instruments Low-Energy Accelerator during runtime, for finite impulse response discrete time convolution. Lastly, GENESIS is the last technique implemented by the authors, which has the aim to fit neuronal networks in resource-constrained systems during design time (e.g. before the

[4]This can only be done to loops which have idempotent iterations. Idempotency is defined as "a mathematical quantity which, when applied to itself under a given binary operation (such as multiplication), equals itself" [52].

application is even compiled), trying out different configurations and making a Pareto comparison for the user to choose – being especially important in the case of limited memory capacity.

Ganesan, Miguel, and Jerger [36] implement the *What's Next Architecture*, which encompasses *subword pipelining*, *subword vectorization* and *skim points*. The usual "habit" of computers is to perform addition and multiplication using two operands at a time. As such, when energy runs out for a transiently powered system, it would have naturally obtained a few 100% accurate results before checkpointing happens. As it was already discussed in the previous section *2.1 Transiently Powered Systems* at the beginning of this chapter, the aim of transiently powered systems involves aiming for a higher throughput, giving out a larger number of results at a time than having fewer results but of perfect accuracy. In this sense, *What's Next Architecture* uses *subword pipelining* and *subword vectorization* to split **multiple** operands (not just two), that are about to be added or multiplied together, into subwords. Addition and multiplication will then operate on sets of subwords from multiple values at a time, instead of whole two operands at a time. The two mechanisms work from the most significant subword to the least significant, so that when a power loss happens, approximated values of more results than just two will have already been obtained and stored on non-volatile memory through checkpointing. The *What's Next Architecture* then treats the approximated results as final variants of the calculation when a reboot happens after a power loss, and does not go back to complete the calculation but continues forward with new values in the same fashion. In this way, throughput is raised.

Skim points are simple assembly instructions that a modified compiler implemented for *What's Next Architecture* inserts in various points in the program, such as in loops, in order to mark places where a system can continue its computation after a reboot if it already passed that point in the program (called a skim point) before the power loss. Skim points are automatically inserted for the subword pipelining and subword vectorization techniques to happen.

The work of Ma et al. [53] targets only non-volatile processors. It defines the behaviour of *rolling forward*, in the case of computation on a energy-constrained system. If the system suffers a power loss, after reboot it would "roll forward" in computation, placing the program counter (PC) to point to the new iteration. The paper however describes that if there is extra energy available, then an attempt is made to compute the old iteration in an single-instruction multiple-data (SIMD) fashion along in parallel with the current one.

Table 2.1: Summary of literature studied, classified based on certain factors

	[36]	[51]	[33]	[53]
Main principle	”What’s Next” architecture	Scheduler for machine learning tasks	SONIC, TAILS & GENESIS	”Roll forward” / Incidental computing
Applicable to	General applications	Machine learning	Machine Learning	General applications
Experimentation platform	Simulator	Real scenarios with various embedded systems	Texas Instruments MSP430	Simulator, RTL running in Modelsim
Programming difficulties?	Modified compiler required	No	No	No
Hardware modification necessary?	Yes	No	No	Yes
Source code available?	No	Yes	Yes	No

2.5 Conclusions on literature

Intermittent computing is a new technology that needs, at its infancy, the help of a strong software ecosystem and easy to implement hardware alternatives to promote adoption. The literature mentioned involves completely new system architectures, such as the non-volatile processor and RAM as per the contribution of Ma et al. [53], or software libraries that cater to specific applications, being for example machine learning, as per the work of Lee et al. [51]. This thesis advocates that the establishment of transiently powered systems as a new technology to happen quite quickly, through the for general use, by first addressing uncomplicated hardware solutions and software frameworks meant for all types of applications. In this way, the companies and individuals that want to take the step to a greener future and substitute the usual embedded systems for transiently powered systems, can take the first gradual, low-risk step of adapting the current age platforms that they have now.

This thesis is advocating that the implementation of transiently powered systems should not rely on unfamiliar or difficult changes in terms of hardware or software at the start. Major hardware modifications or completely new system architectures do not cater to the convenience wanted by individuals and companies, which usually look for low-risk, tested technology. It is not to say that such changes (e.g. the non-volatile processor and RAM suggested by Ma et al. [53]) are discouraged – on the contrary, they pave the way to a greener future – but, to encourage the community to take the next step, the progress has to happen gradually through intermediate changes (e.g. energy harvesting installed as an addon, supplemented by the checkpointing mechanism for the volatile RAM, before using only architectures dedicated for intermittent computing).

More so, the software ecosystem that surrounds a new technology is important to grow, as to encourage new developers to adopt said platform. As previously seen in the *2.4 Literature on approximate computing methods applied to transiently powered systems* section, approximation techniques are sometimes developed specifically for an application family (such as machine learning, in the work of Lee et al. [51]). Specific solutions are certainly performing better than generic ones, when solving the narrowed down problem itself. However, the current work intends to contribute with an implementation that is general enough to be adopted by any type of application, irrespective of their software framework they are working with. In that way, the software support around transiently powered systems can evolve in a progressive manner, which can even quicken the adoption of TPS.

Hence, the literature surveyed was not corresponding to the nature of the application this work was meaning to experiment on. The implementation then is constructed from scratch, aiming to bring a contribution to the software ecosystem of transiently powered systems that can be seamlessly applied in the context of the present, instead of having a base on future improvements that can take a while to become a reality.

Chapter 3

Method and implementation

This chapter explains the steps this research has gone through to make careful, thorough progress in validating the choice, as well as the implementation and evaluation of *memoization* – which is the main approximate computing method selected for this study. The implementation takes the form of a *hash table*, of which its structure and function are also explained in this chapter.

3.1 Methodology

The methodology steps of this work are the following:

As mentioned in the thesis statement (section *1.2*), the aim of the presented work is to implement and evaluate an approximate computing method, in the context of a software application that is usual for an embedded system to run. Success is achieved when approximations bring energy consumption benefits without significant drawbacks, thereby increasing the sustainability of transiently powered computing as a technology.

1. Identify a software application of a general type (e.g. not machine learning), popularly used in an embedded system context while having an intensive workload that has enough energy consumption and computation to be saved that it would benefit from approximate computing (e.g. it should not be just a linear application)

2. Choose an approximate computing technique to experiment with, being either a particular approach found in literature, that applies to the application environment mentioned at step 1, or implement an own algorithm by following approximate computing theory as described by the taxonomy of Mittal [43]

3. Select an ultra-low-power embedded platform to evaluate on. The lesser energy consumption the better. The system will not work with energy harvesting or employ checkpointing by itself, but will have a constant power source due to the always-on connection with the computer, for the transfer of dataset as input, debugging and measurements acquirement

4. Construct a measurement system that interacts with the platform decided at step 3, which has the application from step 1 and the approximation technique from step 2 installed together

5. Systematically input *parameters* specific to the approximation technique from step 2, using the measurement system from step 4. In this way, many experiments are run for different configurations automatically, for a rich evaluation

6. Analyze the energy benefits and the scale of imprecision involved due to approximations, after executing step 5

7. Using mathematical models, emulate energy harvesting and checkpointing constraints and provide a new analysis from the perspective of an actual TPS instead of an usual embedded system as before, as it was done at steps 5 and 6

3.2 Applying memoization: The hash table

Memoization is a subcategory of approximate computing methods, of which the theory has been described in section *2.2.2 Memoization*. In the present work, memoization takes the form of a hash table data structure, with an associated hashing function. The hash table essentially pairs sets of input values with sets of output values, in order to reuse computation that has already been done. As approximate computing is implied, multiple sets of inputs can reuse each other's results.

The first step is that the Kalman filter is used to obtain the first output values from a set of inputs. The hashing function is then applied on the inputs, in order to obtain an index at which the pair of input-output values can be stored in the hash table. At the next iteration, the location in the hash table corresponding to the newly obtained index (after hashing the input values) is first checked to see if a set of input-output values is already stored there and if the set can indeed be reused (a second check).

As a hash table can not be infinite, completely different input values can hash to the same index. This is called a collision, and is a familiar issue when discussing hash tables. It is important to not reuse the output values that were meant for completely different input values that the ones given at the current iteration. That is why the input values are also stored in the hash table, in order to do an ulterior check after the hashing function points to a pair of input-output values that could be potentially reused.

Looking at the big picture, the hashing function creates a correlation relationship between the possible input values that can exist and the index values, which are as many as the hash table size. For the current implementation, the input values are ”split” in consecutive intervals, of which the difference between the maximum and the minimum values in an interval is as big as *epsilon* ϵ which is arbitrarily set. The bigger the epsilon ϵ, the more values can share the same index in the hash table. At the same time, the hash table ”wraps”, in the sense that the interval that is immediately ”after” the end of the hash table will then associate to index 0, opening the mentioned possibility of collisions that is inherent to hash tables and therefore can not be avoided easily.

Referring back to the second check mentioned, the current implementation of the hash table compares the given input values at the current iteration with the ones stored at the respective index that was obtained after applying the hashing function. The values are considered to be the same if the respective pairs between the sets of input values are within a difference smaller than the same arbitrary value epsilon ϵ selected. After passing such a check, the stored output values can be reused.

It is immediately obvious then that epsilon ϵ acts as an adjustable *quality knob* that is intrinsic to the approximate method implemented and acts as a balance between accuracy of the output and the energy consumption that is being lowered as computation is skipped. Epsilon ϵ considers multiple coordinates that are close to each other into a single coordinate. It increases the locality between similar input values.

The hashing function of the approximate computing mechanism employed should not waste any more computation than it is absolutely needs to, in order to reduce the almost constant energy overhead for the application as memoization is applied. A hashed value is needed to be calculated before the start of every Kalman iteration. That is why, only a single coordinate – the x-axis coordinate of just the accelerometer – is being hashed every time in order to obtain an index to use for the hash table, instead of hashing all the coordinates of both the accelerometer and the gyroscope. This saves a number of multiplication and division operations per iteration.

If a hit is made and a pair of inputs-outputs is found at the index obtained, then a second check is done. All of the coordinates of both the incoming and saved inputs of the accelerometer and gyroscope are compared, in order to be precise and not allow outputs of totally different input values to be reused. Otherwise, results could end up being selected for sending only because both set of inputs have same x-axis accelerometer value while the other coordinates can be different. The hashing function can also point to the same index while the x-axis coordinates of the incoming and saved values are not comparable at all, the situation being of a collision as mentioned that hash tables usually have. This would lead to very unsafe behavior, where some inputs can be leading to a whole different output than expected, which could then be used to e.g. signal mechanical devices to do inappropriate or drastic movements, dangerous for their surroundings.

Chapter 4

Experimentation and evaluation

This chapter thoroughly describes the experiment setup, as well as the platform of choice for this work – the *Microchip SAM-L11*. The source of the dataset used as an input trace is also mentioned. The second part of this chapter covers the evaluation and analysis done from the results obtained from the experiments.

4.1 Experiment setup

This section expands on the experiment toolkit developed and the hardware that was used in this work.

4.1.1 Overview

The overall experiment setup was automatized to the benefit of running experiments without much intervention, as the whole experimentation stage took some time and any delay that needed user intervention would otherwise have had to be eliminated to make the process faster. Each experiment took 35 to 45 minutes to execute, and around 450 experiments were made. That means that at least a minimum of 11 days is the total time that experiments would have run on the machine, if they ran uninterrupted.

The experiments start with the embedded platform, being Microchip's SAM-L11 Xplained Pro kit which is discussed in the next section. The Kalman application is being flashed for every experiment, varying different configurations (epsilon ϵ, hash table size). The Atmel Studio software suite from Microchip, which has been developed on top of the Microsoft Visual Studio integrated development environment (IDE), was used at the beginning to adapt, design

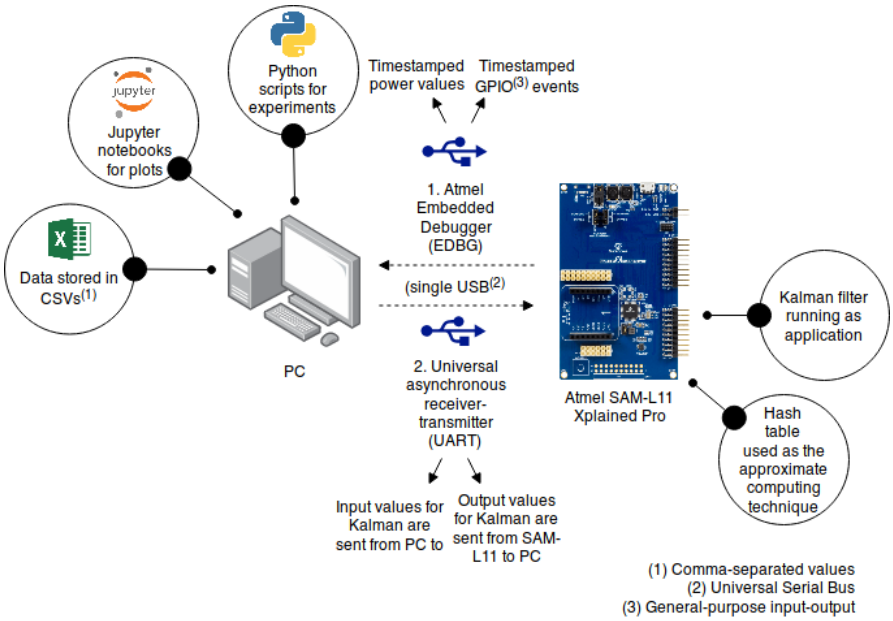

Figure 4.1: The experiment setup

and test the different variants of the same Kalman filter application (baseline and with approximations running on SAM-L11, from the previous Arduino version that was before).

A Python measurement library had to be developed in collaboration with one other thesis worker (Erik Wouters) who developed his thesis on the same embedded platform, as the measurement software from Microchip has been proven to be buggy to the point of being unusable. The SAM-L11 board is a feature-filled embedded platform in the sense that it is capable of taking measurements regarding its power use by itself, without any additional need of external measurement tools (e.g. oscilloscope). The in-house developed Python measurement library is open sourced and available online [54], and credits go to the same colleague who also took the initiative of starting the effort, to which the author of this work had contributed.

The library included an automatic compilation and device programming script that was useful in order to automatize deploying the experiments with different configurations. On top of this ensemble, a multi-threaded Python had been developed to handle power measurements done by the library, while communicating the input sets and receiving the outputs from the board in parallel.

Each experiment would manage to generate output of 1.3 GB data (due to very high resolution power measurement), which put together with the other

experiments' results would occupy tens of GB of space, but which would otherwise archive at a very small size. After an experiment was done, three files would be generated for analysis:

- the output of the Kalman filter itself
- the power measurements
- the general-purpose input/output (GPIO) pin events, through which the board application was signaling the start and stop of every Kalman iteration.

Afterwards, Python scripts and Jupyter Lab notebooks would work to analyze the files

One disadvantage of the Python measurement library was that it had to work on 32-bit architecture, because the dynamic link library (DLL) file that was obtained from Microchip to access functions to interact with the measurement capabilities of the board [55] was compiled to 32-bit only. However, the Windows operating system (OS) the experiments ran on was limiting 32-bit processes to use only 2 GB of RAM at most [56]. The machine that ran the experiments had 8 GB of RAM and a 64-bit version of the OS installed. Therefore, two processes of 32-bit and 64-bit had to be running on the OS, that had to do multi-process communication to synchronize. The 64-bit process was necessary in order to load the aforementioned 1.3 GB of data into memory before processing for analysis.

The experiments would then have their average current in amperes calculated, as well as time duration per iteration and of the whole Kalman filter. It is estimated that about 14800 power samples were obtained every second, while the GPIO events would appear only when the GPIO pins' overall status has changed, efficiently leading to fewer samples being acquired in regards to this.

For every configuration of epsilon ϵ and hash table size, 3 experiments were done. By doing so, information about reliability of the experiment results was able to be conveyed in figures and charts (in the form of error bars, min/max information).

4.1.2 Experiment platform: SAM-L11

Description

As mentioned, Microchip's SAM-L11 Xplained Pro platform is the latest in ultra-low power architecture, having a 32-bit processor released in 2018 that

manages 25 µA/MHz in active mode. This is a major improvement over other well-known platforms, such as the Texas Instruments (TI) MSP430, which has 230 µA / MHz in active mode [57] and is a 16-bit processor. The SAM-L11 Xplained Pro board also incorporates a debugging interface, as well as self-measurement facilities to obtain power information and GPIO status in real-time. No additional hardware measurement tools alongside the SAM-L11 Xplained Pro were therefore needed.

The SAM-L11 is equipped with an ARM Cortex-M23 processor. The writing in flash memory is similar to how Bhatti and Mottola [42] describe the Cortex-M3 writes data to flash memory, being only by writing entire pages even when a few bytes were changed. Reading from the flash memory, however, can be done from any place in memory by any amount of bytes, without an entire page being loaded.

SAM-L11 is equipped with the ARM Cortex-M23 variant that does single-cycle multiplication[1] [59]. The hardware integer division speed is suspected to be able to do 17 cycles maximum[2], however no reliable source for this was found. This is a tremendous benefit for the hash function, which includes a number of multiplication and division operations.

The used platform was equipped with 64 KB flash memory and 16 KB static RAM (SRAM)[3]. For reference, the part number of the SAM-L11 used is `DM320205`.

4.1.3 Experiment data

The input dataset was extracted from the *MotionSense Dataset* [60], used in many research papers such as of Malekzadeh et al. [61], Malekzadeh et al. [62] and Kuncan [63]. The repository offers a number of CSV files produced by 24 different test subjects that were tasked to do normal movements of everyday life on an university campus, such as going upstairs, downstairs, jogging, sitting and standing.

For this thesis, a single set of CSV files was extracted, of all 3 coordinates (x, y, z) for both the accelerometer and gyroscope. Specifically, the selected set was of subject 9 (`sub_9`) at location 4 (`ups_4`), doing the activity of going upstairs.

[1] Otherwise, ARM Cortex-M23 also has a model variant that does 32-cycle multiplication [58].

[2] ARM Cortex-M23 has another specification that does a maximum of 34 cycles integer division [58].

[3] Other variants of the SAM-L11 platform on the market have lower memory specifications [59] [58].

4.2 Evaluation

4.2.1 Experiment parameters

Epsilon

Epsilon ϵ is one of the main parameters for the approximate computing technique designed. Epsilon is essentially the quality knob for the approximate computing employed. A higher epsilon ϵ means that more values, that are close to one another, are grouped to be considered as one single value. Through this, similar inputs are then able to reuse each other's already calculated output set if at least one output is already precomputed. A lower epsilon increases accuracy, as less input values are mapped to the same hash table index.

An epsilon ϵ of 0 would mean each value represents its own actual value in the end, which is the case of the baseline version of the application – being without approximations involved. An epsilon ϵ value of 0.5 would mean the values 0.0 to \$0.499..\$ would be hashed to the same index 0 and reuse each other's outputs, values 0.5 to \$0.999..\$ would be hashed to index 1 and so on. When the second check happens, all coordinates received as input are checked against what is already in the hash table. For example, if any two x-axis accelerometer values, such as 0.46 and 0.57, are to be compared during the second check, the two values will be considered the same only if the absolute value of their difference is less than or equal to epsilon ϵ. If $\epsilon = 0.5$ for example, the pair of values in the aforementioned case are considered equal.

A bigger epsilon ϵ is desired to raise energy savings, but the output's quality needs to be supervised, in order to not deviate too much.

Hash table size

The *hash table size* affects how often pairs of input-output already stored are overwritten. As, naturally, a hash table can not be infinite, the mapping between the input values and the indexes of the hash table is not perfect.

This can be explained easily with an example: if a hash table size would be 5 and epsilon ϵ would be of value 0.5, then the x-axis accelerometer values $0.0..0.499$ would be mapped to index 0, $0.5..0.99$ would use index 1 and so on. The last index would be 4, being used by values $2.0..2.499$. As there is no index 5, the hash table would "wrap" around, and the next values, being $2.5..2.99$, would again use index 0. If at first index 0 would be populated by a pair of input-outputs after the Kalman filter encountering inputs that were hashed for the interval $0.0..0.499$, later on the application can meet inputs

hashed for the interval $2.5..2.999$, being the same index 0. In this case, an *overwrite* is necessary, as the "second check" explained in the previous section *4.2.1 Epsilon* would fail. If the hash table would have been bigger, this specific collision would not have occured. Therefore, a bigger hash table lowers the chance of collisions. Increasing the size of the hash table, however, comes with the tradeoff of using more memory that then translates into more energy consumption due to the longer checkpointing process.

Modulo precision

Modulo precision (sometimes abbreviated as *mod precision*) is a self-defined term for a factor that was multiplied to all of the numbers involved (inputs, epsilon ϵ) that are floating point values. The number multiplied is a power of 10, in order to turn as many digits that are after the "point" of the floating point numbers to be part of the integer side of the same values. From the few adjustments explored, this one considerably makes the hashing function less energy hungry, so much so that the whole approximate computing technique gives positive results. To put it differently, without this adjustment, the approximate computing would have had negative results that would have made it all-in-all unfeasible. The adjustment is necessary also despite of the Cortex-M23 processor's fast multiplication and division capabilities. The "modulo precision" value is obtained after a canvasing of which value would be best, done directly on the computer instead of on the embedded platform. It was then fixed to be a value of 10000 throughout all the experiments. It seemed however that a value of 100 or 1000 of `mod precision` would not make much of a difference from the 10000 value. As the epsilon ϵ itself had two digits after the point, any other digits after that were automatically not considered in the comparisons done and could have otherwise been easily discarded.

4.2.2 Experiment outputs

Charges and duration

After obtaining the measurements after a whole run of Kalman application, obtaining the values of the electrical current intensity in Amperes (A) at different timestamps, to which the GPIO events mark when Kalman iterations start and stop, then it was easy to obtain the charges (measured in Coulombs (C)) of each iteration. A charge is a multiplication of the current intensity (A) and duration of each iteration (seconds). By having the charges of each iteration, the average charges over multiple Kalman application runs, in vari-

ous configurations, are then obvious to obtain. From the charge values, power and energy can be inferred, as well as various time information, being either the duration for every iteration, duration on average per iteration and duration of the whole application run, all while excluding delays due to measurement itself. For reference, the energies of most of the experiments were circling around 100 mJ.

Deviation

The output that the baseline version offers is considered to be the one that gives out the most accurate output that the Kalman filter can give. This 100% accurate version is therefore considered to have 0% deviation, and all the outputs from other experiments have their deviation calculated relatively to this baseline version. The rest of the experiments of course are the ones that have memoization enabled.

As the outputs that are of interest for this thesis are the `x` and `y` coordinates given by the Kalman filter, then for each of the two coordinates their deviation from their baseline values are calculated. If the value at some iteration for the x-axis Kalman coordinate is 150 and the approximation variant gives – at the same iteration and for the same coordinate – a value of 300, then the deviation of the approximation-enabled version is of 100%. If the value given is 225 (mid-point between 150 and 300), then the deviation is 50%. Therefore, the formula to calculate the deviation at a certain iteration i for a certain coordinate c is noted below, as equation 4.1.

$$deviation^{i,c} = \begin{cases} \dfrac{|val^{i,c}_{baseline} - val^{i,c}_{approximation}|}{val^{i,c}_{baseline}}, & \text{if } val^{i,c}_{baseline} \neq 0 \quad (4.1a) \\ |val^{i,c}_{approximation}|, & \text{if } val^{i,c}_{baseline} = 0 \quad (4.1b) \end{cases}$$

To avoid division by 0, that can naturally occur with the standalone formula 4.1a, the exception case 4.1b has been introduced, where if the baseline value would be 0, then the approximated coordinate's absolute value would be considered the deviation percentage by itself, with no extra calculations done.

Number of checkpoints

The value $V = 3.3$V is the chosen supply voltage that the SAM-L11 allows [64]. As the capacitor size can be chosen arbitrarily by the user, it is considered a flexible variable in the formula 4.2. The formula 4.2 expresses the maximum energy that a capacitor can store when it is fully charged, noted as W_{stored}.

Table 4.1: Charge values for read and write operations for the flash memory of Microchip SAM-L11. *Values displayed are after own measurements from experiments specifically made to survey the operation of the flash memory. The write operations are significantly more costly than the read operations, which are of very minimal impact. The write operations are also signicantly more costly than for the FRAM.*

Operation	**Size [Bytes]**	**Measurements**		
Write	1	Time & Charge	2.26 ms	3.01 µC
	256	Time & Charge	5.32 ms	5.59 µC
	512	Time & Charge	10.83 ms	12.46 µC
		Fixed Overhead	-	
Read	1	Time & Charge	0.078 ms	0.0 µC
	256	Time & Charge	1.95 ms	0.71 µC
	512	Time & Charge	3.83 ms	1.41 µC
		Fixed Overhead	-	
		Operating current	230 uA	
		Operating frequency	4 MHz	

Table 4.2: Charge values for read and write operations for the ferroelectric RAM (FRAM) considered. *The FRAM measurements were conducted by another group at the host company this thesis was conducted at. The model of the FRAM itself was not given. The FRAM manages to balance the energy cost between the read and write operations overall better than the flash memory, despite the latter having better energy cost for read operations.*

Operation	**Size [Bytes]**	**Measurements**		
Write	1	Time & Charge	0.0313 ms	0.048 µC
	256	Time & Charge	8 ms	1.88 µC
	512	Time & Charge	15.89 ms	3.7 µC
		Fixed Overhead	0.179 ms	
Read	1	Time & Charge	0.0271 ms	0.039 µC
	256	Time & Charge	6.937 ms	1.63 µC
	512	Time & Charge	13.8 ms	3.21 µC
		Fixed Overhead	0.144 ms	
		Operating current	230 uA	
		Operating frequency	4 MHz	

In other terms, W_{stored} represents how much energy in Joules (J) that can be used before the system will be shut down. During the time allowed before the depletion of the energy W_{stored}, 3 different processes should be allowed to run: (1) an amount of computation should be dedicated to the application running on the embedded platform, (2) the checkpointing procedure itself and (3) the restoration process that happens at the start of the program as well. All of such values should fit in a capacitor cycle represented by W_{stored}. For reference, in formula 4.2, the variable C stands for capacitance, being a characteristic of the capacitor measured in Farads (F). V is the supply voltage mentioned.

$$W_{stored} = \frac{1}{2}CV^2 \tag{4.2}$$

As the average charge of the whole Kalman application running on the entire input dataset is measured and known, as mentioned in the previous section *4.2.2 Charges and duration*, only the supply voltage $V = 3.3$V needs to be then multiplied to the average charge in order to obtain the total energy used by a single application run[4]. Dividing this number by the W_{stored} above, the number of complete capacitor cycles, that are needed for the Kalman application to go through with a single experiment (with a specific epsilon ϵ and hash table size), is obtained. This number is then considered the *number of checkpoints* needed for said application.

Using the formula at 4.2, a mathematical model was achieved. Such a model would obtain an analysis of how the Kalman filter would manifest under the various configurations on a transiently powered system, where a capacitor charged by energy harvesting is considered. The capacitor was fixed to have a capacitance of 40 µF, after a survey of the range of values intermittent computing usually implies. The context in which the model works in is as follows: the system starts with the capacitor fully charged, after which the Kalman filter is running until the capacitor gets fully depleted. During the active times when the application is running, the energy harvesting mechanism does not work. When the capacitor is fully depleted, the application itself is disabled as the capacitor gets replenished. However, the capacitor being recharged is not depicted in any of the final analysis plots and so, can be considered that it happens instantly. In real-life scenarios, the energy harvesting can charge the capacitor even during the running of the application. This would be a better case than the worst case considered in the described mathematical model above, as the number of checkpoints done overall would be lower.

[4] As *energy* is voltage, current intensity and time multiplied, then current intensity and time can be substituted with charge directly.

Memory

Memory occupancy of the application and the approximate computing method is an important factor to obtain from the experiments. Memory occupancy is dependent on two factors: (1) the Kalman application size, which is fixed all throughout the different configurations, and (2) the hash table size, which is dynamically changed throughout the experiments. With the knowledge of memory occupancy of both the application and the hash table size together, the checkpointing process can then be deduced energy-wise easily. The operations of writing and reading to/from flash memory was done as a separate experiment, and the energy needed for such operations for any number of bytes could then be extrapolated. More so, the same values could be extrapolated for a ferroelectric RAM as well, and plots are offered regarding this.

4.2.3 Results

Table 4.3: Overall energy and duration min / max values encountered during the experiments conducted

Measurement	Note	Value	(alternative unit of measurement)
Min-Max Energy of Experiments	All iterations One iteration	0.81 mC - 17.99 mC 2.69 uC - 6.24 uC	2.69 mJ - 59.40 mJ 0.93 uJ - 20.57 uJ
Min-Max Duration of Experiments	All iterations One iteration	2.11 sec - 35.09 sec 0.73 ms - 12.15 ms	
Baseline Energy	All iterations One iteration	16.5 mC 5.71 uC	54.44 mJ 18.85 uJ
Baseline Duration	All iterations One iteration	33.17 s 11.49 ms	

Overview of outputs

Figures 4.2 show the outputs from both the baseline version and the approximate version as well. Through these, it is easier to see the approximations being applied on the output.

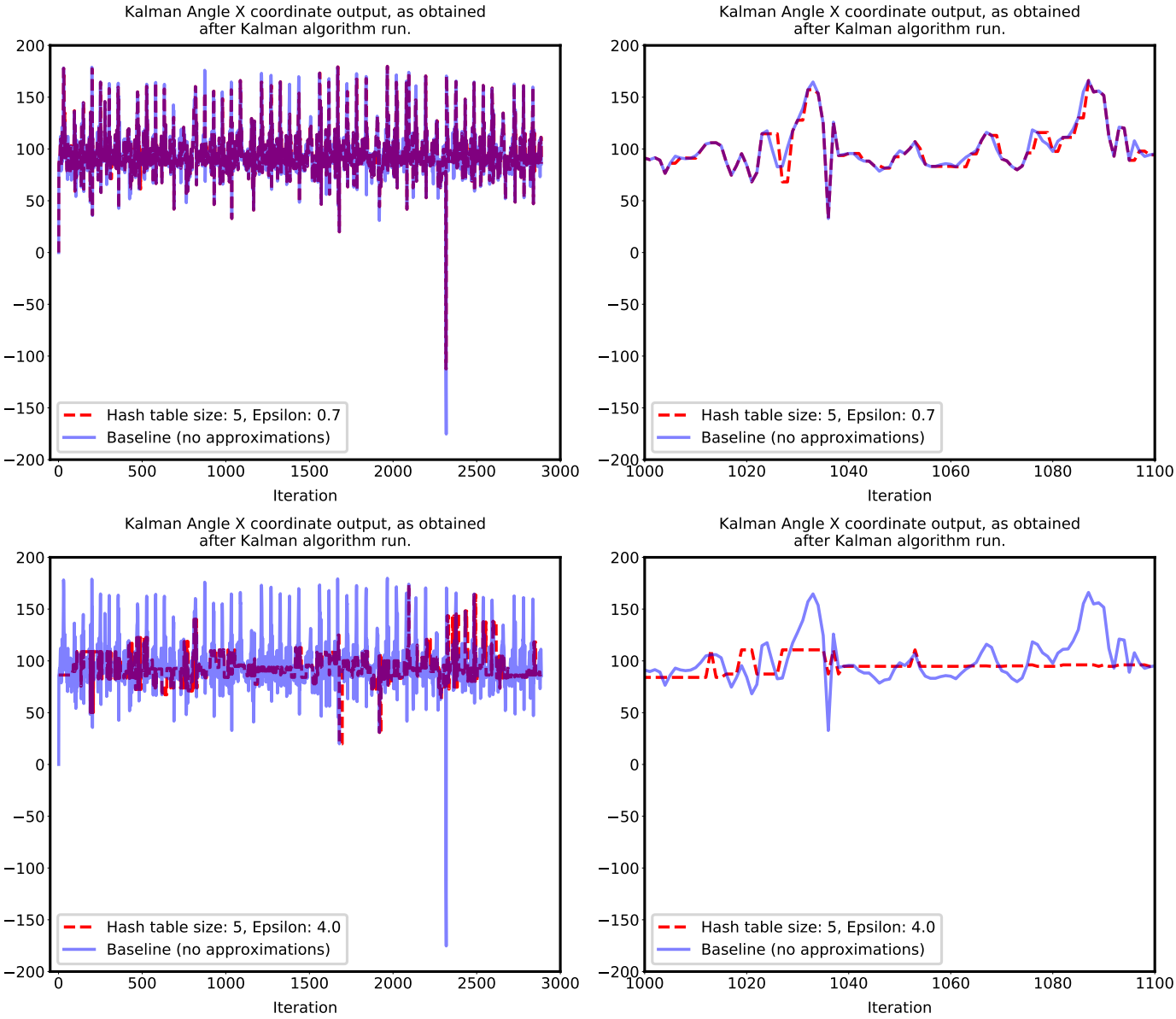

Figure 4.2: The outputs of both baseline and approximations-enabled versions of the Kalman application. On the right side are zoomed in plots of the left side.

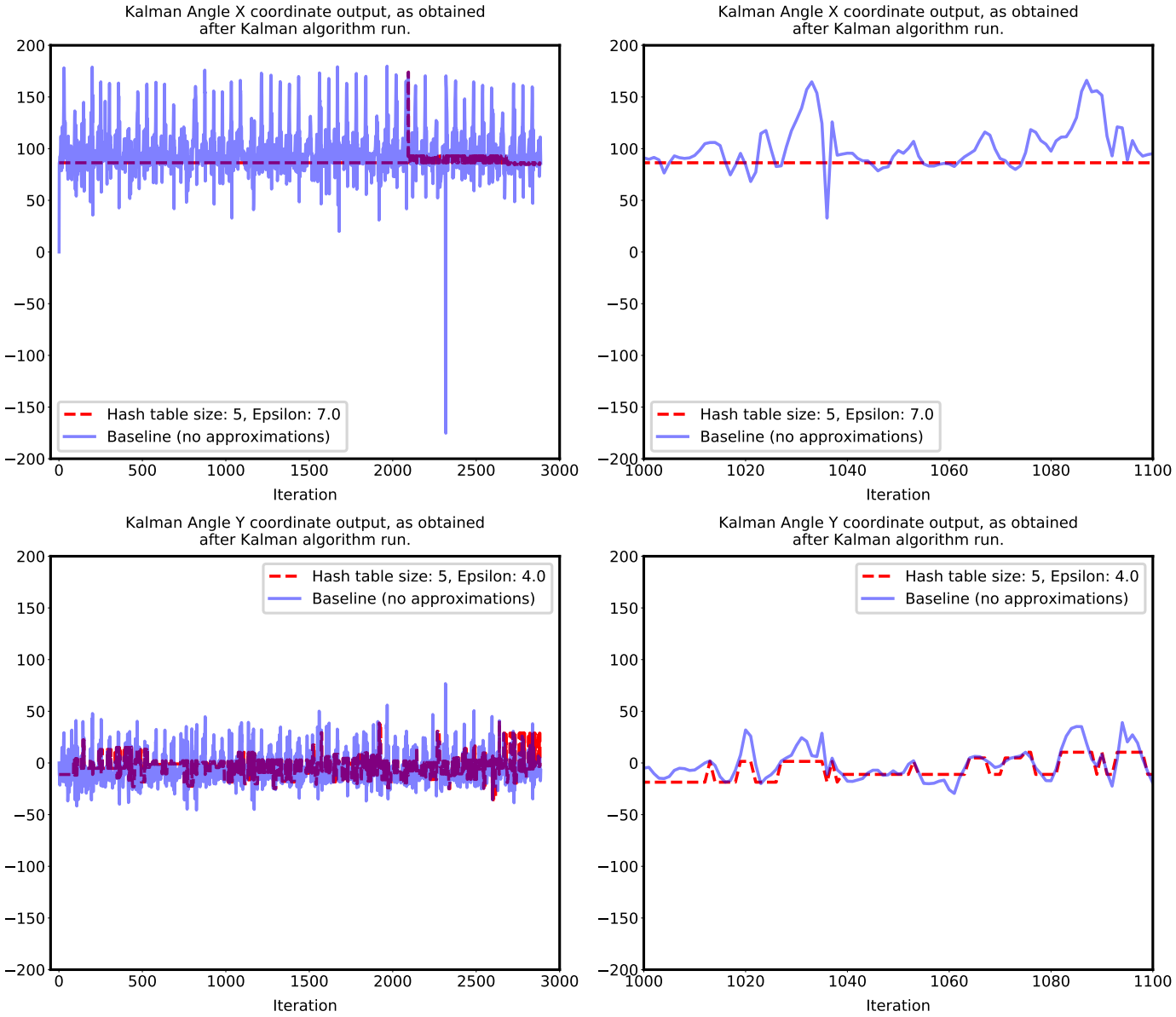

Figure 4.2: (continued) The outputs of both baseline and approximations-enabled versions of the Kalman application. On the right side are zoomed in plots of the left side.

Parameters for the end application

1. Epsilon ϵ

   Epsilon ϵ is a value the user would want to narrow down to a few selections or just one, after evaluation, and use them in the end. In comparison to previous sections, where epsilon was discussed as an *input* of the approximate computing mechanism employed, this section discusses epsilon as the most important information obtainable after running the tests as they were conducted in this thesis. By running the main application on a version working on the desktop, a user can find out almost instantly the deviation in output from attempting a large number of epsilons as parameters for the experiments. As the user becomes comfortable with selecting a few epsilons to test further, more time can be spent running energy analysis on the chosen embedded platform, as it was done through the experiments conducted for this work. From the many epsilons tried, the user can select a few or only one to use in the end application.

   The figures 4.3 all depict the trend of energy, duration and deviation over all the experiments combined, categorized by the epsilon ϵ value, which is on the x-axis. It can be observed that, as epsilon increases, the accuracy of the output data decreases drastically, reaching even more than 200% deviation as epsilon grows towards a value of 7. The energy consumption, however, becomes better following the same increase.

   More accurate results however can be desired. Wanting an average deviation under 75%, the favorable epsilons are all under 1.0. If we can be above 75% as well but under 100% for the average deviation, then the chosen epsilon can be 3.0 or under. This is not linear, however, as epsilons in-between favorable ones can have a ”jump” in average deviation that might make them undesirable. Such epsilons are ”outliers”, the relationship between the epsilon and the produced deviation not being a predictable one. In the figures 4.3, this is clearly showing at epsilon 3.0 for the hash table size `5`, or at epsilon 5.2 at hash table size `120`.

   The figures at 4.3 have vertical lines marking wherever the experiments were actually done. In between such lines, a forced linearity was obtained (albeit not to rely on to be correct). From 0.3 to 1.0, experiments were done at a 0.1 incremental steps of epsilon ϵ. That is because it was seen that the deviation average is well below 100% and trending around 50%, in that range – which is a desired effect to evaluate. Of course, in

the end the wanted outcome can only be dictated by the requirements of an application. The 50% average deviation target is an arbitrary choice considered by this project.

Both the dashed and the non-dashed (continuous) vertical lines in figures 4.3 mark experiments that were done at their respective epsilons. The continuous lines are only marked so to show that their epsilon label were also written on the x-axis in the figure. The hash table size `120` was the very first hash table size value selected to experiment with, hence it was analyzed thoroughly with many more epsilon value steps. That is why it shows a more fluctuating line along the deviation average values.

The deviation average varies the way it does due to the dynamism of attempting to store non-linear Kalman filter's outputs into a linear structure that is a hash table. The input values are the ones being hashed, and an un-evenly dispersed input trace contributes to how the hash table is being populated and used, which then affects the deviation average greatly.

The 0.3 value of epsilon ϵ is from where the energy savings comparing to the baseline are starting to be positive, even before considering the checkpointing mechanism included, as well as the system being restricted by a single capacitor. Indeed, figures 4.3 is without any such considerations yet. In a latter section *4.2.3 Improvements in energy consumption*, it will be shown that the epsilon that has a positive effect will start from a bigger value, when considering checkpointing and a capacitor are considered.

A maximum epsilon would be the maximum difference between two values of the same coordinate, and that would mean that after the first iteration ever calculated by the Kalman filter will always be reused and would produce maximum energy savings but, in return, what would be obtained are overall useless output values, forming a straight line for the resulting output coordinate in the respective output plot.

2. Hash table size We can observe that hash table sizes 20, 40, 60, 80 and 100 (and even 120 discussed later) do not have any observable difference between their figures shown at 4.3. One conclusion that can be drawn is that the entirety of the hash table, as it grows larger, is being underutilized regardless of the hash table size increase, due to the specific input trace that this study is using. The input trace tends to have values that, if scattered on a number axis individually, would show that most

would naturally form a group close to each other by a difference possibly smaller than epsilon ϵ 0.3. That is why we seem some fluctuation at values under 0.3 (a "jump") for the hash table size 120, which has a higher resolution of the experiments done, in figure from 4.3. For the rest of the hash table sizes figures – from 20 to 100 – we do not see much fluctation between one another due to the same reason. The clusters of input values would naturally use the same index and overwrite themselves in the same fashion, no matter the hash table size. Only after lowering the hash table size to `5` or `10`, some values that would otherwise index at a value bigger than `5` or `10` respectively, would now manage – as they are restricted by the now lower hash table size – to affect the plots by indexing with the same clusters mentioned to be formed, that would populate the first few indexes. More values are indexing at smaller indexes (under `5`) due to the accelerometer having values that naturally are close to 0 – as that is how an accelerometer works.

All in all, the hash table size was not a significant determinant to the increase of energy savings. This statement will be more or less applicable to different input datasets, which vary themselves for the same application and/or for different applications. Unfortunately, this study did not propose itself to use multiple input datasets, as the proof of concept of attaining energy savings using memoization was the goal. Studying more in this regard, with multiple datasets, remains then under the future work for this project.

Improvements in energy consumption

When a Kalman iteration can reuse a previous result, a power drop is obvious as shown in figures 4.4. In such a way, it is noticed how memoization helps gain energy savings depsite having overhead when results are not reused. The overhead is obvious as the blue line representing the energy use of the application with approximations has a maximum values higher than the baseline in regard to power consumption. The many power dips when there is a hit in the hash table however compensate to obtain an average energy consumption lower for the memoization mechanism. To be noted is that the lines of the plots obtained are thicker as to avoid crowdedness in them, which has the negative side effect of the power drops seemingly looking unimpactful.

Reduction in checkpointing

The results, that are closer to a real-life scenario perspective, are those shown by the barcharts 4.5 for when using flash memory and 4.6 for when using FRAM. Such graphs take into consideration the checkpointing and restoration mechanisms for the whole system, which incurs an extra energy overhead alongside the application itself, not depicted in previous figures. The overhead varies depending on the total memory occupancy of the application and approximate computing method's data structures, as well as considers the constraints that a 40 µF capacitor would entail, of which the pattern of use was explained previously in the *3 Method and implementation* chapter

1. Iterations

   It is expected that the more iterations are coming through, the better the hash table would be populated, resulting in more energy savings achieved. The figures 4.7 can be observed, where the same Kalman application run is done over a hash table size `5` multiple times while using the FRAM to checkpoint to. The runs differ in the sense that each of them are stopping at different number of iterations. The smaller hash table size gives reasonable expectations that more collisions are to happen and, so, the effect of energy savings is nullified as input-output sets overwrite each other's locations in the hash table array. However, as seen in the figures 4.3 and as it is already discussed in the section *2 Parameters for the end application*, the particular input trace used for this project forms clusters that occupy only some locations of the hash table even as the size becomes bigger. As the hash table size gets lower, however, it is suspected that at least two clusters of hashed values manage to overwrite each other's array locations in a consistent manner as the iterations are being done. That means that the reuse of a stored iteration's results is short-lived, as it is expected to be overwritten soon with another set of input-outputs corresponding to values hashed at the same index but which differ significantly. The energy savings in this case are lowered, being close to the worst case, but they are not entirely cancelled out – as seen through the fact that approximations are indeed working. In the ideal case however, a hash table of bigger size is preferred in order to prevent frequent overwrites. If the input trace would be a "lucky one" and, by selecting a proper epsilon, it would map out in a balanced manner over all the locations of the hash table to use all of them equally – then the input-output sets that would happen to overwrite each other would also do so in an equal amount of times between all locations. A

bigger hash table size is less likely to be preferred, however, as the energy overhead of checkpointing it becomes significantly bigger relative to the size (As noted in section *2 Parameters for the end application*).

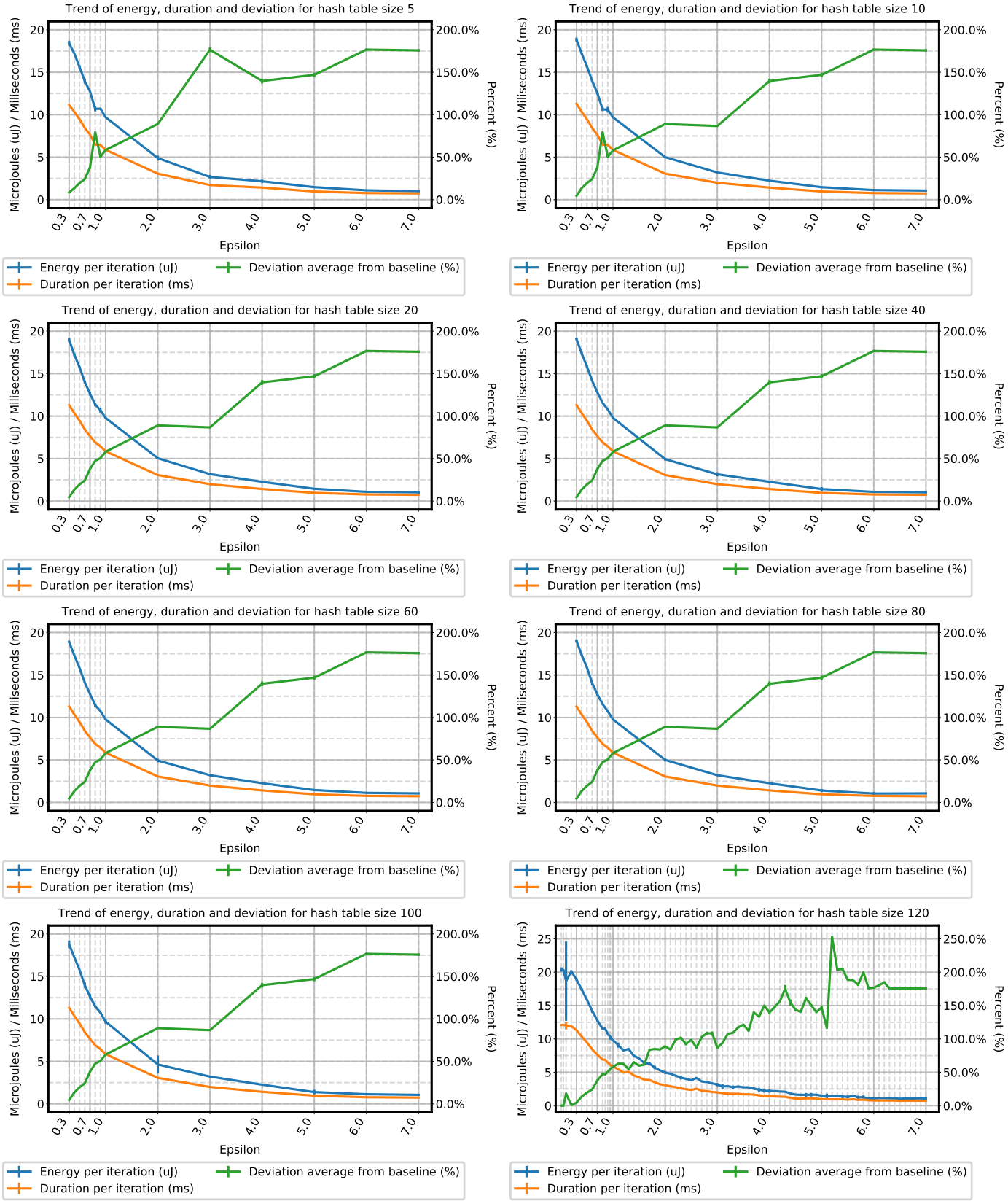

Figure 4.3: Trend of energy, duration and deviation over all hash table sizes and all iterations

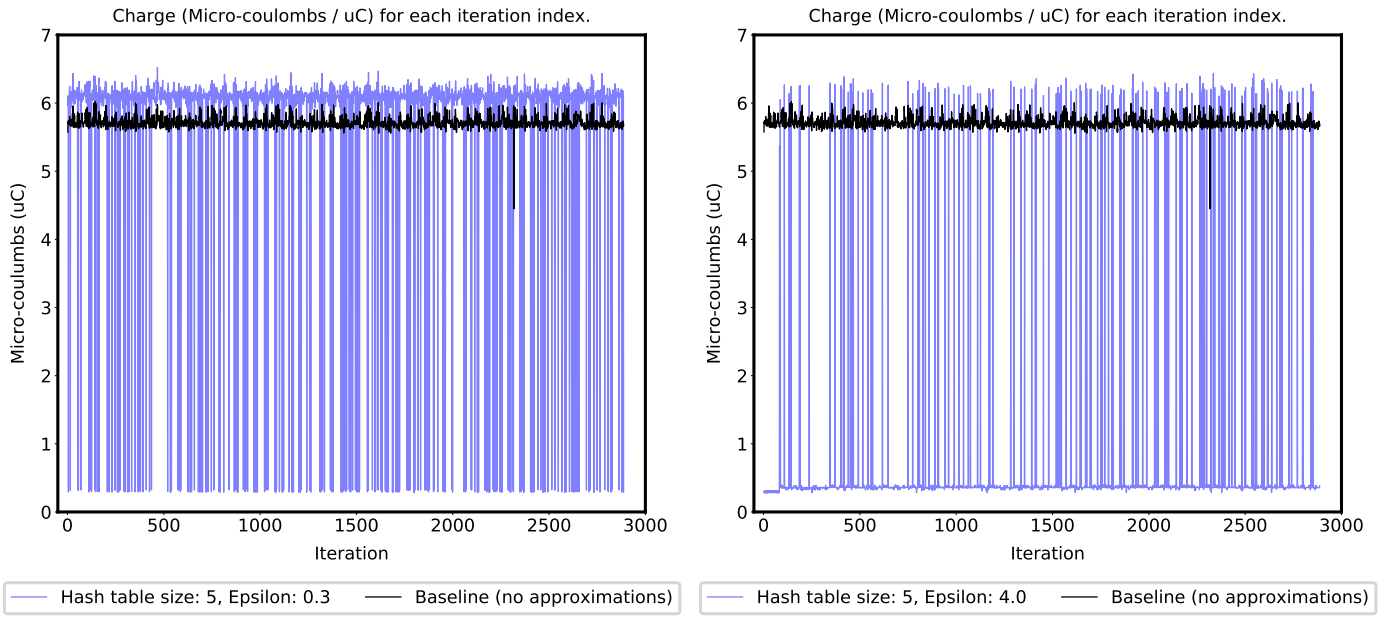

Figure 4.4: The comparison of charge curves between the baseline and approximations-enabled versions of the Kalman application

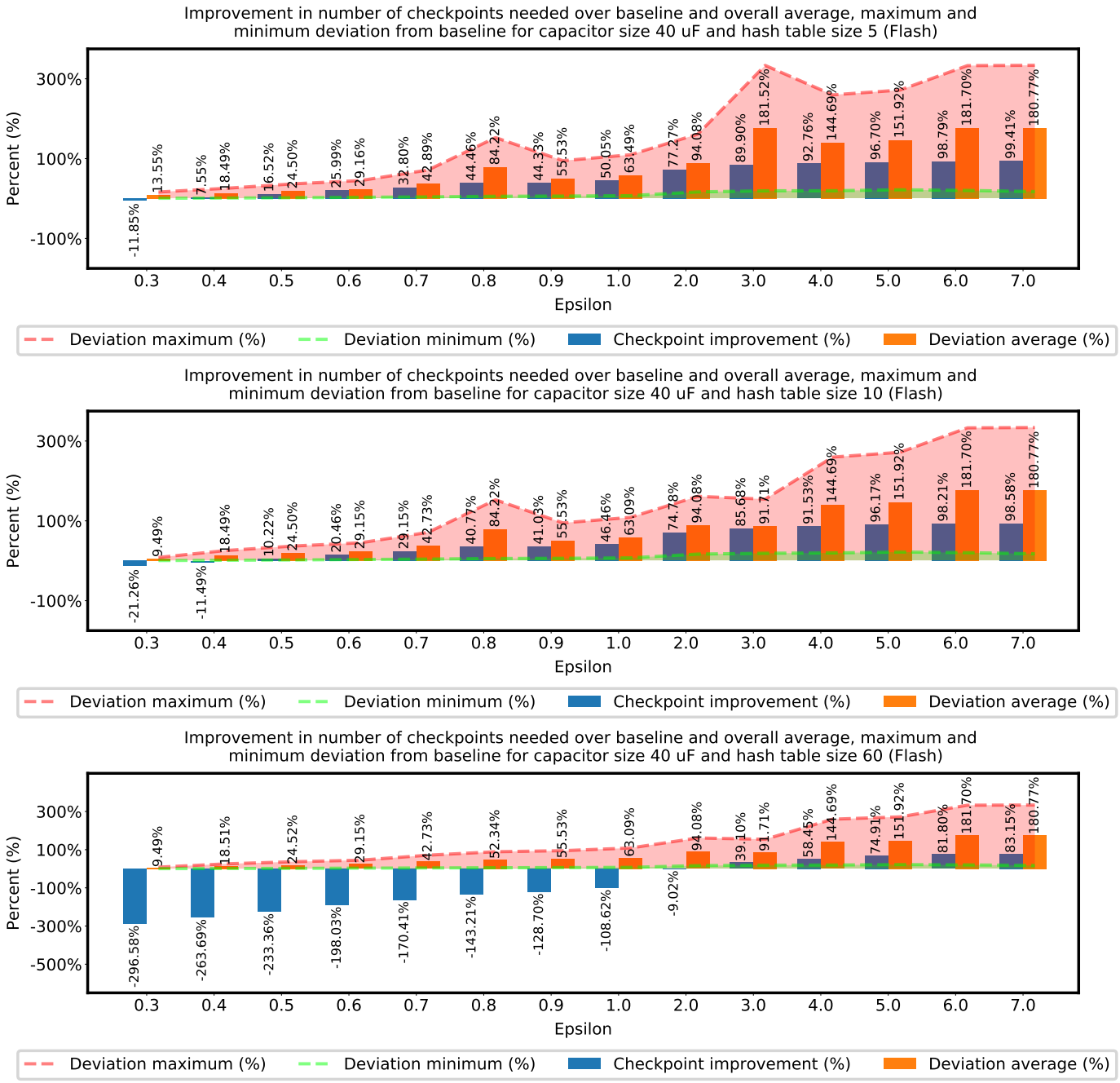

Figure 4.5: Improvements in number of checkpoints over baseline, as well as overall average, maximum and minimum deviation from baseline, for capacitor size 40 µF and hash table sizes 5-60, considering checkpointing to *flash* memory

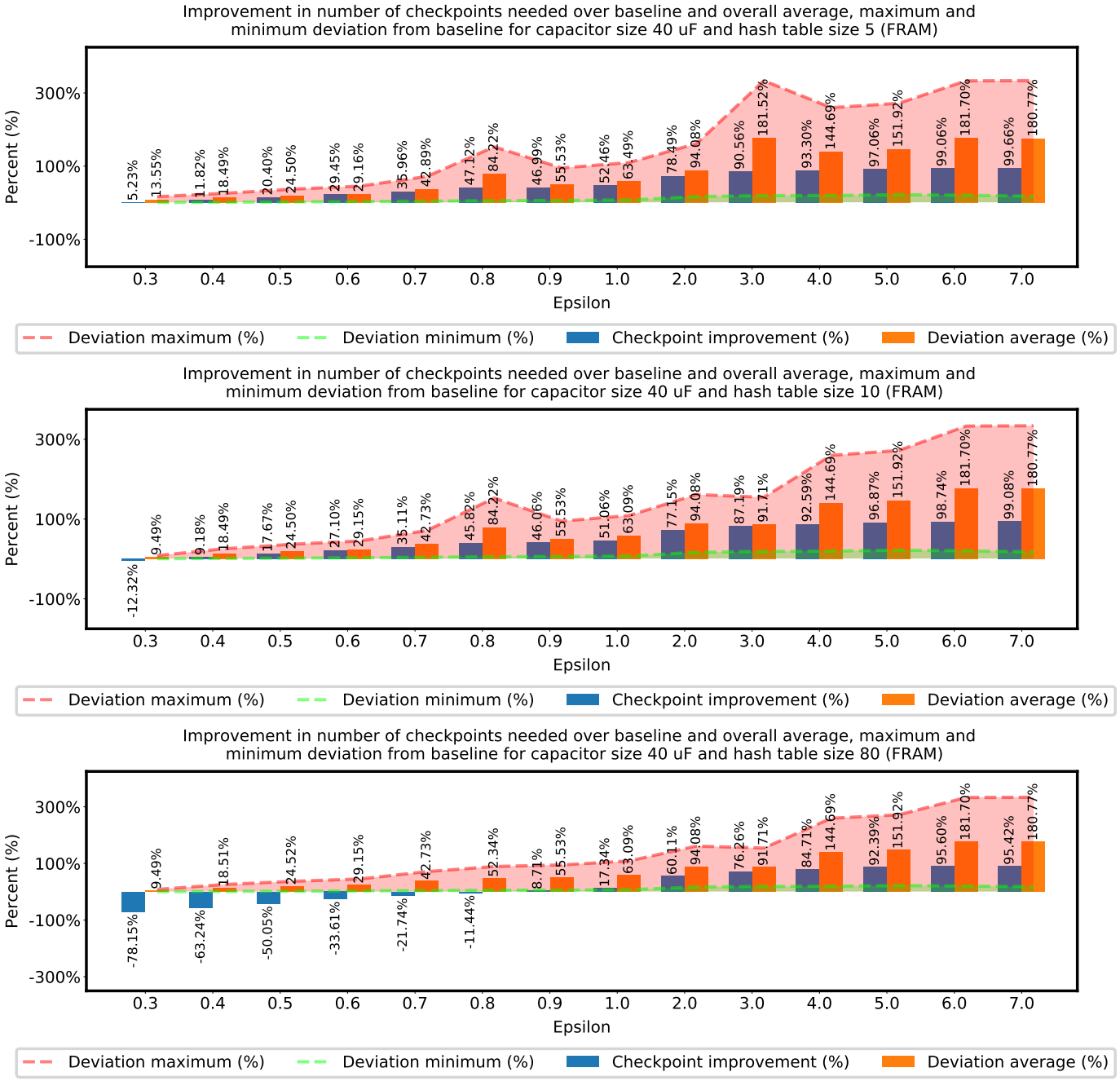

Figure 4.6: Improvements in number of checkpoints over baseline, as well as overall average, maximum and minimum deviation from baseline, for capacitor size 40 µF and hash table sizes 5-60, considering checkpointing to *FRAM* memory

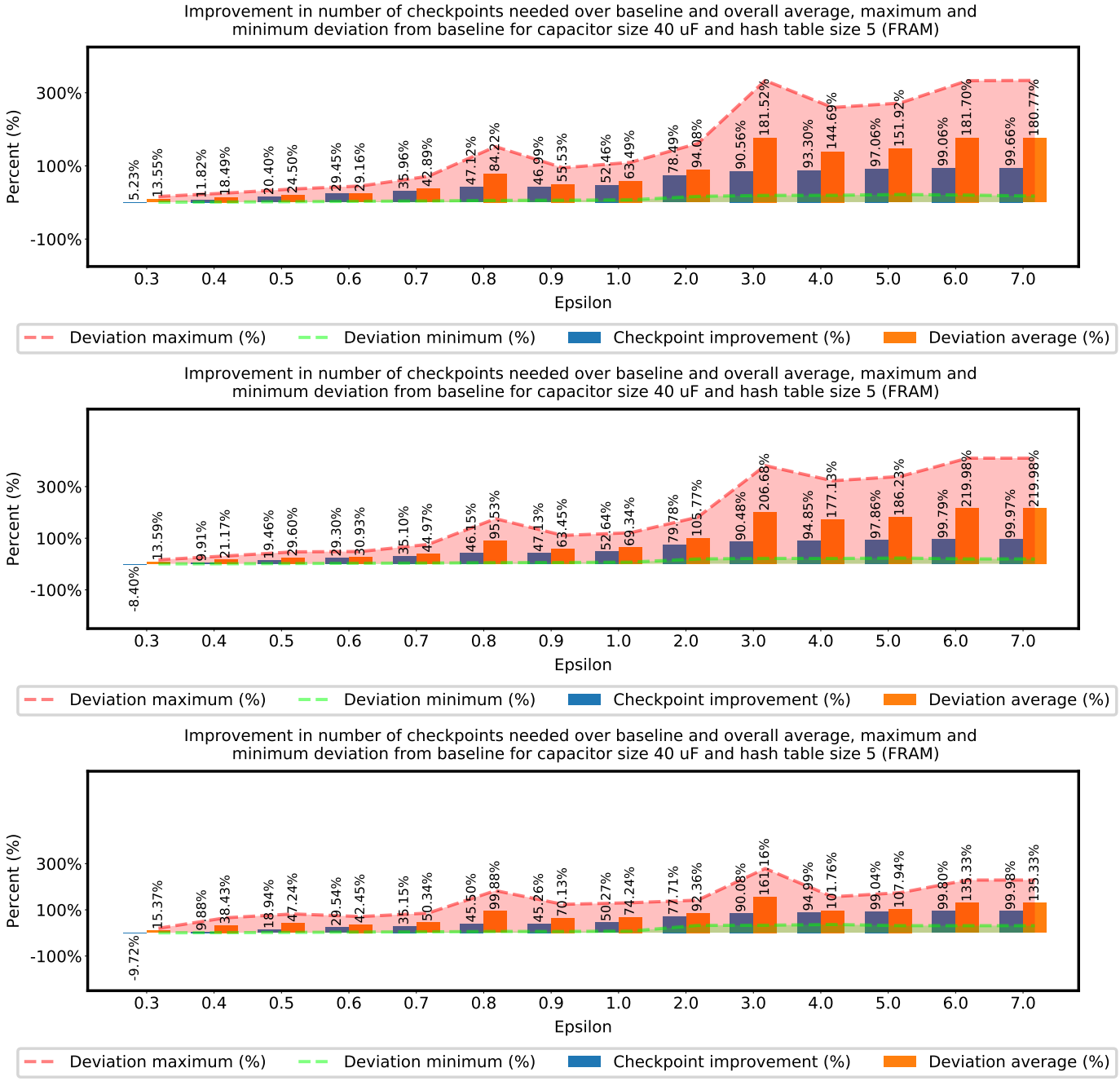

Figure 4.7: Improvements in number of checkpoints over baseline, as well as overall average, maximum and minimum deviation from baseline, for capacitor size 40 µF and hash table sizes 5-60, considering checkpointing to *FRAM* memory. The number of iterations executed however are being limited to: *all iterations*, *1500 iterations* and *500 iterations*

Chapter 5

Conclusion

The innovative technology that are transiently powered systems come with the cost of discontinued runtimes of the application, as power losses are frequent with energy harvesting. To save computation progress, a checkpointing mechanism that also saves the RAM into non-volatile memory (e.g. flash) is usually employed in the case of such systems. Having all memories on such embedded platforms of non-volatile type would discard the overhead that the checkpointing method would entail, however that would make the computation slower and more financial costs would have to be supported, as such technology is new. This thesis contributes with a consolidation of the current age embedded systems to become transiently powered systems, by building on top of the checkpointing mechanism with approximate computing.

The memoization technique that was applied shows promise – being the reuse of results obtained by the main application, to then be reused if inputs are deemed similar to what was received in the past. Such results are saved in a hash table for a fast access when accessing them. The variable *epsilon* ϵ is introduced which, through its adjustment, lowers the accuracy of the output but raises energy savings and vice versa. Through the experiments conducted, the improvement over the number of checkpoints done by the application employing approximations, which translates to better energy consumption, is observed. By using approximations, the application manages to have improvements over the baseline version of around 32% to 44% (for epsilons ϵ equal to 0.7 and 0.9 respectively, and for a hash table size of 5). With such improvements, the average deviation of the outputs from the 100% accurate values are kept at 42% to 55%.

However, the work is restricted to one single trace and one application, being the Kalman filter. The experiments done in this work therefore can not

predict with certainty if energy improvement without drastic accuracy loss is feasible for other types of applications, when using approximate computing. What this work shows, however, is indeed a proof of concept that approximate computing and, in the case of this work, memoization, can achieve energy savings to complement transiently powered systems.

5.1 Future work

The approximate computing method described in this thesis could further benefit if the checkpointing mechanism would include checking for changes beforehand before committing parts of the RAM to flash memory. As such, then it is then possible to explore bigger hash tables, as not all of the hash table is being checkpointed all the time. This means that energy overhead for checkpointing a bigger hash table size would be closer to the overhead needed for a smaller hash table size, that is currently recommended. Nevertheless, by checkpointing only changes from the previous saved state, the overall approximate computing solution would benefit from a bigger hash table with lesser performance downsides. However, this would still need testing to be proven feasible.

Even more, a popular solution would be to create and populate a bigger hash table before deploying the application, to which the application can only refer instead of also write to. The hash table would then be saved directly in non-volatile memory, and would not then be need to be included in checkpoints. However, there would be a decrease in the speed of accessing the table, due to now conducting flash read/write operations instead of the same operations on RAM – which are faster. In contrast, there could be energy overhead eliminated due to checkpointing much less memory, which could compensate. This tradeoff would however have to be investigated.

Lastly, during the implementation and experimentation process of this work, the hash function was ported to a simple Python script and then evaluated in order to get the best parameters to try out first and/or hard code less relevant parameters (e.g. mod precision). Through this practice, it was essentially demonstrated that it is possible to build a design-time application that would run on the user's machine and could offer suggestions for the best parameters to be used for the approximations. This would be done by importing a sample of the input data that the user application could meet in real-life scenarios, and the user could then provide preferences – such as what level of accuracy would be desired for the situation and other options. The suggestion algorithm would then offer the user certain epsilon ϵ values to test for energy consumption.

Bibliography

[1] Mari Carmen Domingo.
"An Overview of the Internet of Things for People with Disabilities".
In: *Journal of Network and Computer Applications*. Simulation and Testbeds 35.2 (Mar. 2012), pp. 584–596. ISSN: 1084-8045.
DOI: 10.1016/j.jnca.2011.10.015.

[2] Antonis Tzounis et al. "Internet of Things in Agriculture, Recent Advances and Future Challenges".
In: *Biosystems Engineering* 164 (Dec. 2017), pp. 31–48.
ISSN: 1537-5110.
DOI: 10.1016/j.biosystemseng.2017.09.007.

[3] Yuehong Yin et al.
"The Internet of Things in Healthcare: An Overview". In: *Journal of Industrial Information Integration* 1 (Mar. 2016), pp. 3–13.
ISSN: 2452-414X. DOI: 10.1016/j.jii.2016.03.004.

[4] F. Shrouf, J. Ordieres, and G. Miragliotta. "Smart Factories in Industry 4.0: A Review of the Concept and of Energy Management Approached in Production Based on the Internet of Things Paradigm".
In: *2014 IEEE International Conference on Industrial Engineering and Engineering Management*. Dec. 2014, pp. 697–701.
DOI: 10.1109/IEEM.2014.7058728.

[5] T. Stock and G. Seliger.
"Opportunities of Sustainable Manufacturing in Industry 4.0".
In: *Procedia CIRP*. 13th Global Conference on Sustainable Manufacturing – Decoupling Growth from Resource Use 40 (Jan. 2016), pp. 536–541. ISSN: 2212-8271.
DOI: 10.1016/j.procir.2016.01.129.

[6] J. A. Guerrero-ibanez, S. Zeadally, and J. Contreras-Castillo.
"Integration Challenges of Intelligent Transportation Systems with

Connected Vehicle, Cloud Computing, and Internet of Things Technologies".
In: *IEEE Wireless Communications* 22.6 (Dec. 2015), pp. 122–128. ISSN: 1536-1284. DOI: 10.1109/MWC.2015.7368833.

[7] Peter M. Corcoran. "Third Time Is the Charm - Why the World Just Might Be Ready for the Internet of Things This Time Around".
In: *arXiv:1704.00384 [cs]* (Apr. 2017).
URL: http://arxiv.org/abs/1704.00384 (visited on 05/09/2019).

[8] Ericsson. *Ericsson Mobility Report 2019*. Jan. 2019. URL: https://www.ericsson.com/assets/local/mobility-report/documents/2019/ericsson-mobility-report-world-economic-forum.pdf (visited on 05/05/2019).

[9] Rob van der Meulen. *Gartner Says 6.4 Billion Connected "Things" Will Be in Use in 2016, Up 30 Percent From 2015*. Feb. 2017.
URL: https://www.gartner.com/en/newsroom/press-releases/2015-11-10-gartner-says-6-billion-connected-things-will-be-in-use-in-2016-up-30-percent-from-2015 (visited on 05/05/2019).

[10] J. Lee, C. Chuang, and C. Shen.
"Applications of Short-Range Wireless Technologies to Industrial Automation: A ZigBee Approach". In: *2009 Fifth Advanced International Conference on Telecommunications*. May 2009, pp. 15–20. DOI: 10.1109/AICT.2009.9.

[11] O A Gracia Osorio, Brayan S Reyes Dazai, and Octavio J Salcedo.
"Comparative Study of Performance for 804.15.4 ZigBee and 6LoWPAN Protocols". In: *SOFSEM (Student Research Forum Papers/Posters)* (2016), pp. 59–71.

[12] Cisco. *At a Glance - Internet of Things*. 2016.
URL: https://www.cisco.com/c/dam/en/us/products%20/collateral/se/internet-of-things/at-a-glance-c45-731471.pdf (visited on 05/03/2019).

[13] A. Nordrum. "The Internet of Fewer Things".
In: *IEEE Spectrum* 53.10 (Oct. 2016), pp. 12–13. ISSN: 0018-9235. DOI: 10.1109/MSPEC.2016.7572524.

[14] United Nations. *World Population Prospects (The 2017 Revision) - Key Findings and Advance Tables*. 2017. URL: https://population.un.org/wpp/Publications/Files/WPP2017_KeyFindings.pdf (visited on 08/07/2019).

[15] Intel.com. *The Looming IoT Energy Drain*. Apr. 2015. URL: https://newsroom.intel.com/editorials/iot-sensor-energy-harvesting/ (visited on 05/08/2019).

[16] IPCC. *AR5 Synthesis Report: Climate Change 2014*. Tech. rep. 2014. URL: https://www.ipcc.ch/report/ar5/syr/ (visited on 05/12/2019).

[17] Europa.eu. *2030 Energy Strategy*. Oct. 2014. URL: https://ec.europa.eu/energy/en/topics/energy-strategy-and-energy-union/2030-energy-strategy (visited on 05/12/2019).

[18] Steven Max Patterson. *4 Reasons Cisco's IoT Forecast Is Right, and 2 Why It's Wrong*. Apr. 2017. URL: https://www.networkworld.com/article/3187891/4-reasons-ciscos-iot-forecast-is-right-and-2-why-its-wrong.html (visited on 05/03/2019).

[19] Andrew Burger. *Report Forecasts Four Drivers for Data Center Growth*. Feb. 2016. URL: https://www.telecompetitor.com/report-forecasts-four-drivers-for-data-center-growth/ (visited on 05/09/2019).

[20] J. Baliga et al. "Energy Consumption in Wired and Wireless Access Networks". In: *IEEE Communications Magazine* 49.6 (June 2011), pp. 70–77. ISSN: 0163-6804. DOI: 10.1109/MCOM.2011.5783987.

[21] Perry Lea. *Internet of Things for Architects: Architecting IoT Solutions by Implementing Sensors, Communication Infrastructure, Edge Computing, Analytics, and Security*. Packt Publishing Ltd, Jan. 2018. ISBN: 978-1-78847-574-7.

[22] Texas Instruments. *Texas Instruments CC2650 Wireless MCU Datasheet*. July 2016. URL: http://www.ti.com/product/CC2650 (visited on 05/30/2019).

[23] Eurostat. *Electricity Production, Consumption and Market Overview*. June 2019.
URL: https://ec.europa.eu/eurostat/statistics-explained/index.php/Electricity_production,_consumption_and_market_overview (visited on 05/12/2019).

[24] Mia Romare and Lisbeth Dahllöf. *The Life Cycle Energy Consumption and Greenhouse Gas Emissions from Lithium-Ion Batteries*. 2017.
URL: https://www.ivl.se/download/18.5922281715bdaebede9559/1496046218976/C243+The+life+cycle+energy+consumption+and+CO2+emissions+from+lithium+ion+batteries+.pdf.

[25] mCube. *MC3230, MC32323-Axis Accelerometer Datasheet*. 2014.
URL: https://mcubemems.com/wp-content/uploads/2014/10/MC3230_2-Datasheet-APS-048-0007v1.6.pdf (visited on 05/31/2019).

[26] Cisco.com.
Cisco Global Cloud Index: Forecast and Methodology, 2016–2021.
Novermber 19, 2018.
URL: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.html (visited on 05/30/2019).

[27] Hanna Pihkola et al. "Evaluating the Energy Consumption of Mobile Data Transfer—From Technology Development to Consumer Behaviour and Life Cycle Thinking".
In: *Sustainability* 10.7 (July 2018), p. 2494. ISSN: 2071-1050.
DOI: 10.3390/su10072494.

[28] WhatIs5G.info. *Energy Consumption from the Internet of Things and Wireless Technology*. en-US. 2018.
URL: https://whatis5g.info/energy-consumption/ (visited on 05/03/2019).

[29] Kris de Decker. *Why We Need a Speed Limit for the Internet*.
Oct. 2015. URL:
https://solar.lowtechmagazine.com/2015/10/can-the-internet-run-on-renewable-energy.html (visited on 05/31/2019).

[30] Dexter Johnson.
The 5G Dilemma: More Base Stations, More Antennas—Less Energy?
Oct. 2018. URL:
https://spectrum.ieee.org/energywise/telecom/wireless/will-increased-energy-consumption-be-the-achilles-heel-of-5g-networks (visited on 06/01/2019).

[31] OffGridEnergyIndependence.com.
World's First Solar-Powered Family Car. July 2013.
URL: https://www.offgridenergyindependence.com/articles/5611/worlds-first-solar-powered-family-car (visited on 06/10/2019).

[32] SolarTeamEidhoven.nl. *Stella - The Solar Powered Family Car*. 2019.
URL: https://solarteameindhoven.nl/stella-vie/stella/ (visited on 06/10/2019).

[33] Graham Gobieski, Nathan Beckmann, and Brandon Lucia.
"Intelligence Beyond the Edge: Inference on Intermittent Embedded Systems". In: *arxiv.org/abs/1810.07751* (Sept. 2018).
arXiv: 1810.07751.

[34] C. Pan, M. Xie, and J. Hu. "Maximize Energy Utilization for Ultra-Low Energy Harvesting Powered Embedded Systems".
In: *2017 IEEE 23rd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. Aug. 2017, pp. 1–6. DOI: 10.1109/RTCSA.2017.8046325.

[35] Gautier Berthou et al. *Sytare: A Lightweight Kernel for NVRAM-Based Transiently-Powered Systems*. Dec. 2018. URL:
https://hal.archives-ouvertes.fr/hal-01954979 (visited on 06/11/2019).

[36] 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)* (2019), pp. 211–223.

[37] N. A. Bhatti and L. Mottola. "HarvOS: Efficient Code Instrumentation for Transiently-Powered Embedded Sensing".
In: *2017 16th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. Apr. 2017, pp. 209–220.

[38] Benjamin Ransford, Jacob Sorber, and Kevin Fu. “Mementos: System Support for Long-Running Computation on RFID-Scale Devices”. In: *ACM SIGARCH Computer Architecture News* 39 (2011), pp. 159–170.

[39] D. Balsamo et al. “Hibernus: Sustaining Computation During Intermittent Supply for Energy-Harvesting Systems”. In: *IEEE Embedded Systems Letters* 7.1 (Mar. 2015), pp. 15–18. ISSN: 1943-0663. DOI: 10.1109/LES.2014.2371494.

[40] Matthew Hicks and Joel Van Der Woude. “Intermittent Computation without Hardware Supportor Programmer Intervention”. In: *the Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI ’16)* (Nov. 2016), p. 17.

[41] Matthew Hicks. “Clank: Architectural Support for Intermittent Computation”. In: *Proceedings of the 44th Annual International Symposium on Computer Architecture - ISCA ’17*. ACM Press, 2017, pp. 228–240. ISBN: 978-1-4503-4892-8. DOI: 10.1145/3079856.3080238.

[42] Naveed Anwar Bhatti and Luca Mottola. “Efficient State Retention for Transiently-Powered Embedded Sensing”. In: *Proceedings of the 2016 International Conference on Embedded Wireless Systems and Networks*. Junction Publishing, 2016, pp. 137–148. ISBN: 978-0-9949886-0-7. URL: http://dl.acm.org/citation.cfm?id=2893711.2893731 (visited on 01/15/2019).

[43] Sparsh Mittal. “A Survey of Techniques for Approximate Computing”. In: *ACM Computing Surveys (CSUR)* 48.4 (May 2016). Article No. 62. URL: https://dl.acm.org/citation.cfm?id=2893356 (visited on 02/07/2019).

[44] T. Yeh et al. “The Art of Deception: Adaptive Precision Reduction for Area Efficient Physics Acceleration”. In: *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*. Dec. 2007, pp. 394–406. DOI: 10.1109/MICRO.2007.9.

[45] Stelios Sidiroglou-Douskos et al. “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*. ESEC/FSE ’11. ACM, 2011, pp. 124–134. ISBN: 978-1-4503-0443-6. DOI: 10.1145/2025113.2025133.

[46] Haskell.org. *Haskell Functional Programming Language - Homepage*. 2014.
URL: https://www.haskell.org/ (visited on 06/15/2019).

[47] Harald Søndergaard and Peter Sestoft.
"Referential Transparency, Definiteness and Unfoldability".
In: *Acta Informatica* 27.6 (May 1990), pp. 505–517. ISSN: 1432-0525.
DOI: 10.1007/BF00277387.

[48] Melda Ulusoy.
Understanding Kalman Filters, Part 1: Why Use Kalman Filters?
Jan. 2017. URL:
https://se.mathworks.com/videos/understanding-kalman-filters-part-1-why-use-kalman-filters--1485813028675.html (visited on 06/10/2019).

[49] Arthur Carcano. *Derive Yourself a Kalman Filter*.
URL: https://ngr.yt/blog/2019-04-10-kalman.html
(visited on 06/10/2019).

[50] TKJElectronics.
Kalman Filter on GitHub - TKJElectronics/Kalman Filter. 2012. URL:
https://github.com/TKJElectronics/KalmanFilter
(visited on 06/10/2019).

[51] Seulki Lee et al. "Intermittent Learning: On-Device Machine Learning on Intermittently Powered System".
In: *arXiv preprint arXiv:1904.09644* (2019).
URL: https://128.84.21.199/abs/1904.09644.

[52] Merriam-Webster.com. *Dictionary Definition of IDEMPOTENT*.
URL: https://www.merriam-webster.com/dictionary/idempotent (visited on 06/15/2019).

[53] K. Ma et al. "IAA: Incidental Approximate Architectures for Extremely Energy-Constrained Energy Harvesting Scenarios Using IoT Nonvolatile Processors".
In: *IEEE Micro* 38.4 (July 2018), pp. 11–19. ISSN: 0272-1732.
DOI: 10.1109/MM.2018.043191121.

[54] Dragos Perju and Erik Wouters. *Pydgilib*. June 2019.
URL: https://github.com/EWouters/pydgilib (visited on 06/24/2019).

[55] Microchip. *Dgilib.Dll Download Page*.
URL: https://www.microchip.com/developmenttools/ProductDetails/ATPOWERDEBUGGER (visited on 06/19/2019).

[56] Microsoft.com. *Memory Limits for Windows and Windows Server Releases - Windows Applications*. May 2018.
URL: https://docs.microsoft.com/en-us/windows/desktop/memory/memory-limits-for-windows-releases (visited on 06/19/2019).

[57] Texas Instruments. *Texas Instruments MSP 430 Datasheet*.
URL: http://www.ti.com/lit/ds/symlink/msp430g2553.pdf (visited on 06/19/2019).

[58] ARM.com. *ARM Cortex-M23 Processor Technical Reference Manual*. 2016. URL: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0550c/cortex_m23_r1p0_technical_reference_manual_DDI0550C_en.pdf.

[59] Bill Giovino. *Microchip Introduces SAM-L10 and SAM-L11 Microcontrollers with TrustZone and picoPower*. June 2018. URL: http://microcontroller.com/news/Microchip_SAM-L10_SAM-L11.asp (visited on 06/19/2019).

[60] Mohammad Malekzadeh.
MotionSense Dataset (Time-Series Data Generated by Smartphone's Sensors: Accelerometer and Gyroscope) - Github. June 2019.
URL: https://github.com/mmalekzadeh/motion-sense (visited on 06/17/2019).

[61] Mohammad Malekzadeh et al.
"Protecting Sensory Data against Sensitive Inferences".
In: *Proceedings of the 1st Workshop on Privacy by Design in Distributed Systems - W-P2DS'18* (2018), pp. 1–6.
DOI: 10.1145/3195258.3195260. arXiv: 1802.07802.

[62] Mohammad Malekzadeh et al. "Mobile Sensor Data Anonymization".
In: *Proceedings of the International Conference on Internet of Things Design and Implementation - IoTDI '19* (2019), pp. 49–58.
DOI: 10.1145/3302505.3310068. arXiv: 1810.11546.

[63] F. Kuncan. “A Novel Approach for Activity Recognition with Down-Sampling 1D Local Binary Pattern Features”. In: *Advances in Electrical and Computer Engineering* 19.1 (Feb. 2019), pp. 35–44. ISSN: 1582-7445. DOI: 10.4316/AECE.2019.01005.

[64] Microchip.com. *Microchip - SAM L10/L11 Family Datasheet*. 2019. URL: http://ww1.microchip.com/downloads/en/DeviceDoc/SAM-L10-L11-Family-Data-Sheet-DS60001513C.pdf.

Appendix A

Trend of energy, duration and deviation of all hash table sizes over 1500 iterations

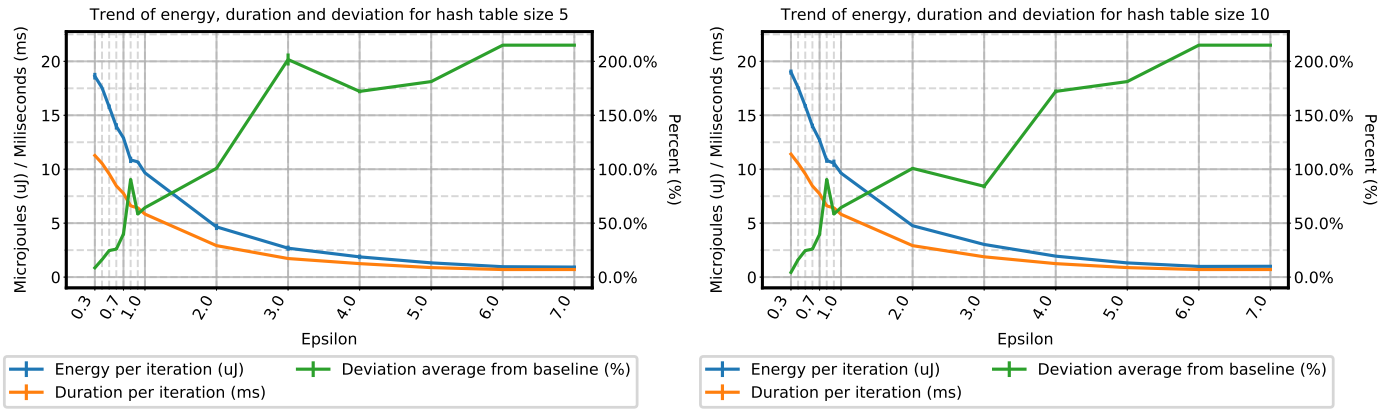

Figure A.1: Trend of energy, duration and deviation of all hash table sizes and limited to 1500 iterations

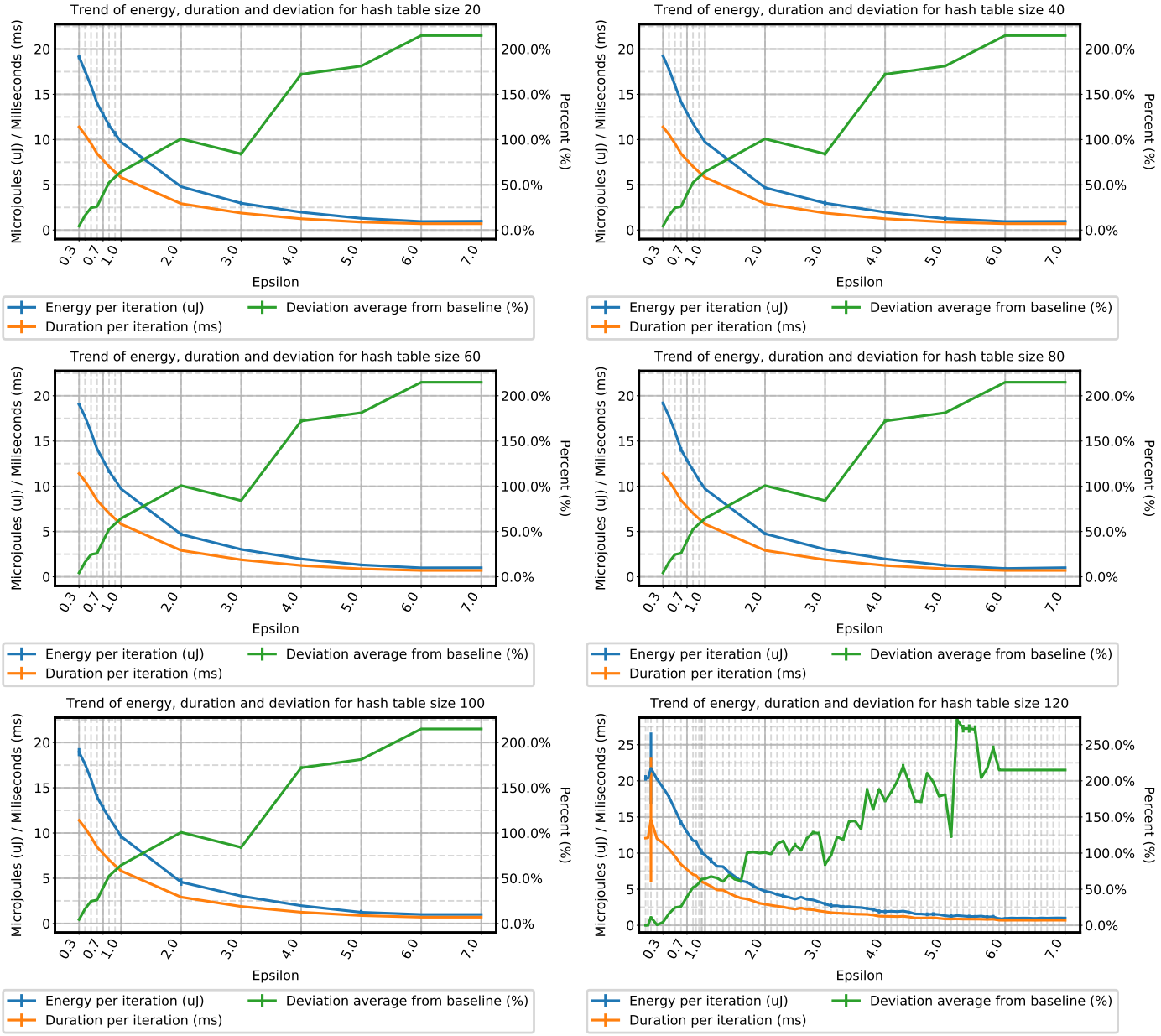

Figure A.1: (continued) Trend of energy, duration and deviation of all hash table sizes and limited to 1500 iterations

Appendix B

Trend of energy, duration and deviation of all hash table sizes over 500 iterations

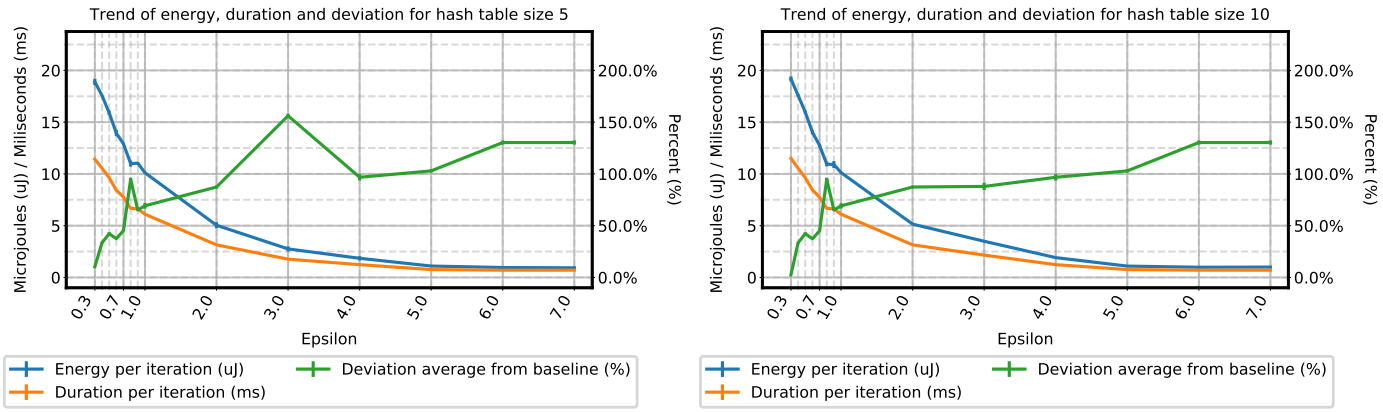

Figure B.1: Trend of energy, duration and deviation of all hash table sizes and limited to 500 iterations

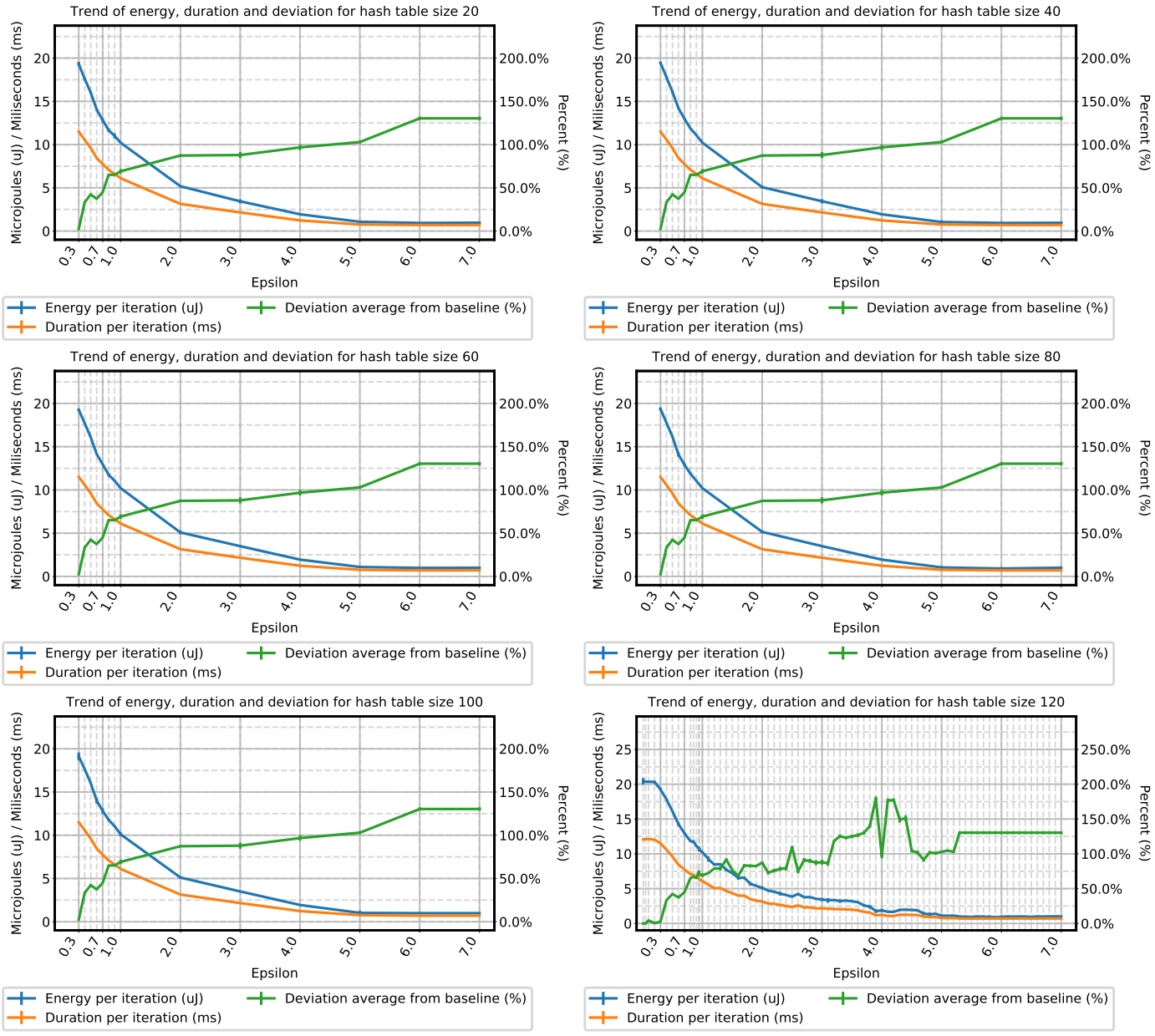

Figure B.1: (continued) Trend of energy, duration and deviation of all hash table sizes and limited to 500 iterations

Appendix C

Number of checkpoints improvement and deviation over baseline, for $40\,\mu F$ and using flash memory

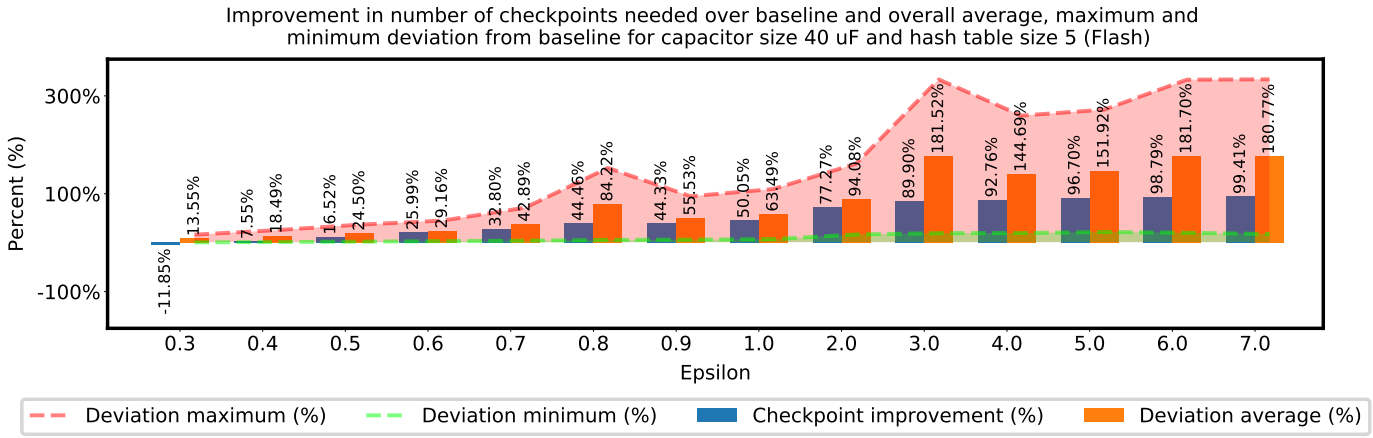

Figure C.1: Improvements in number of checkpoints over baseline, as well as overall average, maximum and minimum deviation from baseline, for capacitor size 40 µF and hash table sizes 5-60, considering checkpointing to *flash* memory

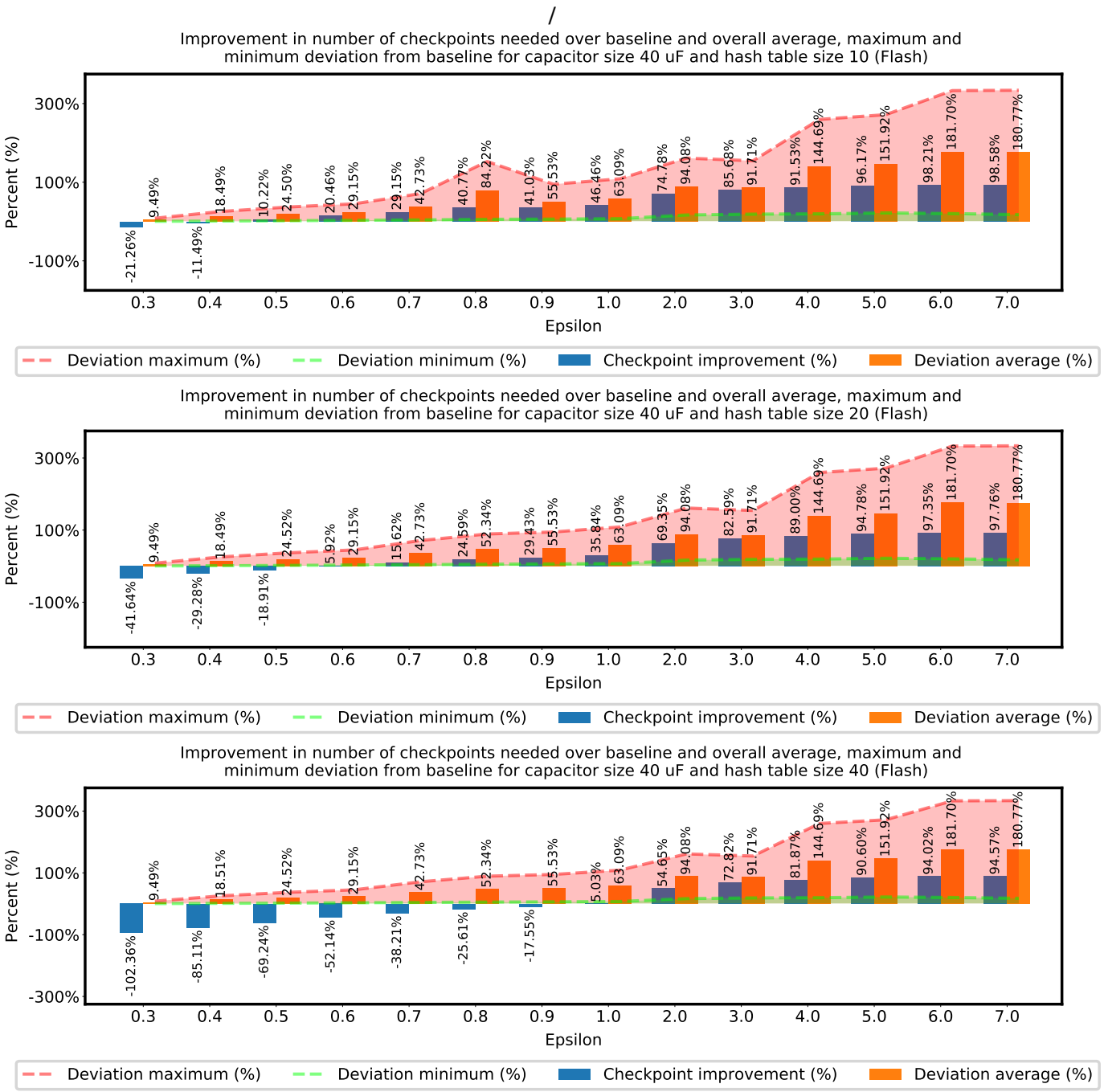

Figure C.1: (continued) Improvements in number of checkpoints over baseline, as well as overall average, maximum and minimum deviation from baseline, for capacitor size 40 µF and hash table sizes 5-60, considering checkpointing to *flash* memory

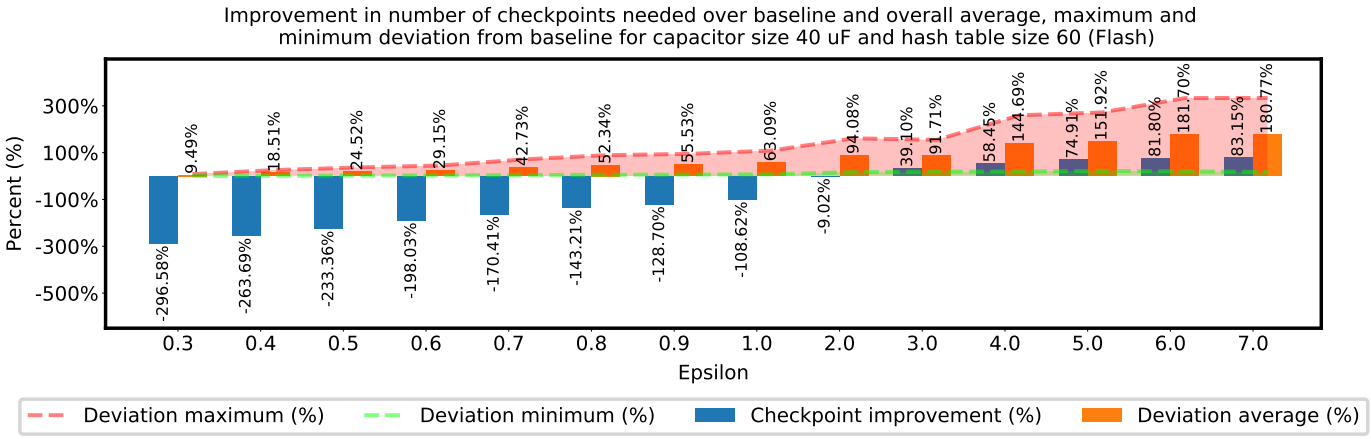

Figure C.1: (continued) Improvements in number of checkpoints over baseline, as well as overall average, maximum and minimum deviation from baseline, for capacitor size 40 µF and hash table sizes 5-60, considering checkpointing to *flash* memory

Appendix D

Number of checkpoints improvement and deviation over baseline, for $40\,\mu F$ and using FRAM memory

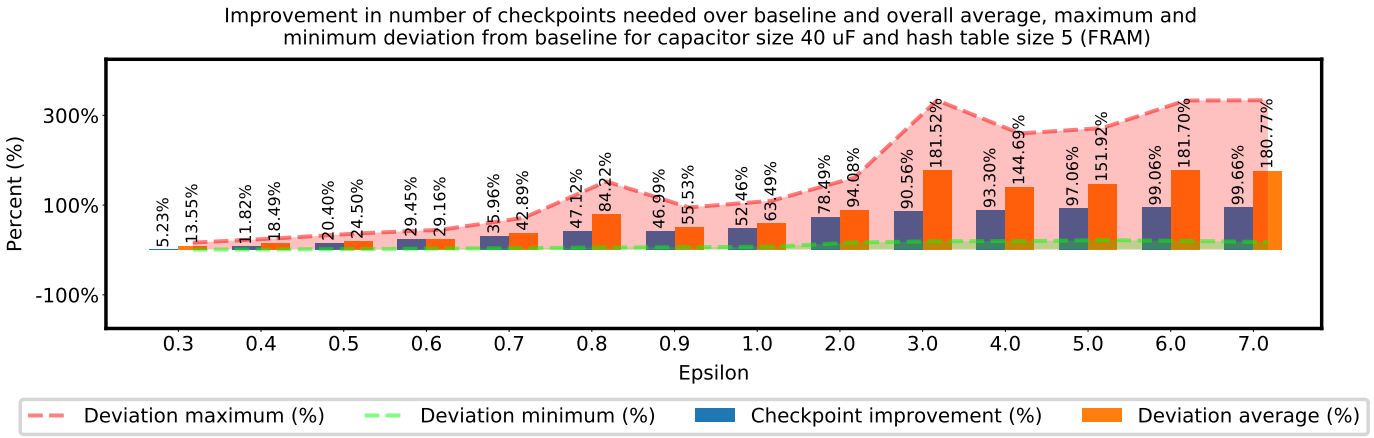

Figure D.1: Improvements in number of checkpoints over baseline, as well as overall average, maximum and minimum deviation from baseline, for capacitor size $40\,\mu F$ and all hash table sizes, considering checkpointing to *FRAM* memory

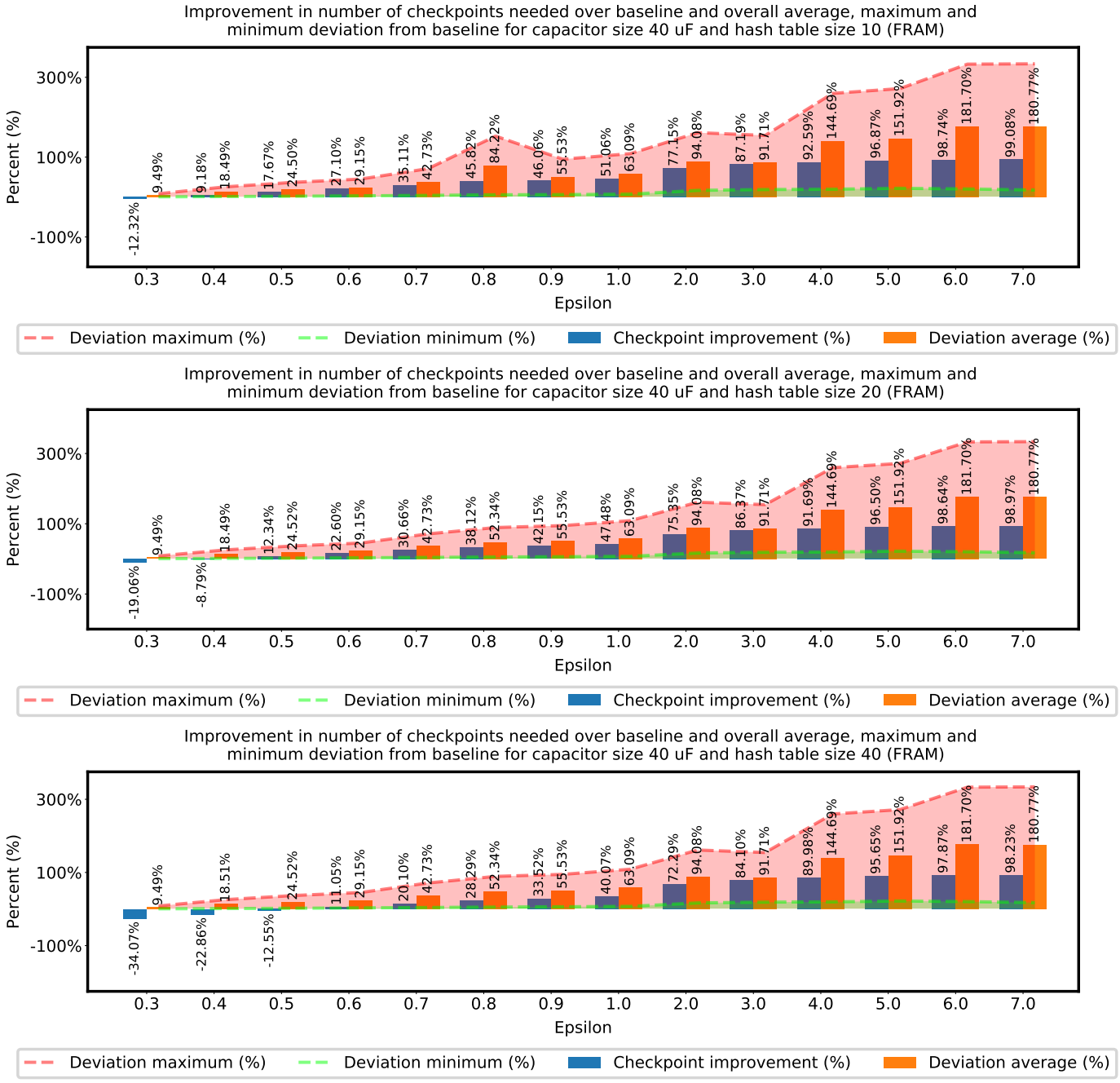

Figure D.1: (continued) Improvements in number of checkpoints over baseline, as well as overall average, maximum and minimum deviation from baseline, for capacitor size 40 µF and all hash table sizes, considering checkpointing to *FRAM* memory

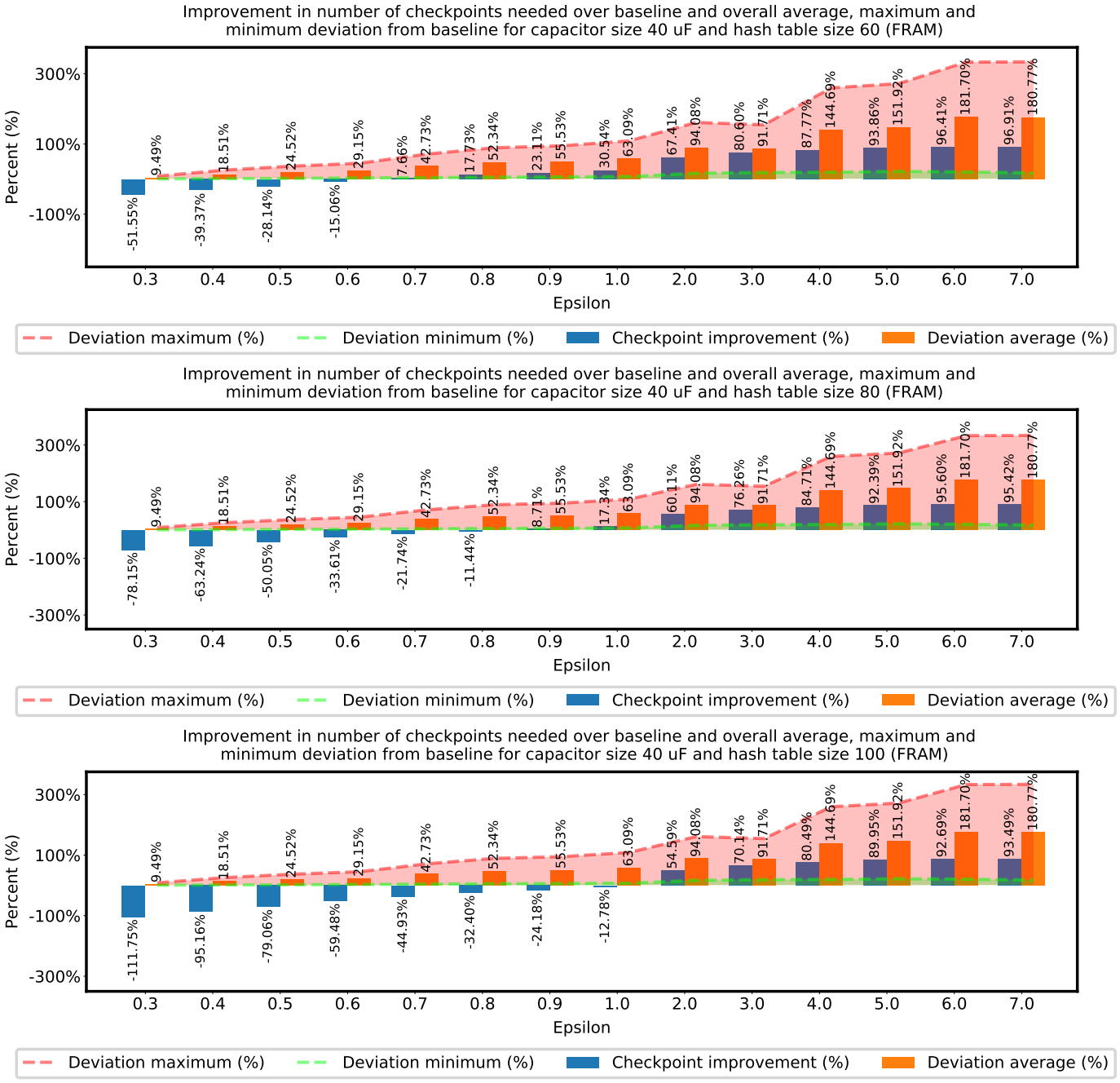

Figure D.1: (continued) Improvements in number of checkpoints over baseline, as well as overall average, maximum and minimum deviation from baseline, for capacitor size 40 µF and all hash table sizes, considering checkpointing to *FRAM* memory

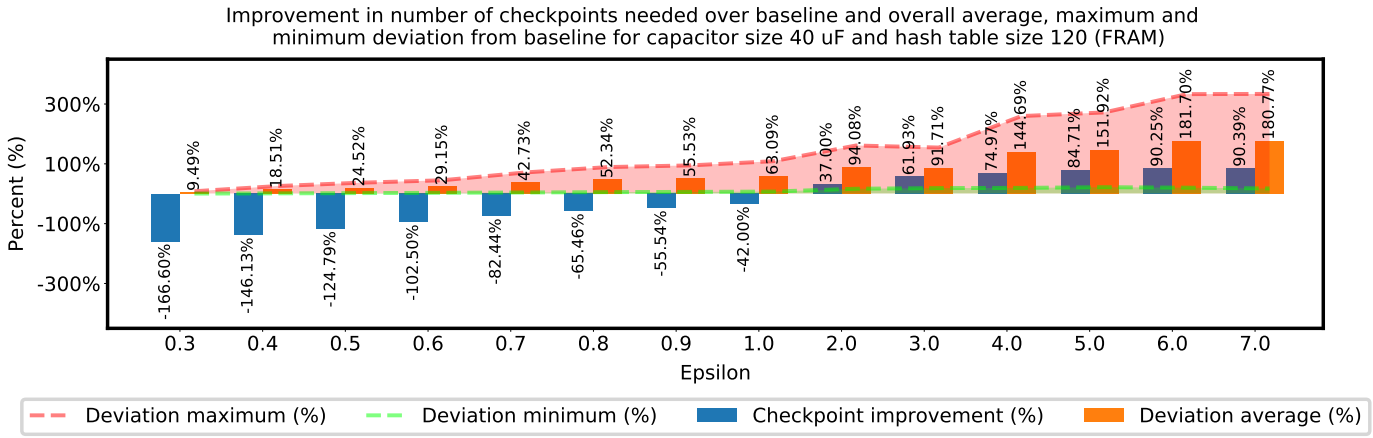

Figure D.1: (continued) Improvements in number of checkpoints over baseline, as well as overall average, maximum and minimum deviation from baseline, for capacitor size 40 µF and all hash table sizes, considering checkpointing to *FRAM* memory