POLITECNICO DI MILANO
Scuola di Ingegneria Industriale e dell'Informazione
Dipartimento di Elettronica, Informazione e Bioingegneria
Corso di Laurea Magistrale in
Computer Science and Engineering

A Case for Approximate Intermittent Computing

Supervisor: Prof. Luca Mottola

Student: Fulvio Andrea Bambusi

Abstract

The Internet of Things is nowadays part of our daily lives: embedded devices animate sensors, actuators and other tools that can collect or transmit information about physical objects. At the same time, powerful data analysis techniques are making it possible to make sense and control an amount of data never seen before. Despite the opportunities the IoT is creating, there are important criticalities that still hinder the diffusion of this technology.
An increasingly pressing issue is how to make the massive deployment of billions of devices sustainable: the vast majority of embedded devices is powered up by disposable batteries. Not only do batteries have an important environmental impact at the end of the lifecycle of the devices, but they are also hard and costly to replace when devices are deployed in remote or inaccessible areas.
These characteristics of batteries make it necessary to find alternative solutions to unlock a massive adoption of the IoT. As a consequence, a new paradigm is gaining momentum: we are talking about Transiently Powered Computing (TPC). The idea of TPC is to use energy harvesting techniques to extract from the environment small and unreliable energy streams, which can be used to power up embedded devices virtually independent from batteries.
However, applications deployed in TPC devices are subject to problems totally different from traditional computing. The sources that are used for energy harvesting produce a highly unpredictable power throughput, and therefore TPC devices can turn off unpredictably, losing their internal state and the partial results of the computations they were performing.
As a consequence, even small energy savings can make the difference from producing a result or nothing at all. The goal of this thesis is to address the domain of TPC using the techniques of Approximate Computing (AC). AC aims to alter the execution of arbitrary applications introducing some degree of error to produce a higher degree of saving. Approximate Computing Techniques (ACTs) make it possible to save computation time, memory and energy.
In this work, we will focus on the ACT of Iterative Refinement: it consists of starting from an initially rough solution, and continuously invest computational effort to improve it. By construction, Iterative Refinement (or Anytime) algorithms can always yield a feasible result: as a consequence, whenever a power failure occurs, it is possible to react by submitting the partial solution computed so far.
We have applied this ACT to create an Anytime version of Support Vector Machines (ASVM). We applied ASVM to a Human Activity Recognition (HAR) problem, and compared the performance of our method with a state-of-the-art TPC solution.
ASVM saved up to 41% of the energy cost of executions while paying an accuracy loss always below 20% also in the scenarios where we applied the most aggressive approximations. Moreover, the results produced by ASVM were more timely by up to 10s when the sampling rate is 50Hz.

Riassunto

L'Internet delle Cose è oggi parte della nostra vita quotidiana: i dispositivi embedded animano sensori, attuatori e altri strumenti in grado di raccogliere o trasmettere informazioni a proposito di oggetti fisici. Allo stesso tempo, potenti tecniche di analisi dei dati stanno permettendo di dare un senso e di controllare una quantità di dati mai vista prima. Nonostante le opportunità che l'internet degli oggetti sta creando, ci sono importanti criticità che ancora ostacolano la diffusione di questa tecnologia. Una questione sempre più pressante è come rendere sostenibile il massiccio dispiegamento di miliardidi dispositivi: la stragrande maggioranza dei dispositivi embedded è alimentata da batterie usa e getta. Non solo le batterie hanno un importante impatto ambientale alla fine del ciclo di vita dei dispositivi, ma sono anche difficili e costosi da sostituire quando i dispositivi sono installati in aree remote o inaccessibili. Queste caratteristiche delle batterie rendono necessario trovare soluzioni alternative per sbloccare l'adozione massiccia dell'internet delle cose. Di conseguenza, un il nuovo paradigma sta prendendo piede: il Transiently Powered Computing (TPC). L'idea del TPC è di utilizzare tecniche di energy harvesting per estrarre dall'ambiente piccoli e inaffidabili flussi di energia, che possono essere utilizzati per alimentare dispositivi embedded praticamente indipendenti dalle batterie. Tuttavia, le applicazioni implementate nei dispositivi TPC sono soggette a problemi totalmente diversi dall'informatica tradizionale. Le fonti utilizzabili tramite energy harvesting producono flussi inaffidabili e altamente imprevedibili, e quindi i dispositivi TPC possono spegnersi in modo imprevedibile, perdendo il loro stato interno e i risultati parziali dei calcoli che stavano eseguendo. Di conseguenza, anche piccoli risparmi di energia possono fare la differenza tra ottenere un risultato, o nulla in assoluto. L'obiettivo di questa tesi è di affrontare i problemi del TPC utilizzando le tecniche di Approximate Computing (AC). Lo scopo dell'AC è di modificare l'esecuzione di applicazioni arbitrarie introducendo un certo grado di errore per produrre un maggior grado di risparmio. Tecniche di calcolo approssimato (ACT) consentono di risparmiare tempo di calcolo, memoria ed energia. In questo lavoro ci concentreremo sull'Iterative Refinement: esso consiste nel partire da una soluzione inizialmente approssimata, e investire continuamente ulteriore sforzo computazionale per migliorarla. Per costruzione, gli algoritmi che fanno uso di iterative Refinement (o Anytime) possono in ogni istante restituire un risultato accettabile: di conseguenza, ogni volta che una il dispositivo embedded si spegne per via della mancanza di energia è possibile reagire presentando la soluzione parziale calcolata fino a quel momento. Abbiamo applicato questa ACT per creare una versione Anytime di Support Vector Machines (ASVM). Abbiamo applicato ASVM al riconoscimento delle attività umane (HAR), e confrontato le prestazioni del nostro metodo con una soluzione stato dell'arte basata sulla tecnica del checkpointing. ASVM ha risparmiato fino al 41% del costo energetico dell'esecuzione, perdendo al contempo sempre meno del 20% di accuratezza anche negli scenari in cui abbiamo applicato un'approssimazione più aggressiva. Inoltre, i risultati prodotti da ASVM hanno un anticipo fino a 10s quando la frequenza di campionamento è di 50Hz.

Ringraziamenti

Un enorme abbraccio alla mia famiglia, che mi ha supportato e aiutato lungo tutta la scrittura della tesi e il percorso universitario: senza di voi questo lavoro non esisterebbe.
Grazie al mio relatore per la scrupolosa attenzione che ha dedicato a questo lavoro. Grazie ai colleghi del NESLab che hanno dato un contributo enorme, senza cui il capitolo 6 sarebbe molto diverso, e peggiore.
Grazie agli ADA: la PCA sarebbe ben più misteriosa senza le nostre discussioni.
Grazie ai wisecolleghi per il sostegno tecnologico (e non). Grazie a tutt* quell* che hanno percorso un pezzo di questa bellissima strada al mio fianco.
Ad maiora!

Chapter 1

Introduction

The Internet of Things is becoming increasingly popular as embedded devices become cheaper and more powerful, and data analysis techniques make it possible to process the ever-increasing flow of data produced by sensor networks.
While the Internet of Things is revolutionizing our lives, its dark sides and disadvantages are starting to show up. One of the most prominent technical, social and economic problems that hinder the diffusion of the IoT is that the vast majority of embedded devices is powered up by disposable batteries. Batteries have an important environmental impact at the end of the lifecycle of the devices, they are hard and costly to replace when devices are deployed in remote or inaccessible areas.
The characteristics of batteries make it unsustainable to continue to use them when the IoT will be an even more pervasive aspect of our lives, so new alternatives are being explored.
Energy Harvesting (EH) is novel paradigm to solve the problem of powering up tiny devices without major environmental or logistic consequences: while battery-powered embedded devices are given a certain amount of energy at the beginning of their lifecycle, and they use it until they permanently run out of power, the idea of EH is to constantly replenish the energy storage of devices by extracting energy from the environment.
As a matter of fact, there are plenty of such spare power sources whose throughput is normally wasted: vibrations produced by vehicles, small temperature gradients and oscillations caused by heavy industrial machinery, currents and breezes or chemical gradients are only some examples [Bha+16].
EH has a positive impact on the sustainability and the cost of embedded devices. Thanks also to the reduced maintenance effort, EH can be seen as the enabler to deploy IoT devices on a larger scale than ever [Tal+15].
The design of EH-powered IoT devices is driven by the same goal: to foster the massive and pervasive deployment of the devices. Such devices are very lean in terms of memory, computational power and software. Typical microcontrollers (MCUs) have a volatile memory (typically SRAM), and a nonvolatile portion of the memory (NVM), and one or more processing units: however, they typically

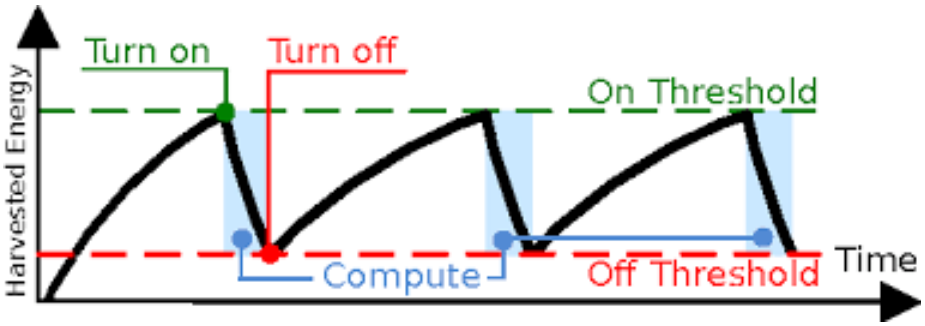


Figure 1.1: Intermittent execution of a program [Luc+17].

lack any kind of Operating System (OS) and have low performance compared even to low-end laptops([Luc+17] [19o]).
EH has by definition one main problem: power streams gathered by harvesters are unpredictable, unstable and unreliable, so it is normal for power failures to occur during the normal execution of EH-powered devices ([Luc+17] [HSS14]). Figure 1.1 shows the workflow of EH-powered devices: the harvester accumulates energy up to a given threshold. When the threshold is met, the device turns on and quickly consumes the accumulated energy.
This way of functioning gives name to the discipline of Transiently Powered Computing (TPC), the branch of computer science that studies the peculiarities of intermittent executions.
The main aspect of TPC is that power failures happen frequently and that they have more serious consequences than in the context of continuous executions. As we mentioned before, IoT devices are extremely simple to be cheap, unobtrusive and massively deployable: however, their simplicity has one major drawback, that is to say, it makes devices vulnerable to the consequences of power failures. Due to the absence of the OS, when the power goes off the state of the volatile part of the memory is immediately totally erased. So, if partial results have not been committed to the nonvolatile part of the memory, there is no forward progress in the computation.
This problem is summarized this way: the impossibility to obtain continuous execution causes power failures, that hinder or block the progress of applications, that cannot advance unless they save partial results. In the literature, it is possible to find several solutions to mitigate the effects of power failures. In particular, two main paradigms encompass the majority of the existing solutions.
The first paradigm consists in storing the state of volatile memory according to a given trigger, such as low battery level or function calls/loop iterations ([Bal+15], [BM17] [VBM17]). These solutions are called checkpoint-based. The state of volatile memory can be restored at the end of power shortages, whose effects are made invisible to the application.
The second approach is to decompose programs in small tasks that are executed

atomically and whose result is committed to nonvolatile memory when they terminate ([LR15],[HSS17],[MCL17]). Tasks can communicate by exchanging their outputs, to create arbitrarily complex applications.
Both approaches have some limitation: writing to nonvolatile memory is an energy-intensive task, so checkpoint-based solutions sacrifice a fraction of the available energy to afford the overhead of storing the state. As for tasks, one of the main difficulties of using tasks is how to split programs into tasks of the correct size.
The problem of selecting the best boundaries to enclose tasks has been addressed, among others, by Colin et al. [CL18] and Ahmed et al. [Ahm+19]. This problem is not trivial, because both short tasks and long tasks have drawbacks. Short tasks guarantees forward progress, because they require a small amount of energy to execute, so it is enough that the harvester produces some throughput for a short time to achieve a small step forward. Unfortunately, when tasks terminate they spend energy to store their results to NVM, and this overhead is more costly for programs composed by a multitude of short tasks.
On the other hand, long-lasting tasks have lower overhead but they need more energy to execute: as a consequence, it is more likely for a long task to fail wasting all the energy invested in the computation.

In this thesis, we propose a different approach to mitigate the consequences of intermittent executions: we propose to adapt the energy cost of TPC executions to the actual availability of energy to guarantee at the same time forward progress and low overhead.
We propose to use Approximate Computing Techniques (ACTs): ACTs are software or hardware mechanisms that reduce the quality of the output of computations and lower their cost. The advantages of our solution can be summarized this way: adapting the behaviour of a program and hence its cost to the power that is currently available in the environment allows us to optimize the utilization of resources, and to obtain timely results.
In the best possible case, it can be the case that no access to NVM is necessary because computations produce results before power failures occur. To show the feasibility and the usefulness of this approach we have developed a proof of concept application: our goal is to demonstrate our claim that ACTs have a positive impact in the domain of TPC.
Our proof of concept application makes use of Iterative Refinement: the idea of this technique is to refine initially rough solutions investing more and more computational effort.
This approach has one immediate advantage: the computation can be interrupted in every moment, and a partial solution is by construction available. The effect of Iterative Refinement is shown in Figure 1.2. There are two inputs, labelled as Sample 1 and Sample 2, that become available in two different time intervals, and each one triggers the same computation. The computation triggered by Sample 1 can be fully executed, and so it returns a result with full accuracy. The computation triggered by Sample 2 is interrupted by a power failure, so it returns a result before it is fully refined.
Also in the situation a power failure occurs, the result was returned in a very

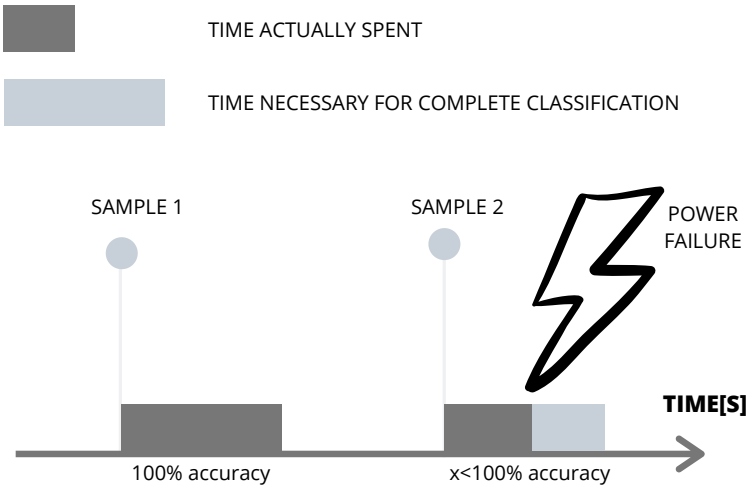


Figure 1.2: The effect of power failures on our Iterative-Refinement-based application.

timely way, immediately before the power failure itself. In this situation, all the available energy has been invested in the computation, which means that the partial result that has been produced is the best result that could be achieved with the available resources.
We applied an Iterative Refinement algorithm to solve a Human Activity Recognition (HAR) problem: HAR consists of classifying the activity that is being performed by a human being using accelerometer data [YWC08].
In particular, we designed an Anytime version of Support Vector Machines (ASVM). The idea of ASVM is to use one feature at a time to continuously refine initially inaccurate classifications, that become more and more precise as more features are used. We measured the performance of our application and compared it with a checkpoint-based solution: our solution is able to cut the energy cost of classification up to 41% and is systematically more timely (up to 400 sampling periods), while the reduction in the accuracy of classifications remains below 20%.

1.1 Problem

TPC consists in using highly unreliable energy harvesters to sustain the needs of embedded devices without making use of batteries. However, energy harvesters do not produce a stable, predictable or reliable throughput in terms of energy, so the available energy could abruptly become insufficient to power up the device.
Therefore frequent power failures take place, and their negative effects are in-

creased by the essentiality of TPC devices. When power failures happen, the state of volatile memory is immediately lost: as explained earlier, by default TPC devices do not have any component that protects them from failures, for example by saving the volatile state. First of all, any computational progress done but not committed to nonvolatile memory is immediately lost.
Moreover, the duration of a power failure cannot be predicted: as a consequence it may happen that devices resume old computations triggered by old inputs. However, the validity of data is limited in time: in practice, data are useful for a given period that depends on their source and on the target application, after which they become useless ([HSS17],[Sch+08]).
These fundamental issues have three important consequences:

1. Power failures can hinder or totally block the forward progress of applications. If no mechanism is adopted to store the partial state of the computation or to commit partial results, every time an application is restarted after a power failure it will redo from scratch computations that have already been carried out in the past.

2. At the same time, introducing mechanisms to cope with power failures may introduce important overhead: writing variables to nonvolatile memory is very costly in terms of energy consumption, and so saving too many variables or too often would constitute a huge waste of resources.

3. Another problem is that it is basically impossible to forecast the duration of a power failure: this means that it could take an arbitrary amount of time to process an input. This could be particularly problematic because some inputs are intrinsically short-lived, as they soon become worthless [HSS17].

1.2 Solution

We can address the issues of TPC by investigating what is the best way to allocate the scarce resource of energy on the computations to perform. In other words, the problems illustrated earlier can be interpreted as resource planning and optimization problems.
Under this hypothesis, the problem of TPC can be immediately translated into the problem of understanding what are the operations that should be privileged and how execution should be organized.
There are three main aspects of this planning problem that should be taken into account: ideally, one would maximize the quality of the obtained result and its timeliness while keeping its cost low. This work is based on the observation that obtaining a timely result is key for an effective TPC application.
This observation is widely accepted in the literature. For example, Hester highlighted in [HSS17] that the input of TPC applications has a limited validity: for example, old sensor data become useless with the characteristic changing time of the associated phenomenon. If a sample is not used quickly, the produced

output is totally useless. InK [Yun+18] is a TPC kernel based on the same principle, that some data have to be handled either urgently or never. Maeng et al. [ML19] evaluated their checkpoint-based solution Samoyed using the execution time as metric.
At the same time, another equally vital aspect of TPC is the energetic cost of executions: as we have highlighted earlier, this is the first-order problem that creates also the difficulty of producing timely outputs. As a consequence, the primary focus of our solution will be the optimization in the usage of energy. In particular, we will leverage on the balance between quality and cost to tune the behaviour of TPC applications.
We will use Approximate Computing, a discipline based on the idea that small performance degradation can bring huge savings in terms of cost.
Our proposal is the following: the power consumption of TPC applications should be adapted to the amount of resources that are available when the execution is taking place. While Approximate Computing Techniques (ACTs) in the state of the art consider the quality of the result as a constraint and the maximization of the savings as a goal ([Yeh+07], [Sid+11]), we propose to embrace the dual paradigm where the available resources are a constraint and the goal is to maximize the quality of the output.
This approach guarantees two main benefits:

1. Resources are consumed optimally: the cost of computations is tuned to consume all and only the available resources. Moreover, in the best case it is possible to obtain a have a very low overhead: the only information that has to be committed to NVM is the final result of a computation.

2. Computations are guaranteed to terminate quickly because their cost can be tuned in such a way they produce results before the following power failure.

We applied the ACT of Iterative Refinement, that consist in creating an initially rough solution and step by step improve it by investing more computational effort. The advantage of this technique is that a feasible solution can be yielded in every moment. This ACT is extremely apt for TPC contexts, because it can be used to return results immediately before power failures thus consuming all and only the available resources.
We applied this technique to create Anytime Support Vector Machines (ASVM), and we used this ASVM to carry out a Human Activity Recognition (HAR) task. HAR is the problem of understanding the activity that a human being is carrying out by processing heterogeneous data. We measured the performance of ASVM in terms of timeliness, accuracy and energy consumption and we compared these metrics with the ones scored by a checkpoint-based state-of-the-art solution.
ASVM outperformed the checkpoint-based solution in terms of timeliness, anticipating it up to 20s, while the sampling rate of the samples is 50Hz.
Our application saved up to 41% of the energy cost of the execution, while the accuracy loss was always below 20% in the scenarios where we applied the most

aggressive approximations.

1.3 Roadmap

The thesis is structured as follows: first of all, we will give a picture of the state of the art then we will illustrate our contribution and finally we will present the evaluation of our work. Here follows a summary of the chapters that compose this work.

1.3.1 Transiently Powered Computing

In Chapter 2 we will present the main characteristic of the discipline of TPC. The first part of the chapter is dedicated to the main problems that arise when applications are executed in a TPC context, and the opportunities offered by this novel discipline. Then, we will illustrate the most important techniques that can be used to harvest energy from the environment. Finally, we will present the solutions that can be found in the literature, by making use of a new taxonomy to classify them.

1.3.2 Approximate Computing

Chapter 3 is focused on the field of Approximate Computing. The beginning of the chapter will be dedicated to the advantages, the risks and the constraints to trade performance and cost. Then, we will present the techniques that can be found in the state of the art. This presentation is divided into two parts: then we will describe ACTs as black boxes, classifying them according to the effect they have on applications they are used on. Finally, we will open the box, and we will show the different mechanisms that can be practically used to tune the cost-performance tradeoff.

1.3.3 Approximate Intermittent Computing

In Chapter 4, we will discuss why the idea of using AC in the domain of TPC is particularly promising.
The main benefit of AC is to reduce the cost of applications, and this can make the difference in TPC. On the other hand, its main disadvantage is to reduce the performance of applications, and we will argue that in TPC this is not a particular problem.
Then, we will explain the principles that drive the application of Approximate Computing in traditional, continuously powered scenarios: namely, AC is used to lower the cost of executions while maintaining a decent quality of the output. We will support this claim with several examples from the literature, and we will propose a dual approach to maximize the efficacy of AC in TPC contexts: in particular, the goal should be to maximize the quality of the result wile upholding with a resource budget.

1.3.4 Making Things Happen

In Chapter 5 we will describe the functionalities of the concrete application that we developed to demonstrate the feasibility and the usefulness of the principles illustrated beforehand. We will discuss the precise ACT we want to use in the application, that is to say, Iterative Refinement. This technique offers several advantages, one of the most important being its low runtime overhead. Afterwards, we will present the precise algorithm that we want to approximate using Iterative Refinement, Support Vector Machines: after a brief theoretical overview, we will study the properties of the Anytime version of SVM we have developed in terms of how this application handles the cost-performance trade-off. Finally, we will present the use case that we will analyze, namely Human Activity Recognition.

1.3.5 Prototyping

In Chapter 6 we will describe the implementation details of the application we developed to measure on the field the performances of ASVM. The implementation has to uphold with strict constraints in terms of memory and computational power, so we tested its correctness by comparing its results with the output of a Python implementation. This comparison allows us to demonstrate that our implementation is correct, and so its performance can be scientifically measured.

1.3.6 Evaluation

The goal of Chapter 7 is to measure the performance on the field of our application, both in absolute terms and compared to a state-of-the-art solution.
First of all, we will measure the accuracy of the probabilistic description of ASVM we provided in Chapter 5. After having measured that it is quite accurate (the error is on average below 5%), we will describe the experimental setup that we have used to measure the performance of our application. Finally, we will present the metrics of our application, in terms of timeliness, accuracy and energy saved with respect to a state-of-the-art solution.

1.3.7 Conclusion and Future Works

Chapter 8 is used to summarize the results obtained in our work: we have demonstrated that it is feasible to apply ACT to improve the effectiveness of TPC applications. We will also discuss the generality of our Proof of Concept since the choices that we necessarily made to implement an application do not narrow the scope of the work.
In the rest of the chapter, we will illustrate several research lines that opened up during the work, and that we will further investigate in the future.

Chapter 2

Transiently Powered Computing

2.1 Abstract

The goal of transiently powered computing is to overcome the usage of batteries to power up embedded devices. The basic idea behind this paradigm is to harvest energy from various sources naturally present in the environment, to accumulate the energy in small storage devices and to spend it when needed.
Although the nominal power output of the sources can be very high, like in the case of solar energy, the low efficiency of harvester makes only a low amount of energy available for the devices to use [Bha+16]. Energy harvesting can be carried out by exploiting a plethora of sources, with different characteristic times and power intensity.
All these sources, however, do produce an energy throughput far lower than the consumption of an even small device, which is therefore, victim of unpredictable power failures. Those failures make execution of programs in transiently powered contexts radically different from traditional scenarios. In particular, their main consequence is that devices are unpredictably turned off without notice multiple times per second.
Therefore Transiently Powered Computing (TPC) is characterized by classes of errors and inconsistencies that are totally unobserved in traditional scenarios: for example, sudden shutdowns may hinder the termination or even the forward progress of a program, as it is continuously restarted and all its progress is lost. The main characteristics and consequences of these inconsistencies will be better explained in Section 2.3.5.
This chapter is meant to be a gentle introduction to the domain of TPC. The exposition will be a journey from the high-level, qualitative concepts behind this domain to the concrete implementations of solutions developed for this domain. The chapter is structured as follows: first of all, a section is dedicated to the description of the opportunities and the challenges of this field of computer science.

Then, the goal of the second section is to display the types of scenarios where it is concretely possible to apply energy harvesting techniques. It delves deep into the topic of energy harvesting, describing the power sources from which it is technologically feasible to extract power. The third section contains an overview of the actual techniques that can be found in the literature to address the challenges of Transiently Powered Computing, listed in Section 2.3. The commonalities shared by different solutions are used to frame them in a simple taxonomy. After the presentations of the state of the art, a brief conclusion is used to highlight the main characteristics of existing works in a comparative fashion.

2.2 Opportunities and Challenges of Transiently Powered Computing

The adoption of transiently powered computing is fostered by some of its intrinsic features and by the inadequacy shown by traditional, battery powered devices in specific application domains. Batteries do have a limited energy content, which sets a hard limit to the lifespan of any connected device. Such predetermined duration can bear heavy consequences in situations when the maintenance or the replacement of devices is either impractical or costly.
Moreover, disposing exhausted batteries has a huge impact both in economic and environmental terms. Despite the disadvantages shown by traditional computers, transiently powered computation opens up a series of technical, design and operational challenges.
First of all, the unreliability of the energy sources used in the domain of energy harvesting makes it impossible to guarantee hard boundaries on the reliability or availability of transiently powered devices.
Moreover, sudden shutdowns can break the execution flow of normal programs, as shown in Figure 1.1. This kind of behaviour is caused by the fact that volatile memory is unable to maintain its state unless it is powered up. Therefore, whenever a program is restarted after a shutdown, all the progress which has not been stored to persistent memory is immediately lost, and, as a consequence, the state of nonvolatile memory may become incoherent with volatile memory.

2.3 Challenges

Several works in the literature explicitly list the most urgent open questions in the domain of Transiently Powered Computing, while others do implicitly stress specific problems by directly proposing related solutions. Below follows an inventory of the problems that are most commonly met in the state of the art. There are two broad categories of problems. The problems in the first class are the technical difficulties specific to the very domain of Transiently Powered

Computing. The second class of problems contains the difficulties physiologically related to the fact that TPC is a very young domain, and therefore it still lacks standards and canons. The first four points of the list are taken from a position paper carried out by Hester and Sorber [HS17].

2.3.1 Constrained Resources

To work with the low amount of energy that is available in the environment, the devices used in the domain of TPC have to consume a very low amount of power [19a].
This implies that the chips built for this domain must be highly efficient [19l]. Sometimes this is not enough: complex sensing applications, which are becoming increasingly popular [19c], are computationally intensive.
To give an idea of how hard it is to develop an application on a Transiently Powered Computer, several techniques require dedicated hardware acceleration to work flawlessly. For example, machine learning applications, which are being used in the domain of edge computing [19j].
MSP430, a microcontroller commonly used in industrial and academic applications [Ahm+19], has less than 66kB of RAM, and at most 25 MHz clock frequency [19o].
To give an idea of how low these parameters are, STM32H7, a high-end microcontroller by ST Microelectronics, boasts 1MB RAM and up to 480MHz clock frequency [19g].

2.3.2 Uncertain Environment

There are many possible sources where energy can be harvested. This variety adds a level of complexity to the deployment of transiently powered applications: the fact every source has his characteristic profile creates a range of possible behaviours of programs. As a consequence, developers have to take precautions against all the possible execution paths, either manually or with the help of automatic tools.
Another layer of complexity is added by the fact that even the harvesters which make use of the same type of energy source are optimized for specific working conditions: for example, each mechanical harvester is maximally efficient when the vibrations that it exploits have a specific characteristic frequency. When mechanical harvesters are used for vibrations at different frequencies, their efficiency is greatly reduced Madankan et al. [MKS14].
The same holds for other types of harvesters: to summarize, the amount and the shape in time of the energy that is made available by the environment have a major impact on the efficiency the energy itself can be harvested and used by computing devices.
Mechanical stress received from the environment can make energy harvesters, especially the ones which make use of kinetic energy, shorter-lived than batteries [19f].

Ahmed et al. [Ahm+19] showed that the energetic cost of computations is highly dependent on the voltage of the power supply of a device, and that this fact can be used to enhance the quality of execution of transiently powered applications. Gomez et al. [Gom+17] stressed the variability of the characteristics of embedded hardware, both in terms of supply voltage and power requirements. Therefore, it is hard to design a solution whose performance generalizes well on arbitrary systems.
Ahmed et al. [Ahm+19] investigated the relationship which binds the supply voltage of a device with its efficiency, discovering that each voltage level has a corresponding optimal operating frequency.
Compile-time settings have already been investigated with profit in the domain of traditional computing [XMM03b], but such techniques are still unexplored in TPC, whereas adaptation to the context in a transiently powered setting could have a major impact, as suggested by Maeng and Lucia [ML18].

2.3.3 Timely Execution

Transiently powered applications cannot offer any guarantee about the time when a given task will be executed. Hester et al. [HSS17] suggested that it is the case that some computations become irrelevant after some time, and therefore mechanisms must be designed to detect and address the problem of time. Gawali and Deshmukh [GD19] remarked the fact that sensing devices are one of the main application domain of energy harvesting, and that sensor data are can be the first step of a pipeline which brings to real-time, critical decisions.
Therefore, accessing and using old data can have catastrophic consequences. The problem of timeliness is one of the hardest challenges in this domain, so it will be further treated in Section 2.6.

2.3.4 Coordination and Sharing

Contrary to traditional networks, faults are the normality and not the exception for TPC devices. The usual protocols must be adapted or dropped completely to face this issue. Additionally, the transmission of data with traditional protocols [19al] is an extremely power-consuming activity, therefore it must be handled with extreme care. New protocols, such as LoRa and NBIoT, are increasingly being adopted [19c] to reduce the pain of transmission.
However, these new technologies typically make use of constantly-powered gateways [19e], which simply shifts the problem of energy harvesting to a higher level.
Many traditional protocols involve the election of a leading node, which is often selected according to specific builtin characteristics such as its computational power or serial number Al Nahas et al. [ADL17].
However, the type and amount of energy that is available for each node to use has a major impact on the operational characteristics and effectiveness of the node itself. As a consequence, a node may be perfectly fit to lead a network in a continuously powered context, but may be deployed in a position where it is

unable to properly work due to the absence of free energy.
So, it is necessary to take into account the types and power intensities of energy sources of the environment where a system will be deployed to effectively design and operate the transmission of information.

2.3.5 State Inconsistencies

Lucia and Ransford [LR15] showed that TPC executions can produce results that are incoherent with a traditional sequential execution. They did also produce two different models to describe and treat formally intermittent execution. Nominally, they propose to consider TPC as an instance of concurrent execution, or to analyze it by altering the control flow graph to add all the possible execution paths following a power failure. In another paper, Ransford and Lucia [RL14] highlighted the danger of using at the same time volatile and nonvolatile memory for a TPC program: while volatile memory is erased at every shutdown, nonvolatile memory survives. Therefore, in two subsequent executions, the same line of code can meet two different program states.
For example, Figure 2.1 shows a situation when a power failure makes an execution follow a path which is different from the one of a continuously powered execution. A function is called, and so its return address is pushed on the stack. Then, it is partially executed, modifying nonvolatile memory, but the state of volatile memory is not saved by the runtime environment. When a shutdown happens, the stack contains the return address of the caller but the memory has already been modified by the callee.

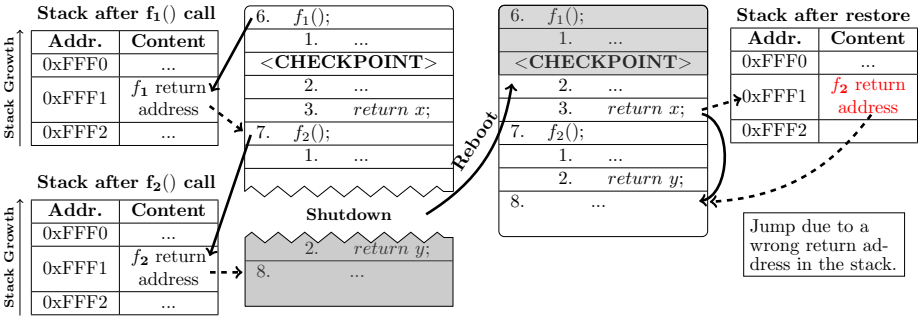


Figure 2.1: Consequences of a power failure

Maioli et al. [Mai+19] did further investigate the nature of these inconsistencies, classifying them in five different families and inventing an algorithm to analyze them.
Maeng and Lucia [ML19] explored the consequences that checkpoints can have on peripherals. In particular, they showed that it is possible to observe several erroneous behaviours when a peripheral is activated by an embedded device which later on fails due to lack of energy. For example, the peripheral may wait indefinitely or produce an incorrect output. This issue is particularly sensitive

because the vast majority of embedded devices incorporates some kind of sensor or actuator [19m].

2.3.6 Security

In his PhD thesis, Ransford [19af] discussed the topics of security and storage on transiently powered devices.
TPC devices do depend on external entities for working. This characteristic can be problematic in case the transiently powered device has to communicate with the source, as it will talk to anyone who gives it energy. Wouters [19p] discussed the problems that are brought about by some of the most commonly used technologies in the domain of Transiently Powered Computing, that is to say, checkpoints (see Section 2.6.1). Specifically, checkpoints consist in storing the volatile state of an embedded device into its persistent memory just before a power failure, but there are several other techniques which are based on committing values to nonvolatile memory (see Section 2.6). In the same work, Wouters discussed the opportunity to encrypt the state when saving it, to prevent theft and tampering.
Krishnan et al. [Kri+19] explored the possibility to use power interruptions as an attack vector, and designed a protection mechanism to shield Transiently Powered Computers against this menace.

2.3.7 Storage

Gomez et al. [Gom+17] showed that it is necessary to use persistent memory to log the data recorded by sensors, as power failures make RAM unsuitable and transmission is too expensive. Moreover, nonvolatile memory is widely used to persist state across power failure. However, nonvolatile memory is two to three orders of magnitude more expensive to write than volatile memory [Par+11] [19ah]. Even ignoring the logical problems which derive from blind usage of persistent memory (see Section 2.3.5), its huge energy cost makes it not feasible to rely systematically on it. Verykios et al. [VBM17] highlighted the limits of using state retention techniques which are agnostic concerning the underlying hardware and opened the research proposal to adopt both software and hardware techniques to use persistent storage optimally.
As we said before, traditional volatile memory is not particularly fit for TPC, as it is completely erased by power failures. Therefore, several scholars have worked to design alternative technologies which can retain information even in case of a sudden shutdown. For example, NonVolatile RAM NVRAM offers performances similar to traditional RAM but it does not need to be powered up to keep information [Sch+15].
NVRAM has been used by Aouda et al. [FMS14] to make transiently powered executions less costly than a checkpoint-based tool that makes use of traditional memory architecture.

2.3.8 Low Technology Readyness Level

Technology Readiness Level is a scale to measure the maturity of a technology, from basic research to complete market penetration [19q].
This scale ranges from 1, which means "Basic principles observed and reported", to 9, meaning "Actual system "flight-proven" through successful mission operations".
Groen states that the TRL of Energy Harvesting is 7 [19n], hence that it is not completely mature yet.
An important consequence of the immaturity of TPC is the lack of standardized testbeds and benchmark suites to evaluate and compare different applications developed in the domain of TPC [AMP16][HS17].
Balsamo et al. [Bal+19] suggested that Transiently Powered Computing-specific solutions should be integrated within existing operating systems for the IoT in order to make them more accessible to academia and industry.
The lack of standardization and large-scale adoption is a serious menace to the broad-scale adoption of Transiently Powered Computing: Jackson et al. [JAD18a] forecast a scenario where the pain of embracing this new paradigm becomes too difficult for developers to use.

2.4 Opportunities

From a high-level point of view, TPC unlocks the large-scale adoption of a variety of application, especially in the domain of sensing [HS17].
Sagentia [19d] suggest that energy harvesting should be conveniently applied in scenarios ranging from industry to automotive, or environmental monitoring.
Moreover, energy harvesting solutions fit well wearable devices, which will be a major driver in the market of IoT according to McKinsey [19k].
There are several characteristics of intermittent computing which can unlock these benefits. Below follows a list of the main ones.

2.4.1 Cheapness of Power Source

Jackson et al. [JAD18b] highlighted the fact that batteries are not the dominant cost in IoT devices. However, they show that batteries have a price which is slightly higher with respect to harvesters, so the cost of the energy management part of circuits can be halved by adopting the batteryless paradigm. An additional saving can be obtained thanks to a specific class of energy harvesters, that is to say antennas. Talla et al. [Tal+15] demonstrated that it is possible to use the same antenna to transmit information over WiFi and to harvest energy from radio waves. Therefore, if a device has to transmit using WiFi it can be adapted to use the same hardware components to gather extra energy from the environment.

2.4.2 Form Factor Reduction

When a computer is powered up by a battery, its lifespan is proportional to the energy content of the battery, which is proportional to the size of the battery itself. Energy harvesters decouple the duration of a device from its size. Moreover, as explained in Section 2.4.1, it is possible to use the same antenna to harvest energy and to transmit data. When this approach is adopted, the power source occupies no extra space at all, allowing further miniaturization of TPC devices. The weight of the battery in smartphones is more than 80% of the total weight [19ai]. In the context of IoT, even small-size batteries weight around 1/3 of the node [19aj] [19ak].

2.4.3 Sustainability

Hao et al. [Hao+17] estimated in 0.109 kg CO $_2$-eq/Wh the GHG (GreenHouse Gas equivalent emission) of lithium-ion batteries during their life. On the other hand, Gerbinet et al. [GBL14] measured that the equivalent emissions of photovoltaic panels are around 0.0039 kg CO $_2$-eq/Wh. This is an example of how energy harvesters incorporate less carbon than batteries.
Benecke et al. [Ben+12] compared the environmental impact of energy harvesters with a conventional LiPo battery, and found out that 5 to 8 years of operations in normal conditions are enough for harvesters to outperform batteries.

2.4.4 Low Maintenance

Batteries cause constant operative costs, as they have to be periodically replaced. Apart from its financial cost, maintenance may be problematic in case devices have been deployed in remote and hardly accessible areas, or in the case of medical devices [Chou et al. 2010]. Enocean [19ag] estimated that the labor cost for replacing batteries amounts to 10% of the total cost of ownership of an IoT node. Quite plainly, this cost would be totally saved by the adoption of batteryless solutions.

2.5 The Problem of Energy in TPC

Although the importance of energy is paramount in computation in general, in the domain of TPC this aspect is even more vital.
While in traditional scenarios energy is a matter of efficiency, the efficacy of transiently powered devices depends on the amount of energy they have at their disposal. Due to this tight bond, an overview of TPC would not be complete without talking about the actual energy harvesting techniques. This section owes much to the work of Bhatti et al. [Bha+16].

2.5.1 Kinetic Energy

Energy can be extracted from movement using various technologies. The energy throughput does greatly vary according to the type of source which is used, from uWs to tens of mWs. The characteristic time and the reliability of harvesters which make use of this kind of energy also depend on the source.
The spectrum of possible scenarios ranges from situations where energy is provided in a continuous manner (such as vibrations from machinery) to extremely intermittent and unpredictable sources (such as human movement). Sazonev et al. [Saz+09] used this kind of technology to monitor the structural health of bridges, extracting energy from the vibrations induced by vehicles passing by.

2.5.2 Radiant Sources

Electromagnetic radiation from different bands of the spectrum can be converted to usable electric power. Both the energy density and the technology used to scavenge power depend on the wavelength of the radiation, ranging from the mWs of visible light to tens of nW for radio frequencies.
These factors do also affect the characteristics of the extracted energy profile in terms of time. For example, solar energy is not available at night, while RFID tags are only powered when they are put close to a reader. One of the most widely used radiant sources is the sun, whose energy can be harvested through photovoltaic panels [18]. The utilization of solar energy to power up embedded devices has been pioneered by Gutierrez et al. [Gut+14], who built a system to autonomously irrigate fields.

2.5.3 Thermal Sources

Thermal harvester use the difference between the temperature of two surfaces to generate an electric potential. This kind of technology can produce up to hundreds of uW. As far as reliability is concerned, it is one of the most dependable ones, as there exist various heat sources which are almost continuously working. For example, this kind of harvesting has been successfully applied to recover waste heat from data centers or warm water pipes. Rizzon et al. [Riz+13] used the heat dissipated by microprocessors to power up a Wireless Sensor Network for environmental monitoring.

2.5.4 Bio-Chemical Sources

This last class of energy sources uses the fact that several metabolic processes naturally emit electrons. These processes can be exploited by feeding living microorganisms or by reproducing them in controlled conditions. In any case, harvesters using this principle are incredibly reliable, as they just need some kind of chemical substrate to work properly. Pietrelli et al. [Pie+14] exploited this energy source to power up a Wireless Sensor Network for precision agriculture.

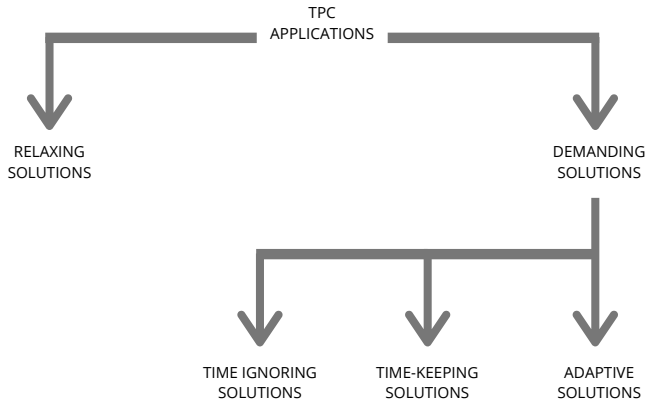


Figure 2.2: Taxonomy of System Solutions

2.6 Existing System Solutions

Some of the problems listed in Section2.3 have already been addressed in the literature. The solutions are presented according to the following taxonomy, sketched in Figure 2.2: first of all, we make a broad distinction between relaxing and demanding solutions. The former class represents all those techniques and applications which aim at making the intermittence of computation transparent to the developer, who can develop code as if he/she were in a traditional scenario. The latter does instead contain all the solutions which consider embedding mechanisms to cope with the effects of power failures and minimize their consequences. Programmers must comprehend and appropriately incorporate such mechanisms in their code.
Demanding solutions are further split according to how they address the problem of time. Islam et al. [ILN19] highlighted that one of the main consequences of the frequent power failures it is possible to observe in the domain of TPC is the inability of devices to reliably measure the passing of time. They classified state-of-the-art solutions based on how this problem is addressed, distinguishing time-keeping solutions from adaptive solutions. Time-keeping solutions succeed in measuring time by leveraging various physical properties of the board, while adaptive solutions change their behaviour according to the profile and amount of available energy.
A further dimension to frame a system solution is whether it guarantees or not semantic equivalence in the execution of a program executed in a transiently powered context to its traditional execution in a continuously powered environment.

2.6.1 Relaxing Solutions

Taking the point of view of the developer, the class of relaxing solutions comprises the techniques which want to make the code of an application in a TPC scenario as similar as possible to the one that would take place if the device was continuously powered.
In other words, these solutions do not contain any component which allows programmers to modify the program's behaviour according to possible failures taking place.
Relaxing solutions typically consist of an additional layer which wraps the program and protects it against the effects of power failures, making it possible for programmers to use transiently powered devices without any kind of intervention.

An example: Checkpoint-based Solutions

One way to address the problem of intermittence bugs is to store the state of the computation when certain conditions are met. When the device turns on after a power failure, the first operation it performs is to restore the last saved state.
Possible conditions to trigger the storage of the state are function calls, entrance to loops, timers or low energy level. Some of the main properties a system of checkpoints should guarantee are a low amount of overhead and the certainty not to get stuck between two checkpoints.

2.6.2 Demanding Solutions

Demanding solutions are characterized by the fact they require programmers to adopt some kind of new paradigm to design their code.
Demanding solutions are typically frameworks which offer ways to describe both computation and failure. Programmers must use them to describe both the code that must be executed and its interaction with power failures.
In other words, executions adapt their behaviour to failures and modify their flow according to guidelines that are suggested by the programmer.
The goal is to make failure causes no damage, and to consider it a somehow physiological, acceptable event.
Several solutions in the literature address the problem of measuring time. This aspect is particularly important in the domain of TPC because one of the most popular applications of embedded devices are the deployments of sensor networks. By their own nature data come with an expiration date, which makes their value decrease less punctual they are. Islam et al. [ILN19] considered the way time is treated as the fundamental factor to classify TPC solutions. They distinguished time-keeping solutions from solutions which modify their behaviour according to the context (from now on, adaptive solutions). For the sake of completeness, it is also necessary to consider also those systems that do not take into account time but are anyway successful in providing some kind of

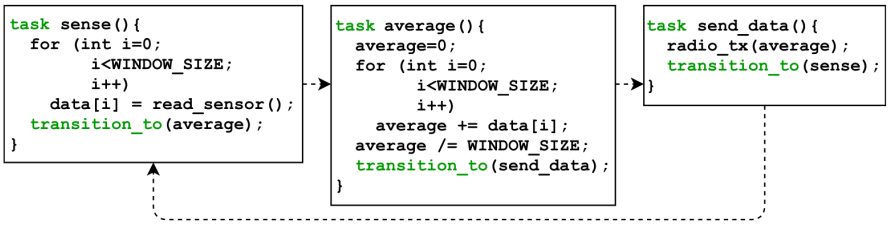


Figure 2.3: An application written in Alpaca

consistency guarantee.
The way time is treated is meaningful only when speaking about demanding solutions: relaxing solutions do not have any way to expose any aspect of power failures, and so they do not expose any special mechanism for the user to inspect the duration of shutdown periods, which are among the main consequence of power failures.

Time-ignoring Solutions

Some works [RSF11], [Bal+15] do not take into account the problem of time, but are designed in a way to minimize the impact of failures. Task-based solutions, such as Alpaca [MCL17] are probably the most popular examples of demanding solution. In task-based systems tasks are executed atomically, i.e. either completely or not at all. The consequence of this policy is that whenever the device experiences a power failure, it dies after the effects of a precise number of tasks have been applied. In other words, the board will always, by definition, be observed in a state defined as meaningful by the programmer.

An example: Alpaca

Alpaca is a programming model developed by Maeng et al. [MCL17], which consist in splitting programs into tasks, which are committed atomically at the end of their execution. This way, programmers are given the guarantee that the execution will go on and, moreover, that the device will always be in an inherently coherent state. Figure 2.3 shows an application written using this framework. The program gathers data from a sensor, computes a mean and transmits the result.

Time-keeping Solutions

The idea behind time-keeping solutions is to exploit the regularity of physical phenomena with predictable characteristic times to have an idea of the time elapsed between two instants. This high-level idea is quite difficult to apply in the domain of TPC because the device cannot sustain in any way the phenomenon which is taking place, as it is totally deprived of energy during

shutdown periods. The solutions in the literature, such as Tardis [Hes+16] do typically use the discharge of capacitors.

An example: Persistent Clocks

Hester et al. [Hes+16] developed two distinct systems to measure the duration of a power failure. Namely, Tardis measures the amount of charge in SRAM cells, while CusTARD relies on an external capacitor, which is left fully charged in case of a power failure, and whose state is measured again at the end of the power failure itself. These systems allow measuring time intervals lasting from a few seconds to several hours.

Adaptive Solutions

Several scholars devised mechanisms to make devices change their execution flow in order to maximize their results, following the available energy. One of the most popular ways this concept is implemented in the literature is to build energy-aware schedulers or runtimes, which tune the workload taking into account time and energy constraints. Examples of this class of solutions are Mayfly [HSS17] and InK [Yun+18].

An example: Mayfly

Hester et al. [HSS17] developed Mayfly, a task-based programming model in which a full program is split into self-consistent, semantically coherent portions of code which are executed atomically. Tasks are linked by data dependencies: one task can ingest the output of a set of other tasks. Moreover, the programmer can put a constraint on the degree of punctuality data must possess in order to be considered valid.

2.6.3 Comparison

The two classes of solutions, i.e. fail-safe and demanding, have complementary pros and cons, as each of the two does focus on different facets of the problem of TPC. The primary goal of fail-safe solutions is to ease the adoption of TPC making its difficulties transparent to the programmer. On the other hand, the utilization of demanding solutions requires a higher implementation effort, as developers have to design applications in a way faulty behaviours are tolerated. The fact that the complexity due to TPC is completely hidden to the programmer may show a drawback. As a matter of fact, the programmer of a transiently powered system could be tempted to code the same way he/she would do while programming a continuously powered system.
This behaviour can hinder the quality of the result. For example, it can be the case that an application queries a sensor, retrieves some data and then remains offline for a long time interval, at the end of which data are processed and sent to some sort of consumer. In this situation, the consumer would base its computations and decisions on an old and possibly outdated piece of information.

Demanding solutions have serious drawbacks, too. First of all, designing and implementing them requires a huge cognitive effort on behalf of developers, who must take into account the consequences brought by failures additionally to the normal difficulties of writing code.
Moreover, it is not always trivial to find use cases where failure is acceptable: a classic example is constituted by cryptography, where a single error or imprecision can ruin completely encryption or decryption.
However, we think that TPC has so many advantages (see Section 2.4) that it is worth exploring this domain and use it as much as possible.
However, using TPC is extremely difficult from various points of view (see Section 2.3). Therefore, we think that the best way to create applications which run effectively in the conditions enforced by intermittent executions is to tailor them to address all the issues that derive from the aforementioned conditions. As a consequence, we strongly believe in the potential of adaptive solutions. The scope of this work is to investigate the feasibility of adaptive solutions for this domain. To do so, we have developed a proof of concept application and evaluated its effects on the field.
The algorithm is based on the idea of applying to the domain of TPC techniques borrowed from the world of Approximate Computing, which will be introduced in the next chapter.

Chapter 3

Approximate Computing

3.1 Abstract

The diffusion of resource-hungry applications is greatly increasing the need for computational power while, on the other hand, the increase in the number of processors will not keep up with the demand [Mit16].
Approximate Computing (AC) comes to the aid in bridging the demand-offer gap. AC is the set of tools and techniques aiming to trade some degree of performance in order to make the execution of a given program cheaper.
As summarized in Figure 3.1, the interest of academia in this topic is quickly increasing (the number of yearly publications grew more than 3x from 2002 to 2019) [19ae]. As promising as it is, the practical application of Approximate Computing has a set of obstacles that hinder its actual diffusion [MS11].
First of all, Approximate Computing cannot be blindly applied to arbitrary programs: there exist several scenarios in which it is totally unusable, and others when it must be handled with much care because even small errors are intolerable.
Moreover, the literature offers several techniques meant to save different types of resources, at the cost of degrading various types of metrics. However, not all combinations are always available, so another issue is whether it exists a technique which matches the requirements of a given application.
Finally, the technical availability of an Approximate Computing Technique (ACT) is not enough to apply it: the usage of Approximate Computing can be fostered by frameworks which help developers handle all the practical details [Mit16].
The structure of this chapter follows this scheme: an initial section is dedicated to presenting the intuitions behind Approximate Computing, its advantages and disadvantages and the context which motivates these considerations.
Then, we will present a list of the most popular Approximate Computing techniques that can currently be found in the literature. We will frame them in the context of a taxonomy which extends a work by Moreau et al. [Mor+18].

Finally, we will present the existing Approximate Computing Control Frameworks, that is to say, the software instruments that can be used to set up and monitor an application which makes use of Approximate Computing. The frameworks will be inserted into an original classification scheme.

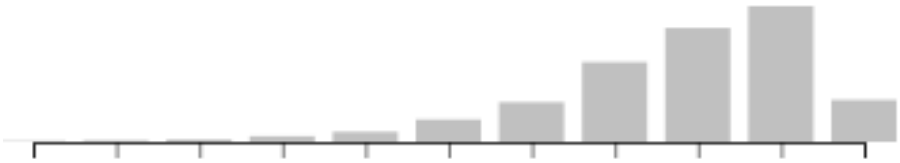


Figure 3.1: Papers about Approximate Computing, 1992-2019. [19ae]

3.2 Context

A major problem affects modern computer science, that is to say how to find suitable techniques to process and analyze every increasing amount of data. Venkataramani et al. [VRR16] highlighted that the benefits brought by semiconductor technology scaling are becoming less effective as time goes by, and at the same time the amount of data we produce is increasing: the World Economic Forum expects a 150x increase in the data production rate by 2025 [19u]. Mittal [Mit16] highlighted the fact that the amount of information processed by data centers will grow 50 times from 2016 to 2025, and meanwhile, the number of processors worldly available will grow by just 10 times. McGaughey [19h] noticed that the amount of data available in the world was increasing at a rate faster than Moore's law, i.e. twice every two years [19i] already back in 2011.
In light of these considerations, we can safely claim that the simple increase in the computation power is not a scalable solution to effectively process information. Approximate computing could be an effective instrument to ingest more and more data without paying an overwhelming cost.

3.3 Opportunities and Challenges of Approximate Computing

The high-level idea approximate computing relies on is to reduce the quality of computation in order to save resources. The advantages Approximate Computing brings about in terms of savings span all the assets necessary to carry out computation: from silicon area [Cho11], to memory accesses [OU19a], to computation time and energy [RR15]. In general, developers can probably find a technique to save whatever resource they need.
Moreover, it can be the case when an application requires nearly as many resources as there are available. If the consumption of an application is immediately above the threshold of availability of the resources, a small saving can

make the difference between successfully executing it or failing to do so. This situation is more common when the resources that are available are particularly scarce, like in the case of embedded devices. This makes the utilization of ACTs particularly promising in these domains.
The main drawback of AC is that some degree of performance has to be sacrificed. As Wyse et al. highlighted [19b], no unique definition of performance exists: several scholars tackled this issue by delegating this definition to programmers, or by using default metrics [Rin+15].
Finally, it is particularly difficult to understand when approximate computing can be applied. As a matter of fact, some classes of applications are intrinsically more sensitive to approximation and errors, in the sense that small approximations do greatly affect the final output of a computation.
Furthermore, different techniques offer variable features in terms of tunability, i.e. the degree of control developers can exercise on the level of approximation of executions.
This huge range of possibilities may become confusing, so Wyse et al. [19b] insisted on the importance of providing a general taxonomy which encompasses in an extensive way all the possible characteristics that can be used to define Approximate Computing techniques.

3.4 Opportunities

Approximate Computing trades performance for cost, and this characteristic makes it particularly suitable in the contemporary IT world for two main reasons.
The most popular techniques to process data are characterized by the fact that the correctness of their output is not a binary concept, but it is a value on an almost-continuous scale [19r]. In other words, modern workloads are not about calculating one numerically precise result, but their correctness is based on whether they can be functional to users [VRR16].
Therefore, data can be naturally processed in a fuzzy way without hindering their usefulness: in this sense, Approximate Computer can help, as its very purpose is to tune the precision of computation in order to further reduce its cost.
The second practical consideration to make stems from the nature of the data. Data coming from the physical sensors are catching an increasing interest from the world of industry: the market of IoT boasts a yearly growth greater than 15% and more than 70% of global executives forecast they will be a competitive differentiator by 2021[19v]. IoT devices are either sensors or actuators which interface with the physical world, and by their own nature they are not perfectly precise [19w] due to the unavoidable presence of noise [19s] and to the fact that the digital representation of a signal is not perfect [19t].
As a consequence, from a qualitative point of view, every application which makes use of physical sensors is not exact. Approximate Computing implies simply a quantitative change in the precision of the application.

Finally, another aspect of computer science that is gaining momentum is the utilization of robots and actuators: the market of robots is expected to grow with a rate of 21.5% up to 2026 [19x]. The utilization of robots and actuators in general is similar to the usage of sensors: several actuators output continuous measures, such as the temperature of a room. In these cases, the physical action of machines on the world cannot be perfectly tuned, hence it is an intrinsically inexact activity (Venkataramani at al. [VRR16]). Therefore, also this domain is a promising field to use Approximate Computing.
In conclusion, some of the most promising disciplines in computer science do not require to handle input or output in a perfectly precise way, and so there are no reasons of principle not to apply Approximate Computing to boost their cost/performance ratio.

3.5 Challenges

There are several application domains where it is impossible to apply Approximate Computing techniques. For example, Mittal [Mit16] cites cryptography and hard-real-time applications. Regazzoni et al. [RAP18] claim that Approximate Computing can be used only in situations where some degree of imprecision is permitted, and that even in those cases the very nature of some techniques in the world of AC opens up novel security threats, related to all the phases in the lifecycle of a device.
Another important criticality that hinders te usage of Approximate Computing is the fact that also applications that do theoretically allow some degree of approximation may be greatly damaged if ACTs are not applied correctly.
To be more specific, some ACTs can be applied to approximate specific types of instructions of functions. This aspect will be better discussed in Section 3.6.2, but a simple concrete example is the technique called Skipping Memory Accesses (see Section 3.6.2), which consists in guessing values instead of loading them from the memory, which can affect only instructions which make use of the memory. In particular, Esmaeilzadeh et al. [Esm+12] showed that one
Another major criticality that hinders the usage of Approximate Computing is the fact that there are portions of the code which are intrinsically more apt for approximation [Esm+12] [Sid+11]. The application of approximate computing techniques to the wrong sections may have dangerous and unpredictable consequences: as an example, the approximation of control flow can overturn the correct functioning of a program.
Also in the case when the correct sections are approximated, the quality of the output could become unacceptably low if the level of approximation is too high. To prevent this issue, Approximate Computing techniques should offer some kind of guarantee on the precision level of the execution. These guarantees can also be probabilistic, like in the case of ApproxHadoop [Goi+15], a framework where the error is bound statistically.
Another desirable feature of Approximate Computing techniques is tunability [Ima+19], that is to say, the capability to choose the degree of approximation.

A further step of complexity would be to dynamically change the degree of approximation online, dynamically adapting to the current context [Khu+15].
Both situations are affected by some kind of overhead: in the first case the developer of the technique must carefully analyze its properties, while in the second one the device itself pays an extra price for the approximation.
In conclusion, Approximate Computing cannot be applied in every context. Even when it can be applied, it is rarely the case that all the parts of an application can be approximated. Even when these hard requirements are met, there are some useful properties that should be guaranteed to approximate programs.

3.6 Taxonomy of Methods

Moreau et al. [Mor+18] developed a taxonomy to classify Approximate Computing techniques trying to assess their properties in terms of three main dimensions, namely Visibility, Testability and Flexibility.
As regards Visibility, some techniques can introduce errors at any time, while others only when specific instructions are executed: the first techniques are called Data techniques, while the latter ones are Compute techniques. Testability is the degree to which error can be measured during development and generalized to production. It is used between ACTs that introduce faults in a deterministic fashion opposed to the ones that do so in a nondeterministic fashion. Flexibility represents the minimum size of the code portions where it makes sense to apply a specific ACT. For example, some techniques affect single instructions, while others act at the level of entire functions.
Mittal [Mit16] completed an influential survey of the domain of Approximate Computing. In his work, he described the main methods that have been applied in this field, and he did also produce an overview of the systems to find the approximable portions of the code and to monitor the quality of the execution.
Venkataramani et al. [VRR16] wrote an extensive survey in which they face all the problems that can be met in the domain of Approximate Computing, from the hardware at the physical level to software and algorithmic issues.
This section is structured as follows: first of all, we will present a taxonomy to classify the existing techniques, obtained by distilling the main criteria that can be found in the literature. Afterwards, we will do a similar job for the methods to monitor and tune performance.

3.6.1 How to Implement AC: Dimensions of Classification

This section owes much to the work by Moreau et al. [Mor+18]. We did simply integrate it with some extra dimension: the original dimensions will be clearly distinguished in the following sections.

Hardware or Software

One fundamental distinction to make when speaking about a particular AC technique is whether it involves customized hardware or it acts only on the software. Mixed techniques are theoretically possible, but we could not find any in the literature. This distinction is implicitly made by Moreau et al. [Mor+18], who does separate hardware and software techniques but does not list this dimension among the fundamental ones. Mittal [Mit16] treats this distinction in an implicit way, too, by specifying which methods do require specific hardware.
We think that the distinction between purely software or hardware techniques cannot be neglected: the usage of customized chips greatly lowers production costs and improves the reliability of a device[19y], therefore, it is necessary to make designers aware that hardware techniques are more powerful but more costly to test.

Saved Resources

The main purpose of Approximate Computing is to reduce the quality of the result of a program in order to reduce the consumption of certain resources. When describing a particular technique, it is very important to make it clear what advantages it can bring, as they can range in a notable spectrum [Agr+16]. In particular, in the literature it is possible to find techniques to save memory [YAK18], computation time and hence energy [Sid+11] or transmission time [Zor+14].
It can be the case that the scarce resource varies according to the peculiar constraints in the deployment of specific applications running on fixed devices in determined scenarios, so this aspect of an Approximate Computing technique cannot be ignored.

Determinism

This dimension has been established by Moreau et al. [Mor+18]. It is used to distinguish whether the error produced by a given technique can vary given the same initial state of computation or not. It may sound unusual to speak about nondeterminism in a branch of computer science, as the behaviour of computers is in general deterministic. However, some ACT is intrinsically nondeterministic, i.e. it can cause different effects when executed many times. Nondeterministic techniques do typically involve a tight interaction with the physical world: for example, Voltage Scaling [Pop+07] consists in lowering the voltage of a power supply. Therefore, the impredictability is introduced by the physical phenomena themselves, which modify the behaviour of the very hardware components. Moreauet al. [Mor+18] stress the fact that nondeterministic methods are particularly challenging to test, because it is necessary to analyze the results of multiple executions to have an idea of their behaviour.
We think that another issue that can be produced by their unpredictability is their unforeseeable performance during actual operations.

Flexibility

This dimension has been defined by Moreau et al. [Mor+18]. Approximate Computing techniques can act at different levels of granularity, that is to say they can be used to approximate portions of code with various sizes.
For example, precision scaling is used at the level of a single number, the size of whose representation can be reduced to save resources [Rah+15a]. On the contrary, entire programs can be substituted with software which performs the same task at a lower cost, for example by means of a Neural Network [Esm+12]. Moreover, several possibilities exist in the middle of these extremes.
This dimension of classification can be useful to understand how tunable is a given technique, viz. how many levels of cost and performance it is possible to choose among.

Visible or Invisible

This dimension has been introduced by Moreau [Mor+18]. It is used to measure the observability of the effects of a given technique, that is to say whether and when it can introduce variations in the state of a computation.
In particular, there is one main distinction to make: while some technique can act in any arbitrary moment [AS18], while others take effect only in specific parts of the code [OU19a].
Like in standard dynamic systems, unobservability adds an extra layer of complexity to the analysis of a process. As a matter of fact, techniques that can affect the state of computation in any moment make it impossible to rely on the correctness of a program, so their effects are impossible to detect and correct.

3.6.2 How to Implement AC: Techniques

In this section, we will list the most important techniques that can be found in the literature in the domain of Approximate Computing. We included this section because we think it is necessary to concretely see the actual applications of Approximate Computing both to knowingly use them and to develop new ideas.
Each technique will be commented and framed in the taxonomy defined in the previous paragraphs (see Section 3.6). This list is not exhaustive: on the contrary, it is meant to show concretely some of the potentialities of Approximate Computing.

- Precision scaling: precision scaling consists in reducing the number of bits used to represent variables. This technique has been effectively used, to give an example, by Yeh et al [Yeh+07] in the domain of physics simulations. Their work involves the utilization of ad-hoc hierarchical floating point units, in the context of multi-core systems. The tuning parameter is the number of bits used in each level of the HFPU.

- Code perforation: this technique consists in skipping parts of the code. In principle, every instruction could be skipped, but some parts are more

promising than others. This is the case of loops: skipping loop iterations is particularly effective in case they are independent from one another [Sid+11].

- Skipping memory accesses: access to memory is one of the most expensive tasks [19z]. Several techniques have been designed to bypass this cost, for example by predicting it [MBJ14], by neglecting the access with low influence [Thw+14], or by interpolating similar values [SSE15].

- Memoization: the idea of memoization is to store the results of function calls and reuse them instead of rerunning the function with similar inputs [OU19a]. A similar idea is followed by Keramidas et al. [19aa], who focus on approximating arithmetic operations in the domain of graphics.

- Using Multiple Inexact Programs: Samadi et al. [Sam+14] developed Paraprox, a system which creates multiple versions of a given program by leveraging the approximability of known software patterns. A runtime component selects the most suitable one during the execution.
  A similar idea has been followed by Baek and Chilimbi [19ac], who proposed Green. It is a framework which learns the relation between the level of approximation and the quality of the output, and tunes the approximation providing statistical bounds on the error.

- Using Inexact or Faulty Hardware: various scholars applied the principles of Approximate Computing at the lowest possible level. Khang and Kang [KK12] created approximate, low-latency adders, while Kulkarni et al. [KGE11] designed an approximate multiplier. Venkataramani et al. [Ven+12] invented a method to synthesize approximate circuits adding extra "Don't Care"s in non-sensitive points. Ganapathy et al. [Gan+15] developed a mechanism to concentrate bit errors in storage towards lower-order bits, reducing the magnitude of errors.

- Voltage Scaling: when the power supply of a circuit is lowered, its consumption decreases at the cost of possible failures [Mit14]. This applies both to memories [Den+19] and to computing units. About the latter, Ahmed et al. [Ahm+19] noted that it is possible to find the optimal point of work for every voltage value. Rahimi et al. [Rah+15b] proposed a GPU architecture which reuses results of FPUs, whose power supply can be tuned to reduce its consumption while controlling the error rate.

- Iterative refinement: anytime algorithms consist in continuously refining an initially rough solution, improving its quality. San Miguel et al. [San+16] proposed a framework to process data streams with multiple computation steps. Each step yields partial outputs while ingesting more and more data, and partial solutions are fed to subsequent steps. Mangharam and Saba proposed to use anytime contract algorithms[1] for real-time

[1] Anytime contract algorithms are algorithms which can tune the quality of their execution according to a budget given before the execution itself.

applications on GPUs [MS11].

3.6.3 Control of AC: Dimensions of Classification

As anticipated in Section 3.5, Approximate Computing is hard to use. The first difficulty is to understand whether a certain application has any approximable part, and what portions of it can be approximated, making sure that the quality of the execution adheres to determined standards.
The techniques in Section 3.6 give an idea of the potentialities of Approximate Computing, but it can be useful to design systems to orchestrate their utilization to maximize their efficacy.
In the literature, it is possible to find several examples of such mechanisms. We designed an original taxonomy to classify them by considering three dimensions: whether they operate online or offline, their tunability, and the amount of human effort which has to be spent to use them.
In the next paragraphs, we will describe these dimensions, and finally, we will frame some popular method into the context of this taxonomy.

Online, Offline

In the literature, it is possible to find two broad classes of techniques to make use of Approximate Computing: some of them [Che+19] [Yaz+15] works by creating an approximate version of an application. The way these approximate applications are used are planned offline, without adapting any aspect of their behaviour during the execution itself.
On the other hand, other methods [GR14] [Khu+15] consist in monitoring the execution of an approximate program, adjusting parameters online.
These paradigms are not mutually exclusive: they can be applied in sequence without any problem.

Manual, Automatic

Vassiliadis et al. [Vas+16] note that current approaches to identify approximable portions of code do typically require programmers to manually help with this task. This is the case, for example, in [Sam+11] or [CMR13].
However, this process is tedious and prone to errors. Therefore, several works [Roy+14] [Rin+12] pursue the idea to analyze the code to understand which parts can be approximated without damaging the quality of the result.

Degree of Approximation

The third and last dimension which we use in this taxonomy is how tunable is an approximation, viz. whether it can choose how approximate an execution should be.
To give an example EnerJ, a tool by Sampson et al [Sam+11] allows to annotate variables as approximate, in a yes-no fashion. The assumption behind the paper is that annotating more variables as approximate the overall degree of error does

typically increase. Under this assumption, it is possible to control how much a program is approximated leveraging on the number of annotated variables.
On the other hand, Rely by Carbin et al. [CMR13] allows to directly indicate the amount of precision that can be lost by each function, in terms of metrics programmers can specify.

3.6.4 Control of AC: State of the Art

This section is used to list some of the techniques that can be found in the literature to tackle the issue of how to successfully apply Approximate Computing. The intuition behind each idea is explained, and the technique is then framed in the context of the taxonomy defined in Section 3.6.3.

State of the Art: Automatic, Manual

There are several ways to identify approximable portions of the code. In this paragraph we will go through two of them, the first one fully manual and the second one highly automated.
The first one is EnerJ, by Sampson et al [Sam+11]. It is an extension for the Java language which allows programmers to manually tag variables as approximate. The system where the execution takes place is fully responsible for the practical handling of approximate variables: the authors of the paper created a Proof of Concept where approximate computation makes use of approximate memory and imprecise functional units, but in principle the framework can make use of arbitrary Approximate Computing techniques.
The second work is dco/scorpio, by Vassiliadis et al. [Vas+16]. It is a framework to analyze arbitrary C/C++ code in order to compute which portions have the highest influence on the final result.
Given an interval of possible values of a function, the framework computes the range of the values in the output: in case the output range is much bigger than the input range, the input is considered as highly influential and treated in a precise way. This allows the framework to restructure the code, splitting it in tasks which are approximated to various degrees.

State of the Art: Online vs Offline

The execution of an Approximate program can take place following a static predefined control flow, or its behaviour can be adapted to the context where it is taking place.
Both techniques have advantages. Namely, dynamically adapting the level of error to the instant needs of users or the availability of resources allows to maintain the best possible quality/cost balance [GR14]. On the other hand, stating a fixed schedule frees the execution from any kind of overhead due to monitoring and tuning [Sam+11].
One perfect example of a fully static technique is Axilog, a work by Yazdanbakhsh et al. [Yaz+15]. It is an extension to the Verilog language to enable the

design and the synthesis of approximated hardware components.
On the opposite side of the spectrum, we can find a framework created by Ringenburg et al. [Rin+15]. It consists in a technique to monitor the quality of an execution, improving it when it falls behind a threshold.
The first proposal of the work by Ringenburg et al. can be used on applications which produce a high number of distinct outputs: they suggest to perform both exact and approximate execution for a subset of these outputs, and to use the difference between their qualities as a proxy of the quality of the execution.
This difference can be computed either by means of a default distance or by using functions provided by the programmer.

State of the Art: Degree of Approximation

As noted in Section 3.6.1, approximate computing techniques may act on chunks of code with different sizes. This concept is related to the precision with which it is possible to control the quality of the execution, i.e., the minimum size of the precision step.
The tunability of Approximate Computing is a desirable property to follow a dynamic workload [XS18].
This necessity is satisfied by several Approximate Computing control frameworks. For example, Carbin et al. [CMR13] developed Rely, a framework where developers can specify the precision requirements of each function, which is then transformed according to these requirements.
However, not every Approximate Computing technique offers good tunability properties. As seen in Section 3.6.1, the tunability of ACT is a characteristic feature which can be used to classify them and distinguish them. Therefore, other works insist on the fact that tuning the degree of approximation does not allow to fully separate the definition of the approximate execution from its implementation [Sam+11], as it restricts the scope of the framework to a subset of the available techniques.
Other works are intrinsically more narrow, as they focus on specific applications or techniques, and so they do naturally offer a low degree of tunability and flexibility: as a matter of fact, they can only be applied to target applications. This is the case of Rumba [Khu+16], a system to control the utilization approximate hardware accelerators.
It consists in predicting the quality of the output of an approximate hardware accelerator based on its input, and using this information to decide whether to use it or not. In other words, its output is a binary decision, without an associated indication of the precision of the execution.

Chapter 4

Approximate Intermittent Computing

4.1 Abstract

In this chapter we will introduce the very topic of the thesis, that is to say, the application of Approximate Computing Techniques in the domain of Transiently Powered Computing, obtaining Approximate Intermittent Computing. In the first part of the chapter, we will highlight the synergies between the two domains that make it particularly promising to combine these two worlds, and in the second one we will propose a framework to study and optimize the application of Approximate Computing techniques, highlighting the differences between traditional scenarios and Transiently Powered scenarios.
It looks very promising to apply Approximate Computing in the domain of Transiently Powered Computing for two main reasons. First of all, Transiently Powered Devices are often deployed to have a tight bond with the physical world, which introduces an inevitable amount of noise to corrupt the input.
Moreover, several Transiently Powered Computing applications make use of machine learning techniques to extract knowledge from data. These techniques are typically probabilistic, and so they naturally contain some degree of imprecision. Therefore, ACTs do only quantitatively increase this imprecision, but do not radically alter their functioning. .
After we will have introduced the reasons why it is promising to combine these techniques, we will propose a method to make this combination more effective: in particular, our focus will be on how ACTs can be tuned in a clever way to produce the best result possible according to the available resources.
We will argue the fact that in traditional computing scenarios ACTs are controlled in a way that is not optimal for TPC. To support this claim, first of all we will show that in traditional computing the objective pursued in the literature is to maximize the savings while maintaining the execution above a certain threshold in terms of quality.

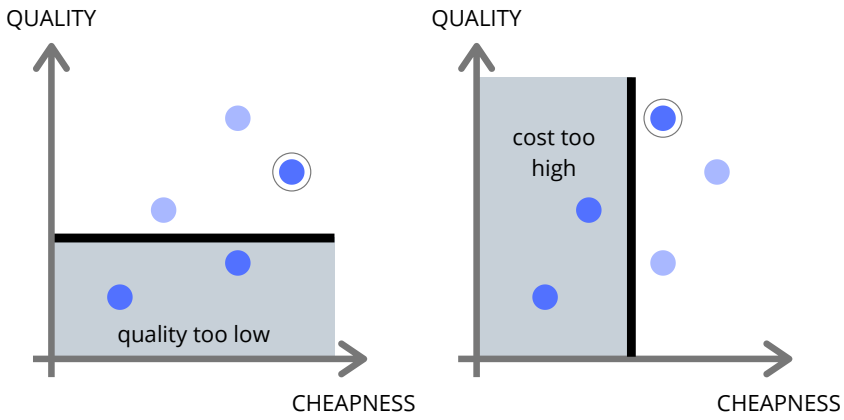


Figure 4.1: Criteria to use AC: traditional approach (left), TPC approach (right).

On the other hand, we will propose to apply the dual of this reasoning in TPC, that is to say, to maximize the quality of the result while not consuming more resources than the amount that is physically available. Both analyses will be complemented by concrete examples of how these concepts can be applied.

4.2 Opportunities

There are three main reasons why Approximate Computing techniques combine well with Transiently Powered applications. First of all, Transiently Powered devices are typically deployed in order to support the execution of applications in the domain of sensing, which process inputs coming directly from the physical world.
This kind of input that is physiologically processed by this kind of applications is intrinsically noisy, so Transiently Powered applications are typically not meant to guarantee a fully precise output. Therefore, it is theoretically possible to increase the magnitude of the error changing the behaviour of the application only quantitatively and not qualitatively, to avoid tearing it apart.

Second, Transiently Powered applications are also affected by another source of imprecision: as a matter of fact, Transiently Powered nodes are often deployed at the edge of networks due to their own unreliable nature: it would be risky to deploy in a TPC way a central hub or server, responsible for the traffic or the services towards a multitude of other hosts.
On the other hand, it more than makes sense to use TPC devices as peripheral

nodes, with the goal either to receive or transmit information to central hubs. As transmission is a very costly operation, TPC nodes have to extract knowledge from the data and cannot send all they sense. Typically, data are processed using machine learning techniques, that in general produce fuzzy or probabilistic outputs, which are not fully precise.
Using ACTs does only increase this imprecision, but does not introduce new sources of errors.
Finally, ACTs can make it possible to carry out more executions by cutting down their cost. This can produce a qualitative difference with respect to the scenario when applications are executed in a precise way, paying their full cost. As a matter of fact, it can be the case that there are simply not enough resources to run an exact program, but cutting down the cost it becomes possible to obtain some result.

4.2.1 Low Damage: Bonds with the Physical World

TPC applications have a tight bond with the physical world: they need to extract energy from the environment to work, and so they do necessarily have some kind of mechanical or electromagnetic interface with the environment for the energy to flow in.
Therefore, TPC devices are physically very close to the physical world, and so it is natural to use them to deploy applications that make use of sensors or actuators, which must be close to the physical entity they need to measure or influence. This claim is supported by experimental evidence: a survey by ElectronicDesign [20b] shows that the interest of industry and institutions towards applying energy harvesting in industrial contexts has created a market worth 250M EUR (2017).
As a consequence, TPC applications are normally used to process data that come from electronic sensors, and to send the result via wireless transmission [Sen+17].
Electronic sensors are always affected by some kind of noise [20d], which limits their accuracy. The error in input measures is by definition propagated all along the data flow of a program, hence impacting the quality of the output [20e].
So, TPC applications are physiologically imprecise due to their very nature.
When one applies an AC technique to this kind of program, he/she can typically select the quantities that are being influenced: if the developer limits the approximation to the variables that are already affected by the noise in input, he/she is quantitatively increasing the amount of noise, but it is not a necessary consequence that the application breaks.

4.2.2 Low Damage: Imprecision in Data Processing

As we proved in the previous Section 4.2.1, TPC nodes are typically embedded devices that are used to sense data, deployed at the edge of networks.
In modern computer science, novel algorithms and hardware resources made it

possible to apply machine learning on embedded devices [20f].
This choice is particularly convenient for many reasons, which follow from the fact that the data remain closer to the place where they are produced. For example, privacy and security can easily be managed when data are physically present on a local machine. Moreover, another indirect advantage for the infrastructure is that when the global amount of information flowing through a network decreases the network itself is under less stress, reducing the probability of potential faults.
There is another advantage of transmitting less data, which is particularly important in TPC applications. Since the transmission is very costly in terms of energy, the application of machine learning techniques can be very useful in TPC contexts to extract meaningful information from raw data, and send or store just the final result of the processing.
However, machine learning algorithms do typically output values that can be interpreted as probabilities [20g]. In other words, they are not deterministic. Moreover, in real-world applications input data are often noisy, and do several techniques have been developed to deal with errors in the input [KM99].
Therefore, it theoretically possible to increase the severity or the number of errors in machine learning applications without damaging the way they work. AC is meant exactly to introduce arbitrary errors in programs, so we can safely conclude that it can be applied to machine learning techniques, also to the ones in TPC deployment scenarios.

4.2.3 Advantages: Unlocking More Executions

AC has been invented to save resources on machines which are continuously powered, and it is typically used to reduce the cost of applications that need to be executed and can be executed even in their exact version. This is possible because continuously powered machines can execute also the exact version of most applications, so it is feasible to select any arbitrary degree of approximation without running the risk to run out of computational resources because the application has not been approximated enough.
In the context of TPC, it can happen that the available resources are simply not sufficient to run a task, and so reducing its cost becomes mandatory to obtain a result. This is particularly true when considering the energy that is available for computers to use, which is a variable out of the control of the developer, who cannot, therefore, make any kind of assumption on its availability over time.
The availability of energy has a direct impact on the possibility to execute a program: a program can be executed if and only if the amount of energy that is available is higher than its cost.
Therefore, programs cannot be executed even in the situation when the available energy is immediately lower than their cost. In this case, ACTs can make a huge qualitative difference: they make it possible to tune the cost of a program and to unlock executions that would have been impossible otherwise.
This effect makes it promising to use ACTs in TPC contexts. We will spend the rest of the chapter to investigate how ACTs can be used in TPC in the

most effective way: in particular, we will formalize the qualitative idea that the goal should be to maximize the quality of the output of TPC applications while maintaining their cost below the resources that are physically available.

4.3 An Optimization Perspective

In general, ACTs can be controlled to pick the best combination of cost and quality when executing an application.
This tuning operation can be performed by leveraging on parameters that are specific for every technique, as explained in Section 3.6.2. Some technique can be tuned offline, while others can be adjusted at runtime, but in both cases it is necessary to choose the optimal value of the tuning parameter to maximize the efficacy of AC in terms of cost-performance trade-off.
The way AC is tuned has a major impact on the usefulness of the results that are produced by approximate applications: it can be the case that they are too precise and so they were produced wasting resources, or they can be so erroneous they are useless, and so it is necessary to design clever methods to carry out this crucial operation.
It is possible to interpret the choice of the value of the tuning parameters as an optimization problem, where the objective function to maximize depends on the needs of the user, i.e., on his/her requirements in terms of quality of result and budget to obtain it.
We will formally discuss how to design optimization problems that represent how to tune ACTs in the next sections.

4.3.1 Tuning Approximate Computing: an Optimization Problem

Optimization is the selection of a best element (with regard to some criterion) from some set of available alternatives [19ad].
Optimization problems can be formalized as

$$maximize\ f(x), x \in X, subject\ to\ g(x) \leq 0, h(x) = 0 \tag{4.1}$$

where X is a subset of R^n, f, g and h are scalar functions. f is called the objective function, while g and h are the constraints.
In the case of Approximate Computing Techniques, the elements to pick will be the parameters to tune the techniques themselves. As for the objective function, the criterion will vary case by case: since it is not trivial to find an optimality criterion that is provably the best for ACTs, we will extensively discuss this issue in the rest of the chapter.
However, it is possible to make an empirical observation: the vast majority of existing ACTs is being used in traditional, continuously powered computing pursuing the goal of minimizing the cost of computations, while maintaining the quality of their output above a given threshold.
In other words, the inverse of the cost plays the role of f(x) in Equation 4.1,

Figure 4.2: An example of image with decreasing quality, from [Sam+13]

while the inverse of the quality of the output plays the role of g(x) in the same equation.
We will support this claim in the next sections, showing examples taken from the literature.
Our goal is to show how ACTs are being used right now, in order to convincingly argue that the principles that guide their utilization cannot be used in TPC contexts: we propose to adopt a dual paradigm, that is to say to maximize the quality of a computation given a constraint in terms of resources.

4.3.2 State of the Art

Approximate Computing enables to obtain arbitrary cost-performance blends while executing programs. Several works in the literature [Sam+14] [Goi+15] [Sam+13] tune the cost-performance trade-off following the same philosophy, viz. reducing costs as much as possible while maintaining the quality of the result above a given threshold. Our explanation, supported by different papers ([Yeh+07], [Sid+11]), for the popularity of this kind of approach is that Approximate Computing is successfully applied in domains where it is useless to push the improvement of the quality of a result above a certain point.
Let us take the example of imaging: Figure 4.2 shows different versions of an image, each one encoded with a different quality. All the images above a given quality threshold, namely around 90%, are indistinguishable. The qualitative consideration that all the performance above a threshold is basically useless has been formalized by Yeh et al. [Yeh+07]. The idea of their work was to reduce the number of bits used to encode numbers in physics simulations for the entertainment industry. The primary goal of this kind of application is to produce simulations that are realistic when seen by a spectator or a player, but it is not mandatory to create results that mirror perfectly the reality.

Physics simulations should follow the law of conservation of energy: the overall energy possessed by the simulated objects should remain constant over time. This property does not necessarily hold when simulations are approximated: it can be the case that the overall amount of energy increases or decreases over time.
Yeh et al. [Yeh+07] noticed that the non-conservation of energy is not a problem until the point this difference becomes bigger than a threshold around 10%, after which people start noticing the incorrectness of the simulation.
Therefore, they proposed to continuously lower the precision of the program, until the threshold of imprecision is met, following exactly the principle of lowering the cost as much as possible while maintaining the quality above a threshold.
A further motivation to consider performance as a constraint is that excessive approximation may cause unexpected behaviours or crashes [Sid+11], which can be avoided by imposing quality of result to be high enough.

4.3.3 Mapping ACTs to Optimization Problems

In the previous paragraphs, we have argued that Approximate Computing makes it possible to reduce at the same time the cost and the accuracy of certain classes of applications.
Afterwards, we have shown a framework to select in the best possible way the cost and the accuracy of executions.
Finally, we have claimed that several works in the domain of Approximate Computing can be encased in this framework.
In the following paragraph, we will concretely show how the tuning of several ACTs in the literature can be expressed as optimization problems. In each example, we will explain the choices made by the developer that demonstrate the technique has been designed with the idea of considering quality as a constraint and savings as a goal. As mentioned in Section 4.3.1, optimization problems are characterized by a set X of possible choices, an objective function which maps every point in X to a scalar value, and one or more constraints. Constraints are composed by functions which map points in X to scalar values: they are used to determine whether a point P is feasible or not, by checking the sign of the image of P. The value of all constraints for a given point must belong to determined intervals for the point to be a feasible solution.

Domain

Optimization consists in picking the one in several alternatives that is the best according to a fixed criterion. Therefore, the first step to map an Approximate Computing Technique to a mathematical program consists in defining the possible choices that can be made thanks to the technique itself. The set of possible alternatives is called X in Equation 4.1. Every $x \in X$ is a tuple of values, which can be continuous or discrete independently on one another. For example, continuous choices can be physical quantities such as voltage values or time intervals, while discrete choices can be yes-no decisions. This dimension

is strictly related to the physical implementation of Approximate Computing Techniques: Section 3.6.2 gives an idea of the possible choices that can be taken when using specific techniques.

Objective Function

Intuitively, the objective function of an optimization problem describes a characteristic that it is desirable to maximize or minimize. When using ACTs, the usual approach is to minimize the consumption of a given resource.
Therefore, it is possible to obtain an objective function from an ACT by computing the amount of a given resource that will be consumed as a consequence of a given decision. The result of this operation is a mapping from X (viz. decisions to take) to real values (viz. resource consumed as a consequence of the choice).
The taxonomy presented in the previous chapter contains the dimension "saved resource" (see Section 3.6.1), which can be used to precisely define the quantity that must be measured to assess the performance of individual ACTs. As a consequence, there is no theoretical difficulty in performing this operation: to give an even more concrete example, we will explicitly show some mapping in the following paragraphs.

Constraints

The constraints of optimization problems represent restrictions of the domain of possible choices, which make points in set X unavailable. In the existing literature, the vast majority of ACTs considers as unavailable the decisions that lead to a huge loss in terms of quality. As we have explained in Section 4.3.2 Precision Scaling [Yeh+09] and Loop Perforation [Sid+11] are examples of this paradigm because both are based on the idea of minimizing the cost of executions while maintaining quality above a threshold, that represents a constraint. Further instances will be provided in the next section.
It is necessary to clarify the concept of performance to create a formally valid definition of the constraints that bind ACTs: this issue has been addressed in the literature, as several authors needed precise metrics to assess the correctness of their work.
A popular approach is to measure the distance between the output of an exact execution and the output of an approximate execution by means of various metrics: the higher the distance, the lower the performance. As highlighted by Siridoglu et al. [Sid+11], this process is logically composed by two distinct steps: first of all, it is necessary to select the variables which are for some reason significant and representative of an execution. Then, it is necessary to choose a metric which maps pairs of point in the chosen space to a single scalar value.
In the following section, we will provide several examples of methods that have been used to measure the degradation of performance.

4.3.4 Some Examples

This section contains some examples of Approximate Computing techniques mapped to correspondent optimization problem. For each of them we will list the domain of the variables, the range of the objective function and the structure of the constraints.

Precision Scaling

Precision scaling [Yeh+07] consists in reducing the number of bits used to represent numbers. This technique can be used to carry out approximate operations on ad-hoc processing units.
The domain X of the tuning parameter is the set $\{1..N\}$, where N is the maximum number of bits that are occupied by a number in exact programs.
Using this technique has a positive impact in increasing the throughput of a program in terms of latency of the instructions: it can turn several operations into trivial operations, such as multiplication or division by one. Moreover, it allows to efficiently cache the results of complex operations.
As a consequence, it allows to directly reduce the number of cycles that are required to execute a program, hence its energetic cost.
According to the needs of the developer, the objective function can be either the time necessary to run a program, or the energy necessary to run it: in both cases, the goal is to minimize the chosen quantity.
As for the constraint, they are application-specific. Yeh et al. [Yeh+09] used it in the domain of physics simulations for the entertainment industry. They found out that a simulation remains plausible to the eyes of the spectator if certain physical properties hold: in particular, they found out that if the overall energy possessed by the objects in the simulation remains constant in time the simulation remains plausible.
Therefore, they imposed the constraint that the difference of the overall energy of the simulated objects in two consecutive instants of time must not vary more than 10%.

Code Perforation

Code perforation consists in skipping certain instructions to make executions cheaper and faster. A popular way to practically implement it is loop perforation [Sid+11], that is to say, skip some iterations of loops. In the original paper by Siridoglu et al. [Sid+11] the decision to take is the fraction of iterations to skip in every loop present in the code.
Let N be the overall number of loops in the code. Then, the domain X of the decisions to take is $[0, 1]^N$, meaning that for each loop it is necessary to choose the fraction of the iterations that will be skipped, so it is necessary one number between 0 and 1 for each loop in the code to represent all the choices that are taken.
This technique has a direct impact on the number of instructions that are executed by an application. Therefore, it can reduce the time and the energy used

by a program. So, the objective function is to reduce as much as possible either the time or the energy consumed.
The constraints that this technique has to meet are application-specific: in the original work that introduced this technique, Siridoglu et al. [Sid+11] tested it on the benchmark suite PARSEC1.0 [19ab], composed by several tasks spanning from image recognition to portfolio pricing. For every algorithm, the idea was to measure the distance between the output of an exact program and the output of a program approximated by the application of loop perforation.
Developers could set a threshold for the maximum distance that they could accept between the exact and the approximated outputs. This threshold was manually tuned by the developers, and then an automatic exploration algorithm searched for the best combination of perforations that could produce an acceptable result while maximizing the number of skipped instructions.

Skipping Memory Accesses

The idea of Load Value Approximation is to skip access to memory. This result can be achieved using different techniques, some of which are listed in Section 3.6.2. For the sake of brevity we will delve deep in just one, that is to say, Load value Approximation [MBJ14].
The goal of this technique is to mitigate the effect of cache misses: in case a cache miss happens, the program obtains immediately a value without waiting for the lower levels of the cache to load the exact result.
The result is just a guess on the real value that would have been obtained by the exact program, obtained using a custom hardware component conceptually similar to a lookup table.
The domain X of the decisions to take is ideally a tuple $\{0,1\}^N$, where N is the number of accesses to memory that take place during the execution of a program. As a matter of fact, for each memory access it is necessary to choose whether it can be skipped or not.
In practice, San Miguel et al. [MBJ14] used a preliminary training phase to understand the instructions that could be safely skipped without affecting the quality of the result.
The objective function to minimize is the time or - equivalently - the energy used to execute a program. The effect of reducing the time and energy consumption can be obtained by skipping as many memory accesses as possible.
As for the quality of the output of an approximated program, it is application specific: San Miguel et al. tested this technique on the benchmark suite PARSEC 3.0 [20t], that consists of 13 different multithreaded algorithms. The algorithms are very different from one another in terms of domain and scope, so it is necessary to use different metrics for every problem.
The paper defines such metrics to measure the distance from an approximated output and its exact version, and states that the metrics are built in such a way that errors in the order of 10% are generally acceptable.
Therefore, also in this situation we find a work where the authors aim at maximizing the savings while keeping the quality of the output good enough.

Memoization

Several classes of functions are stable, in the sense that small perturbations on their input do not provoke huge changes in the output. This characteristic can be exploited to store their results to reduce the number of times they are executed [OU19b].
The decision to take is whether a specific input should trigger the execution of a function, or it is possible to reuse an old result. Formally, it is a tuple $\{0,1\}^N$, where N is the number of function calls in the program. If the i-th number is 1, then the corresponding function call should be approximated. Ono and Usami [OU19b] developed a technique that uses the degree of similarity between a new input and an input that has been used in the past. In particular, if a new input is similar to an old input, the corresponding result that has been computed using the old input is reused.
The goal of this technique is to save energy and computation time: this result can be obtained by maximizing the overall number of instruction that are skipped. Formally, the metric to maximize is the number of clock cycles that are not executed in the approximated program.
As for the constraint function g, it is application-specific like in the previous examples. Ono and Usami tested this technique by approximating the application of a filter on an image, and they compared the image produced by the approximated program with the one produced by an exact version of the program. The distance between the outputs is computed by using Peak Signal-to-Noise Ratio and Structural Similarity. The technique developed by Ono and Usami allows to manually select the threshold, selecting the tolerance in a span from $\pm$1dB to $\pm$30dB. This tolerance is used to select the minimum degree of similarity that must exist between two inputs to allow a function call to be skipped. Also in this situation, we have a technique that aims at saving as much as possible, focusing on the number of clock cycles, while maintaining the quality of the output above a threshold selected by the developer.

Using Multiple Inexact Program Versions

The idea of using multiple version of a program has been investigated by Samadi et al. [Sam+13]. Three main factors impact the performance of GPUs: the number of atomic operations that must be executed at the same time, the limited amount of bandwidth that is available and the number of threads that can run at the same time.
Starting from this consideration, they developed a static compiler that approximates CUDA code, by leveraging on the three parameters above. The compiler creates different versions of the program, and a runtime environment is used during the execution of the program to select the version that is the most appropriate one.
The choices to take using this technique are divided in three main classes: the first decision is whether to skip an atomic operation or not, to reduce the number of conflicts. The second decision is whether to pack various input elements

for the transmission using lossy compression techniques to reduce the consumption of bandwidth. Finally, the last decision to make is whether to skip the computations that should be carried out by inactive threads.
Formally, the domain is the cartesian product of three sets: the first one is $\{0,1\}^N$, where N is the number of atomic operations in the code. If the i-th number is 1, the corresponding operation must be skipped. The second set is $\{0,1\}^M$, where M is the number of transmissions to perform: again, if the j-th number is 1, the content j-th transmission must be compressed. Finally, the third set is $\{0,1\}^P$, where P is the number of threads. If the k-th number is 1, the k-th thread can be skipped.
The goal of this technique is to minimize the time required to execute a program. Formally, the quantity to minimize is the number of clock cycles from the beginning to the end of the execution.
The quality constraint g is again application-specific: programmers must feed the approximator with a metric that quantifies the distance from the approximate version of a program and the exact output. Programmers must also provide a threshold for the metric.
Also in the case of Using Multiple Program Versions we have a technique where the goal is to optimize the savings, in particular reducing the execution time, while upholding with a quality threshold set by the user.

Using Inexact or Faulty Hardware

Inexact hardware has fewer design constraints than exact chips, so it can achieve better performances in terms of speed, power consumption and silicon area. In the literature, it is possible to find this principle applied to various classes of hardware components (see Section 3.6.2). We will analyze in detail the case of undersigned multipliers architecture. Kulkarni et al. [KGE11] designed a hardware component to perform approximate multiplications, composed by 2x2 building blocks.
The decision to take is how many of the building blocks of the multiplier have to be substituted with approximate multipliers: formally, the domain X of the decision to take is the set $\{0,1\}^N$, where N is the overall number of blocks in the multiplier. If the i-th number is 1, the i-th block must be approximated.
The primary goal of undersigned multipliers architecture is to save power: experimental results show that savings between 30% and 50% are possible.
In particular, the objective function is the amount of energy that must be spent to execute a series of benchmark programs.
This goal can be obtained by approximating all the building blocks. However, the authors set a quality constraint to prevent the design of circuits that are too imprecise.
In particular, they measured the distance from the output of an exact circuit and the output of a circuit designed using inexact hardware, quantifying it using Signal to Noise Ratio (SNR), and they enforced the constraint that SNR must remain below 3%.

Voltage Scaling

The voltage level of their power supply has a major impact on the power consumption of integrated circuits [Ion18]. Therefore, reducing it can create serious advantages in terms of energy consumption.
This technique has been pioneered by Xie et al. [XMM03a], and extensively used May et al. [MS16].
The idea of the technique is that when the voltage of the power supply of a circuit is lowered, both the clock speed and the power consumption of the circuit decrease. Tuning the value of the voltage, it is possible to obtain major savings in terms of energy.
The decision to take in every moment is the value of the voltage. Formally, it is a tuple $[0, V_{max}]^T$, where V_{max} is the maximum value of the power supply and T is the number of discrete moments in the execution. The value of the i-th element represents the value of the power supply during that moment.
The goal of this technique is to minimize the energy consumption of the execution.
The constraint to meet is the timeliness of the execution: in the work by Xie et al. [XMM03a], they developed an explicit optimization problem where some instructions are bound to terminate within a given point in time.
May et al. [MS16] followed a similar approach but relaxing the constraint: his work allows the developer to set a threshold to indicate the fraction of tasks that can take longer to execute than stated by their formal deadline.
Voltage scaling has also been applied by Rahimi et al. [Rah+15b] to reduce the consumption of GPUs. The domain is the same as in the previous case: in every instant, one must decide the value of the power supply. Also the goal is equivalent to the previous case: the energy consumed by an execution must be minimized.
However, the constraint to meet is different: as a matter of fact, Rahimi and his colleagues proposed to scale the voltage at levels so low that faults and errors can arise at the level of the hardware, corrupting computations and producing imprecise outputs. They proposed to measure the Hamming distance between the exact and the approximate output, and to maintain it below a threshold that depends on the number of bits of the output.
In both the situations, we can observe that the scholars designed techniques that aim at reducing the power consumption of programs while keeping their outputs accurate enough.

Iterative Refinement

Anytime algorithms work by continuously improving a solution by executing more and more steps of a computation.
The decision to make is when to interrupt the computation: the number of steps to run can be planned prior to the execution [MS11], or the application can be interrupted without notice.
When using an iterative algorithm, the decision to take is the number of steps to

execute: formally, it is an integer number $n \in [0..N]$, where N is the maximum number of steps in the algorithm.
The objective of this technique is to minimize the number of steps that are executed, to save computation time and energy. This technique has been explored by San Miguel et al. [San+16], who investigated the effectiveness of Iterative Refinement in different scenarios. Their goal was exactly to minimize the overall number of clock cycles spent by an algorithm.
Like in some of the previous examples, they put a constraint on the quality of the execution: they measured the distance between an exact algorithm and an anytime algorithm interrupted before its final step.
In particular, they use a threshold on Signal to Noise Ratio to assess whether execution is good enough.

4.4 Merging Two Worlds

In this section we will plunge into the very topic of this work, that is to say the application of Approximate Computing Techniques to the domain of Transiently Powered Computing.
In the previous chapter, we have provided an introduction to the domains of Transiently Powered Computing (see Chapter 2) and Approximate Computing (see Chapter 3). Then, we have shown that the application of Approximate Computing techniques can be seen as an instance of peculiar optimization problems, which aim at minimizing the cost of a program while maintaining an acceptable quality.
This paradigm has not been designed with TPC in mind, so we will propose a small modification to adapt ACTs to the domain of TPC: in particular, we will suggest to use available resources as a constraint and quality as a goal. We have shown that in the current literature the predominant approach is to save as many resources as possible as long as the quality of the computation remains above a given threshold.
Our interpretation is that in traditional computing it is feasible to impose a hard constraint on the quality of the result, because normally it is physically possible to obtain even an exact result. Several papers ([Sid+11], [Rah+15b]) compare approximate executions with their exact version, showing that at least in simplified cases it is possible to perform exact computations.
This is not the case in TPC scenarios: in case the environment does not make energy available for a certain amount of time, no computation at all can be carried out. Consequently, any hard constraint on the quality of the result that would be produced in the given time interval would be violated, so it is not meaningful trying to enforce unachievable results.
As a consequence, we suggest that this paradigm should be abandoned to maximize the efficacy of ACTs when applied in TPC scenarios. In particular, we think that the quality of a computation should be considered as the objective function to maximize while upholding with the resource consumption that is

physically possible given the conditions of the environment, especially the available energy. As a matter of fact, energy is one of the most fundamental resources and, at the same time, the one over which developers have the lowest control. This is extremely problematic because energy is indispensable for every execution: if it runs out, no result at all can be produced.
The formulation in Equation 4.1 can still be used to frame the problem of optimizing the behaviour of Transiently Powered applications. However, the mapping method presented in Section 4.3.3 must be altered to take into account the proposed perspective shift. In particular, the functions that represented constraints in traditional scenarios become goals in Transiently Powered situations and vice versa.

4.4.1 Mapping ACTs to Optimization Problems in TPC

In this section we will propose a list of optimization problems that express how ACTs can be applied in TPC contexts to be maximally useful. We will follow again the framework shown in Equation 4.1. In particular, we will specify the domain, the constraint and the objective function of each optimization problem. Our idea is that constraints and objective should be swapped when passing from a traditional scenario to a TPC scenario. We will exactly do so, and we will need almost no extra step compared to the reasoning in Section 4.3.3 and following. We recall that the domain X is the set of the decisions to take, the objective function maps each point in X to a real number. As for the constraints, each constraint is composed by a function which maps each point in X to a real number: the image of a point X must lie in a specific interval for the point to be admissible.
So, given an optimization problem, both the objective function and every function included in a constraint have the same structure in terms of domain and codomain.
Therefore, from a formal point of view, they can be swapped without creating problems from the syntactic point of view.
Let us take a look at the semantic consequences of this operation. Let

$$maximize\ f(x), x \in X, subject\ to\ g(x) \leq 0 \tag{4.2}$$

be the original optimization problem. For the sake of simplicity we ignored h(x). When we swap f(x) and g(x) we obtain

$$maximize\ g(x), x \in X, subject\ to\ f(x) \leq 0 \tag{4.3}$$

recalling that g(x) represents the imprecision of the execution, and -f(x) represents a resource consumption, we need to alter Equation 4.3 to make it meaningful: in particular, it is sufficient to turn it into

$$maximize\ -g(x), x \in X, subject\ to\ f(x) \leq 0 \tag{4.4}$$

Equation 4.4 means that the goal of the optimization problem is to minimize the imprecision of a result, while maintaining the resource consumption below

a threshold.
Equation 4.4 is coherent with the needs of TPC, and it has also another additional advantage. Since it makes use of the same functions of the original optimization problem, it is extremely simple to formulate an optimization problem suitable for TPC if the original problem is available.
In the next section we will show concrete examples of problems that follow this formulation, explaining in detail how to design optimization problems for the domain of TPC. The structure of the section will mirror the one of Section 4.3.3. We will initially detail how to delineate the domain X, the objective function -g(x) and the constraint function f(x).

Domain

Optimization consists in selecting the element in a pool of possibilities which is best according to a set of arbitrary criteria. In this discipline, the domain of an optimization problem is the global pool of available possibilities to pick.
In the case of Approximate Computing techniques, the choices that can be taken are the values of the parameters that are used to tune them, independently from the application domain or the context where they are applied. Therefore, there is no difference with respect to the framework presented in Section 4.3.3.

Objective Function

The objective function is used to quantify the utility brought by every single element of the domain. In light of the fact that computational resources are extremely scarce in Transiently Powered Computing, the objective of a developer designing a TPC application is to squeeze every single drop of performance out of the limited assets at his/her disposal.
As a consequence, the objective function is an application-specific metric which measures the quality of the output of a program. In traditional scenarios, this metric is treated as a constraint, i.e. executions are bound to keep it above a given threshold.

Constraints

The constraints of optimization problems are restrictions on the global set of possible choices to take, which make some of them unavailable. Constraints are typically used to take into account the limitedness of resources [20c]. Every element of the domain is associated with a cost in terms of arbitrary resources, and elements are feasible if their associated cost is compliant with a given budget.
This line of reasoning can be immediately applied to the case of Transiently Powered Computing: the cost of executions can be quantified by the energy they consume, and by the computational and memory effort that are necessary to complete them.

Chapter 5

Making Things Happen

5.1 Abstract

In the previous chapter we explained the main reasons why Approximate Computing Techniques (ACTs) can be effectively applied to boost the performance of Transiently Powered Computing (TPC) applications, and we have pointed out a provably convenient guideline to follow.
In particular, we have shown that one should aim at maximizing the performance of the application without consuming more resources than it is possible. The way performance and resource consumption are quantified is a function of the specific application.
A concrete case study is necessary to prove empirically that the combination is effective not only potentially but also actually. We have designed, coded and tested an application that implements the concepts that we have spoken about so far: the application consists in a classifier that recognizes the kind of activity performed by a human being, using accelerometer data. This task is called Human Activity Recognition (HAR) [Ang+12a].
We made this choice for several reasons. First of all, we have selected the ACT to apply according to the requirements of the domain of TPC. First and most important, we want the chosen technique to guarantee consistent savings when it is applied. Therefore, it must provide high savings and low overhead at runtime. Then, we are interested in techniques that have broad scope and applicability, to achieve a general and useful result. Finally, we want the technique to be easily applicable, to ease and spread the implementation of applications that make use of it. For all these reasons we have decided to go for Iterative Refinement.
Then, we have chosen a problem to address applying an algorithm that can be approximated using Iterative Refinement. At the same time, the problem must be a task that can be realistically tackled using TPC applications. Human Activity recognition meets both these requirements. This problem consists in understanding the kind of activity that is being performed by a human being by processing data that come from accelerometers that he/she is wearing. When

we move, we produce enough energy to power up small computers, as proven by Haroun et al. [HYW16], and therefore it is perfectly reasonable to think of a TPC device that uses the energy produced by the human movement to classify the kind of movement that is taking place, without having access to external power supplies.
Moreover, it is possible to approximate Support Vector Machines (SVM), a popular and powerful algorithm that has been used by Anguita et al. [Ang+12a] to perform HAR, using Iterative Refinement.
Our contribution is a study on how SVM can be interpreted as an anytime algorithm: in particular, we have modelled how the accuracy of a SVM-based classification varies according to the number of operations that are performed. We called this version of SVM Anytime SVM (ASVM).
Incidentally, the probabilistic analysis that we use to quantify the cost/performance curve of ASVM can be used to study all linear classification models. The results of this analysis are two metrics that quantify the performance and the resource consumption of the application, according to the optimization formulation that we have given in Equation 4.4.
Finally, we have coded an implementation of the application for MSP430 boards [19o], to test experimentally how the application performs on the field.
This chapter contains both the results of the probabilistic analysis, that is in appendix, and the high-level requirements of the MSP430 implementation.

5.2 Applied Technique

The primary goal of this work is to develop a proof of concept to assess the feasibility and usefulness of applying Approximate Computing to programs deployed on Transiently Powered platforms.
To reach this goal, the first decision it is necessary to take is what technique to use.
In this section, we will carry out an evaluation of various ACTs in order to choose the most apt one.
First of all, we will list the desirable properties that Approximate Computing techniques should exhibit to fit well into the domain of Transiently Powered Computing. We will use these properties to formulate evaluation criteria to quantitatively compare the available techniques.
After this preliminary analysis, we will focus on the technique which will emerge from the evaluation as best according to the aforementioned criteria, that is to say Anytime Algorithms.
The second step is to find a suitable case study where it makes sense to use an Anytime Algorithm on a Transiently Powered Computer. We suggest that an interesting application is Human Activity Recognition, i.e. the classification of the activity being carried out by human beings starting from accelerometer data.
We have implemented an application to carry out this task in an anytime fashion. In the appropriate section, we will explain the idea from an intuitive point

of view, and then we will analyze it from a formal point of view.
The analysis of the application becomes simpler when some additional hypothesis is added, and the scope of the application is restricted. To make the presentation clearer, we will initially analyze the most simplified versions of the application, and we will gradually generalize the work in the subsequent sections.
We will test and evaluate of the application on the field in the next chapter.

The goal of this section is to choose the best ACT(s) which are apt for applications in TPC.
We will perform a multicriteria analysis to make this choice in a way that is, if not logical, at least reasonable. First of all, we will enunciate the properties of an ACT that can be useful in a TPC context.
To derive these properties, we will start from the dimensions that we used to classify ACTs in Section 3.6.1. We will go through each dimension, translating it in a feature that ACTs should have to be maximally useful when applied in TPC contexts.
Then, we will quantify the relative importance of each criterion, in qualitative terms of absolute impact on the efficacy of the application of a general ACT in a TPC context.
Finally, we will give a numeric score to ACTs in each dimension, and we will sum their scores in the various dimensions weighting them by the relative importance of the dimension itself, to obtain a single synthetic number that can give an indication of the fitness of an ACT for the domain of TPC.

5.2.1 Desired Properties

In this section, we will analyze the dimensions that we used in Section 3.6.1 to provide a taxonomy of ACTs. In the taxonomy we provided, ACTs can be framed by considering some relevant dimensions: in this section, we will use these dimensions to describe what are the main properties of an ACT that can improve the performance of applications executed in a TPC context.

Hardware or Software

This dimension is used to describe the point of an application where a given ACT operates. As a matter of fact, some ACT affects physical elements of computers such as the power supply of RAM, or the utilization of approximate hardware multipliers (see Section 3.6.1 for the details).
From this consideration, we propose the following criteria to express the fitness of an ACT to a TPC context: it should not require particular hardware elements or physical modifications to the machine to work.
If an ACT can be applied on standard devices, then it can be applied in a wider range of contexts because it is subject to fewer design constraints. In conclusion, we will rank Software Techniques better than Hardware Techniques.

Saved Resource

This dimension summarizes the advantages that are created by an ACT. We gave some example of such advantage in Section 3.6.1: ACTs can involve, among others, savings in terms of energy, memory, computation or transmission time. Energy is the only resource whose availability changes dynamically during the execution of applications.
As a consequence, we are interested in techniques that allow to save energy, because these techniques are the ones that allow applications to adapt to changing environments, hence performing well in many different contexts.
Therefore, we will rank energy-saving techniques higher than the other ones.

Determinism

This dimension is used to distinguish whether different errors can be produced by different executions that consume the same input.
In general, deterministic techniques are easier to use and to debug, because faults and unexpected behaviours can be reproduced and studied. This reasoning holds in TPC contexts as well, so we will follow the guideline to give higher scores to deterministic techniques with respect to nondeterministic ones.
We think that the dimension of determinism is particularly relevant in the domain of TPC because this domain is very young and not so standardized yet. As we demonstrated in Section 2.3.8, TPC is a young discipline, still lacking standardized testbeds and established benchmarking procedures. Deterministic techniques are easier to test and debug, and so they can be easily included in benchmarks thanks to their predictable and therefore comparable behaviour.
As a consequence, the adoption of deterministic techniques does indirectly help the entire discipline of TPC in shifting towards a more mature state.

Flexibility

This dimension is used to quantify the amount of control developers have on ACTs (see Section 3.6.1). In other words, the flexibility of an ACT indicates how finely developers can tune the error that executions produce. Some techniques offer very few possibilities for the execution of a program: for example, when Using Multiple Inexact Programs (see Section 3.6.2) there are only as many possibilities as the number of versions of the program that are available. The problem of having few possibilities is that it can easily be the case that an application is unable to consume a huge amount of resources that are temporarily available because the accessible resources are just below the threshold that would unlock an expensive possibility of execution.
Let us have a look at Figure 5.1. It represents a scenario when there is an ACT with two operative modes, D1 and D2. D1 produces less precise results, and it is cheaper in terms of power consumption. The lowest horizontal line, labeled as Consumption of Decision D1, represents the power consumption of an approximate application that is operating in the conditions enforced by decision

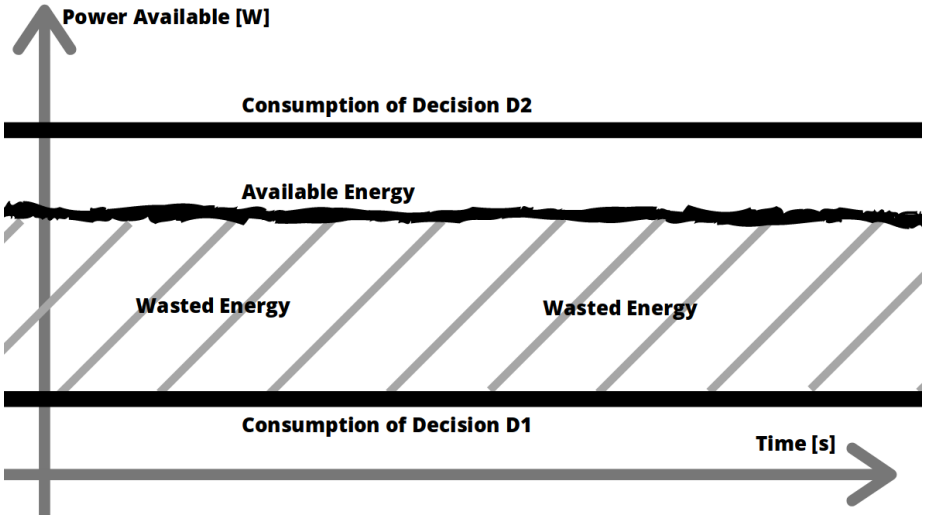


Figure 5.1: Energy wasted by techniques with low flexibility

D1. The highest horizontal line, labelled as Consumption of Decision D2, represents the power consumption of an approximate application that is operating in the conditions enforced by decision D2. In case there is a power availability between the two requirements, it is impossible to go for decision D2, so decision D1 is the only viable option. As a consequence, a given amount of power is wasted at every instant: the maximum amount of power wasted is bound by the difference between the power consumptions of D1 and D2. In conclusion, techniques with fine-grained power consumption levels waste less energy than coarse-grained techniques.
As a consequence, higher scores will be assigned to ACTs that guarantee more decisions levels.
The other aspect that is encompassed by flexibility is the minimum size of the smallest portion of code where it makes sense to apply a given ACT. If this size is small, for example, a single instruction, it is quite general and easily applicable. On the other hand, some techniques can be used on huge functions, and possibly quite specific in terms of behaviour. Therefore, we will assign higher scores to techniques that can be applied on smaller portions of code.

Visible or Invisible

Invisible Techniques can affect the performance of an application also when no instruction is being executed: for example, underpowered volatile memory does naturally deteriorate over time, losing information even if the application does not interact with it. On the other hand, Visible Techniques deteriorate performance only when certain instructions are executed: for example, approximate multipliers/adders produce errors only when they are actually used to carry out

multiplications/additions.
Formally, the (In)Visibility dimension indicates whether a given ACT may produce errors even when applied to an empty sequence of instructions. Invisible techniques are able to do so, i.e., they can produce damage at any arbitrary moment in time. On the other hand, visible techniques can create errors only when some specific instructions are executed.
Visible techniques have a more predictable effect on executions, and they can be debugged more easily than the invisible ones.
As a matter of fact, one can simply execute an application one instruction at a time to evaluate the effect of Visible Techniques: this approach would not work with Invisible Techniques, because they can create problems also during time intervals when no instruction is being executed.
Therefore, Visible ACTs are easier to debug and have an intrinsically more deterministic behaviour. We will then prefer Visible Techniques over Invisible ones.

Relative Importance of the Properties

The desired properties listed so far are not equally important: the impact they have on the fitness of an AC to the domain of TPC does greatly vary. In this paragraph we will provide a qualitative ranking of the usefulness of the properties, explaining the reasons why we consider one property more or less important. The properties will be listed in non-decreasing order of importance. We will turn this qualitative distinction into a numeric ranking in the following section, where we will concretely present our multicriteria analysis.

- Visible or Invisible: this property does not impact the usefulness or the efficacy of an application or an application. Although the utilization of visible techniques helps with the development and the testing of TPC applications, its contribution ends at a super-program level: it helps the discipline and the developers because it makes it easier to detect flaws and implementation bugs, but it does not affect the quality of applications as long as the implementation is correct.

- Determinism: Deterministic ACTs produce the same effect when applied twice in the same conditions, while Nondeterministic ACTs can produce different outcomes even starting from the same context.
  In other words, Nondeterministic techniques show some degree of independence from the input of the application: this makes it harder to test them, because test cases cannot be defined as given-input-expected-output pairs.
  Thanks to their predictability, we will rank Deterministic ACTs higher than Nondeterministic ACTs.
  Visibility and Determinism have a similar impact on the design and the deployment of Approximate, TPC applications, so they will have a similar weights.

- Hardware or Software: this property is more important than determinism because it can create serious difficulties or even block totally the deployment of a TPC application. Some techniques require hardware modifications that are so complex and specific that it is basically impossible for developers to make use of them in real-world scenarios.
  Customizing hardware is, in general, more complex than customizing software. Therefore, ACTs that make use of custom hardware can be more difficult to deploy in practice, because they require to design and produce physical components as well as software components.
  In case special hardware components are needed but unavailable or hard to produce, it can become impossible for developers to use a given ACT. So, the need for special hardware can be a substantial obstacle towards the actual utilization of ACTs, and the distinction Hardware/Software techniques will have a high weight in the analysis.

- Flexibility: the flexibility of an ACT does affect its impact on a TPC application even more than the previous ones.
  All the previous properties make it more difficult to practically implement the application of an AC in a TPC context, but the flexibility of an ACT poses questions on the very usefulness of using the technique itself.
  Therefore, the flexibility of an ACT is a proxy of how good the technique is at using the energy available. Since energy is a very important resource in TPC, flexibility will have a high weight in the analysis.

- Saved Resource: this property is by far the most important one. As a matter of fact, the main reason to apply ACTs to TPC applications is to make them more apt to their context, making them consume less in terms of the resources that are scarce in the place and time where they are deployed. In the vast majority of cases, the single most important scarce resource is energy, and in case an ACT cannot help in reducing the power consumption of a program they do not help in improving its efficacy.

5.2.2 Choice of the Technique

In Figure 5.2 it is possible to find the multicriteria analysis we have carried out taking into account the design criteria that we listed in the previous section.
It is necessary to make a preliminary consideration before commenting on the analysis itself. We had to condense in a single table the scientific results of dozens of papers, and this simple fact that we needed such an extreme synthesis makes it impossible to demonstrate the formal correctness of the procedure we followed. Moreover, we assigned a single numeric weight to the importance of the properties we identified, and to the score of each ACT along each property. It is inevitable to introduce some degree of subjectivity when assigning these scores. As a consequence of these caveats, the multicriteria analysis should not be considered as a perfect oracle whose judgement must be followed blindly.
On the contrary, this tool should be used to obtain hints and suggestions about

the decisions that one should take: when two choices have similar scores, one should reflect on the reasons why one or the other should be preferred, instead of picking the one that is slightly superior to the other.

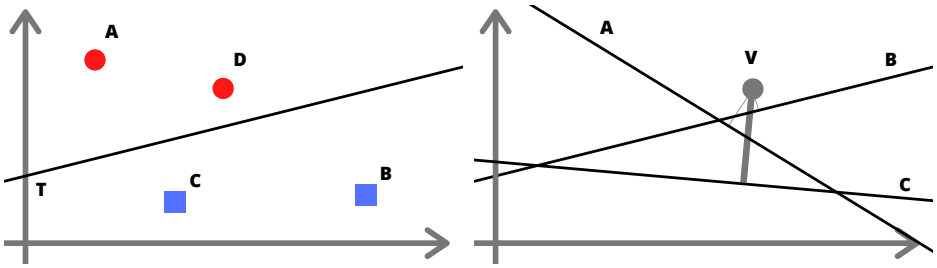
	Visible/Invisible	Determinism	Hardware/Software	Flexibility	Saved Resource	Overall
Precision Scaling	5	5	5	2	2	23.5
Code Perforation	5	5	5	4	4	30.5
Skipping Memory Accesses	5	5	1	3	3	22.2
Memoization	5	5	1	3	2	20.3
Using Multiple Inexact Programs	5	5	5	2	4	27.3
Using Inexact or Faulty HW	1	1	1	1	3	10.6
Voltage Scaling	1	1	1	4	5	19.2
Iterative Refinement	5	5	5	5	5	34

Figure 5.2: Multicriteria Analysis

Observing the overall score of the various techniques, it is possible to distinguish three main clusters: the best one is composed of techniques which score around 30, the second one contains the techniques worth around 20 and the last one is composed of a single technique far inferior to the others.
We can interpret this subdivision by inspecting the clusters to find similarities that bring together the techniques in the clusters themselves.
The best cluster is composed of two techniques (Code Perforation and Iterative Refinement) that are excellent in the most important dimensions, and the third one (Using Multiple Programs) is in general good in all the dimensions but the flexibility.
The second cluster is composed of techniques that are either less flexible or less useful in terms of resource saved than the ones in the best cluster.
Finally, the last technique is somehow the most complex to apply, and this appears in its generally low scores.
In light of these considerations, the main candidates for the proof of concept are the three members of the first cluster. We will not consider the possibility of going for Using Multiple Inexact Programs. Even though this ACT has a high score, it is extremely costly in terms of human effort and memory to keep many different versions of a program on a microcontroller.
So, the choice is between Code Perforation and Iterative Refinement: we will go for Iterative Refinement.
Iterative Refinement algorithms refine results step by step, so they can be used to improve a result as long as there is enough energy to carry out an extra step. As a matter of fact, Iterative Refinement algorithms are by definition compliant with the goal of maximizing the quality of a solution while keeping its cost below a threshold, which is the goal stated in Section 4.4.1.
Moreover, we are interested in using a technique with low runtime overhead, and anytime algorithms can be designed to have almost no runtime overhead at all. On the other hand, it is not always possible to plan a code perforation to consume exactly the resources that are available in the environment.
We will now present an anytime algorithm suitable for the domain of TPC.

5.3 Anytime Variation of SVM

In the previous section, we have decided to go for Iterative Refinement algorithms, and in this section we will describe the precise anytime application that we have implemented.
Iterative Refinement algorithms work by performing a given number of computational steps to refine a solution. Some algorithm requires that the number of steps is planned before the execution: if this policy is applied in TPC contexts, it can be the case that some extra energy becomes available during the computation and it cannot be used, because the number of steps and, consequently, the energy consumption of the algorithm have been decided in advance.
On the other hand, we are interested in algorithms where additional steps can be planned in every moment. This class of algorithm takes the name of Anytime Algorithms.
We have decided to use a known algorithm and to turn it into an Anytime algorithm. To select the algorithm, we followed one main principle: the original algorithm should solve a problem that could be addressed in a TPC context.
In particular, we selected Human Activity Recognition (HAR) as a use case, and Support Vector machines (SVM) as an algorithm. An overview of HAR and SMV is given in the next section.
SVM are a machine learning technique to perform classification: a metric to measure their performance is accuracy, a score that can range from 0 (worst) to 1 (best).
An important reason to start from SVM is that this algorithm works well in Human Activity Recognition (HAR), a task that fits well the requirements of the domain of TPC. Therefore, Approximate SVM to solve HAR are a realistic case study to demonstrate the viability of combining ACTs and TPC.
We need to quantify the accuracy of ASVM as a function of the computational effort that is invested in the execution of the algorithm: this is necessary because it is often the case that applications with low accuracy are useless, and knowing in advance a boundary on the accuracy is necessary for developers to choose whether to use ACT or not.
In other words, knowing in advance the accuracy of am ACT is kay to make the first design decision of a project, that is to say, if that specific technique fits the requirements of the project.
This analysis will be the next part of the chapter. Afterwards, we will present a concrete use case where this algorithm can be used: in particular, Human Activity Recognition (HAR). In this specific domain the combination of TPC and AC can be extremely fruitful, thanks to a series of factor we will highlight along the text.
The next sections are structured as follows: first of all, we will introduce SVM, and explain how this algorithm can be turned into an Anytime algorithm. Then, we will analyse the properties of this Anytime version of SVM, that we have baptised as Anytime Support Vector Machines (ASVM).
The section about HAR will mark the conclusion of the chapter. In the next chapter, we will provide our implementation of ASVM, that we have deployed



(a) A linear classification model

(b) An OvR classifier: vector V is classified as C, as it it is the furthest hyperplane

Figure 5.3: Linear classification models

in a TPC context. The implementation will be the concrete proof that ACT can be applied in TPC contexts.

5.3.1 Design of the Algorithm: State of the Art

In this section, we will present an anytime version of Support Vector Machines (SVM), and we will study how the accuracy of the algorithm varies as a function of the computational effort spent.
This analysis is necessary because it makes it possible for developers to know in advance the performance of applications that make use of ACTs, and this knowledge allows them to decide whether it is a good idea to adopt that specific approach, without running the risk to obtain bogus results after the application has been deployed.
Moreover, the goal of the thesis is to demonstrate that using ACTs in TPC applications it is possible to obtain reasonably good results for a fraction of their original computational cost: the analysis of the application is necessary to prove this claim in a theoretically solid way.
This section is structured as follows: first of all, we will provide a theoretical background on linear classification models. This is necessary because SVM are a particular instance of a linear model. Then we will explain the peculiarities of SVM. After this introduction, we will illustrate our anytime version of SVM, and we will study its properties.

Linear Classification Techniques

The goal in classification is to take an input vector x and to assign it to one of K discrete classes C_k where k=1,...,K [Bis06]. Classification is a task that belongs to the domain of supervised learning: this means that an algorithm must be produced based on a set of pairs (x_i, y_i), where x_i is the i-th input vector, and y_i is its class. This means that classification algorithms learn to classify points starting from a set of examples.

Typically, classification is performed by finding a function $y(\mathbf{x})$=f($\mathbf{x}$,$\mathbf{w}$) that

outputs the probability that an input $\mathbf{x}$ belongs to a class c. In case f is a linear function, the classification model will be called a linear model.
In the case of linear models, decision surfaces y=constant are linear functions of x. An example of linear model is shown in Figure 5.3a. It is possible to observe that y($\mathbf{x}$) is above a given threshold T for the points in the superior halfplane, for example A and D, and it is below the threshold for the points in the inferior halfplane, such as B and C.
In case of linear model, it is extremely cheap from a computational point of view to obtain the probability that a given input $\mathbf{x}$ belongs to a given class: one has to compute the inner product $\mathbf{x}^T\mathbf{w}$, $\mathbf{w}$ being the coefficients of the hyperplane. While classifying points (equivalently, performing inference) given a model is very simple, the real difficulty lies in the identification of the function f. In the literature, it is possible to find various approaches, such as Logistic Regression [PLI02] or Perceptron [Ros58].
Support Vector Machines (SVM) are another popular linear model. Since it is a technique that has been proven to be useful in the domain of HAR, we will give an overview of SVM in the next section.

Multiclass Classification with Linear Models

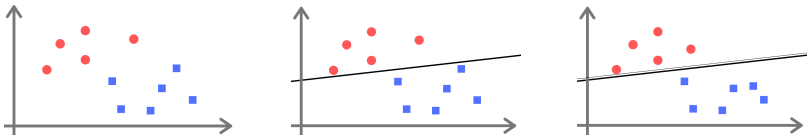
Linear models can be used to classify vectors also in the situation when multiple classes are available. There are two main methods to carry out this task: One-versus-Rest (OvR) [20k] and One-versus-One (OvO) [20l].
OvR classification consists in using one hyperplane for each class, to distinguish among samples that belong to the class and the ones that do not belong to the class.
New samples are classified as belonging to the class from whose hyperplane they are the furthest away. This situation is drawn in Figure 5.3b: the distance from vector V to hyperplane C is greater than the distance from A and B, so V is classified as a member of class C.
The alternative approach is to use one classifier for each pair of classes (C_i, C_j): when a new sample must be classified, every hyperplane C_i, C_j is used to check whether a point belongs more to class C_i or to class C_j, and the class C_m that won the greatest number of direct confrontations is the class the point belongs to.
The former approach is generally preferred thanks to its greater scalability, as it requires O(number of classes) hyperplanes instead of O(number of classes^2). As a matter of fact, OVR classification needs only one hyperplane for each class, while OVO classification needs one hyperplane for each pair of classes. As a consequence, classification is slower in OVO and OVO models require more memory to be stored.
Finally, OVR has higher interpretability. Let us imagine a situation when the hyperplane used to distinguish between class i and the other elements is

$$\boldsymbol{w_i} = [w_{i1}, w_{i2}...w_{in}]$$

in case the k-th coefficient has a greater absolute value than the l-th coefficient, the k-th feature of a sample will have a greater impact on the classification than the l-th, and so we can understand what are the features that characterize every single class (See Figure5.3b).

Support Vector Machines

SVM are a classification technique, invented by Vapnik et al. [BGV92]: this means that a SMV model is created starting from a set of pairs (x_i, y_i), x_i being the i-th input vector, and y_i being the class if the i-th input vector. For the sake of simplicity, we will restrict our example to the case of binary classification, i.e., when only two classes C_1 and C_2 are present.
An example is shown in Figure 5.4a: each element represents a 2D vector, and the shape of the element represents the class it belongs to. Let red circles be class C_1, and blue squares be class C_2.



(a) A set of labelled input vectors

(b) A linearly separable dataset

(c) A hyperplane with a small margin

(d) A dataset that is not linearly separable

(e) A hyperplane with a wide margin

(f) Highlighted points are the support vectors

Figure 5.4: Linear separability

SVM rely on the assumption that the dataset is linearly separable: i.e., it must exist at least one hyperplane H such that all the points in C_1 are on one side of H, and all the points in C_2 are on the other side. The dataset in Figure 5.4b is linearly separable, while the one in Figure 5.4d is not. The idea of SVM is to identify a hyperplane that gives the best distinction between the two classes: in particular, the one which is maximally distant from the closest input vector.
This means that the hyperplane will have the maximum possible margin around it, where the margin is a zone where no input points are present. Figure 5.4c shows one situation when the margin is very small, and Figure 5.4e shows a situation when the separating hyperplane has a wide margin. The hyperplane with maximum margin can be obtained by solving a Quadratic Programming

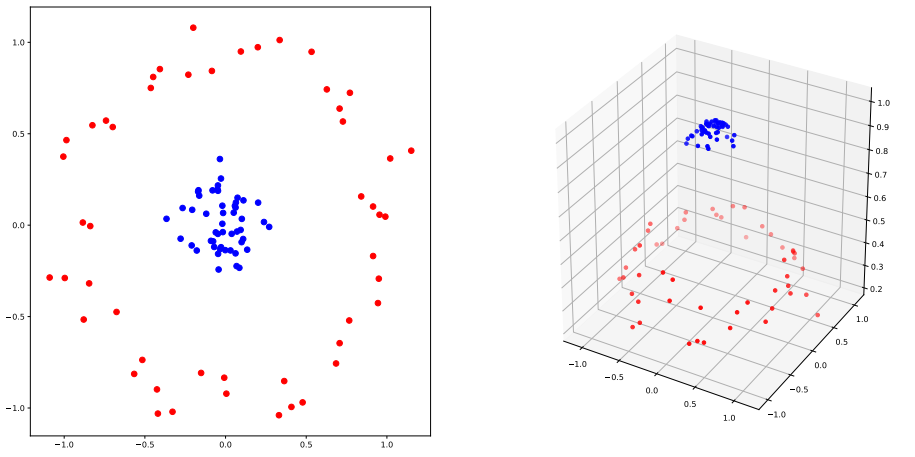


Figure 5.5: Projection in high dimensionality spaces eases linear separability

problem [20h], so it is computationally feasible to train this algorithm: in particular, the computational complexity of training is $O(n^3)$, where n is the size of the training set. The output of the training phase is the separating hyperplane itself. In practice, it is a tuple of coefficients with the same dimensionality of the inputs, and so it is very cheap to store.
The separating hyperplane with maximum margin can be obtained by considering a subset of the input vectors in the training set, that are called "support vectors": this situation is shown in Figure 5.4f. As exemplified in Figure 5.4d, some datasets are not linearly separable, so it is necessary to project them in spaces where this property holds to use SVM.
A popular way to achieve this result is to project points in spaces with high dimensionality, by means of mapping functions called kernels.
Figure 5.5 shows the effect of such a projection on a dataset that is not linearly separable in the original space, and becomes linearly separable in the projection space. The classification of a point P is $sgn \sum K(P)K(S_i)$, where S_i are the support vectors and K is the kernel.
This technique makes it more expensive to classify new points: instead of a single inner product, one must compute as many inner products as the number of support vectors. Moreover, it is necessary to store all the coordinates of the support vectors to use the model.
Several techniques have been developed to mitigate this problem. For example, Zhang et al. [Lan+19] developed an approximated version of SVM, based on the idea of computing an explicit, approximated version of the PSD matrix that identifies the kernel.
Hsieh et al. [HSD13] created a divide-and-conquer solver to train the model in a more efficient way. In the literature it is possible to find iterative refinement algorithms to perform inference on SVM models. We will delve deep into this topic in the next section.

Iterative Refinement in SVM

Iterative Refinement algorithms have interesting properties both in TPC and in traditional computing scenarios. We have discussed the fact that Iterative Refinement algorithms look promising in the domain of TPC in Section 5.2.
As for traditional continuously powered computing, iterative refinement can be used to stop a computation in the very moment a given precision threshold is met, possibly saving a huge amount of resources.
Due to this interesting property, it is no surprise that the idea of applying Iterative Refinement to SVM has been successfully pursued in the past. In particular, two different algorithms have been developed. Decoste [Dec02] developed Interval Valued Support Vector Machines. The fact that a sample point $\mathbf{x}$ belongs to a class C_1 or to a class C_2 can be computed by checking the sign of the quantity

$$Q = \sum_{p \in C_1} d_{xp}^2 - \sum_{q \in C_2} d_{xq}^2$$

where d_{xp}^2 is the squared distance between the vectors $\mathbf{x}$ and $\mathbf{p}$.
Instead of computing directly Q, Decoste proposed an algorithm to compute bounds on Q, that are continuously narrowed until the sign of Q is determined.
Another interesting algorithm has been invented by Wagstaff et al. [Wag+09]. The principle of the work is to train two different models, one extremely accurate but expensive to use and the other less precise but extremely fast, as it is based on a subset of the support vectors.
The idea of the algorithm is to classify all the available samples using the fastest model and to assess the probability that the classification is wrong. Then, the points for which the uncertainty is higher are analyzed again by using the precise model.
This algorithm implements Iterative Refinement at the granularity of the dataset: the average accuracy gradually grows while more and more computational effort is invested.
Both algorithms are very powerful, but they have some drawback that could make it problematic to apply them in a TPC context. Interval-Valued SVM are, in the worst case, slower than the traditional version of SVM: this is a serious issue in the domain of TPC, where it is necessary to optimize the consumption of even the smallest crumb of resources.
As for Wagstaff's algorithm, the main problem is the fact that this technique does exhibit a low tunability. As a matter of fact, there are only two possibilities for tuning the precision for every single sample: one can either use the approximate model or the accurate one.
In case a high amount of samples is immediately available this is not a huge trouble. As a matter of fact, one may find a good blend of the approximate and the accurate model to consume all and only the resources that are available. However, this is not the case for TPC applications: the hardware of TPC devices is very limited, and their memory is no exception. As a consequence, it is basically impossible for a TPC device to have access to a huge amount of samples to classify at the same time.

The goal of this work is to develop a novel application able to overcome the limitations of the existing solutions, and in particular fit for the domain of TPC.

5.4 ASVM: Description and Properties

In this section we will present an anytime algorithm, ASVM, that can be applied to all the linear models for classification, and that does overcome the limitations of the existing iterative algorithms for SVM, that is to say, low performance in the worst case for Decoste's algorithm, and low flexibility for Wagstaff's.
In the rest of the thesis, we will apply ASVM to implement a HAR application, to demonstrate that it is feasible and useful to combine ACTs and TPC. We developed this variation of SVM to implement an application that uses an ACT to obtain better performance than an exact algorithm in a TPC context.
So far, we have shown why we decided to use iterative refinement, mainly for its low runtime overhead, and why we have decided to study the case of Human Activity Recognition, that is to say because it is a scenario that fulfils the requirements of TPC applications.
We will present some variations of the application, starting from the ones that have the narrowest scope, i.e. that require the strongest assumptions in order to work, and we will move towards the most general ones.
This order has the advantage of showing the easiest things first, so as to facilitate the explanation of the most complex models.
Let us consider a set $\{\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{x_3}...\boldsymbol{x_m}\}$ of samples to classify, where $\boldsymbol{x_i}$ is a t-dimensional vector $\boldsymbol{x_i} = [c_1, c_2 c_3 ... c_t]$.
Each vector $\boldsymbol{x_i}$ can belong to one of NC classes $\{C_1, C_2 ... C_{NC}\}$.
We will initially analyze the case when NC is equal to 2, and all the distribution of the coordinates c_i of the input vectors are independent from one another.
Then, we will cover the case when NC is greater than 2, but the distribution of the coefficients remain mutually independent.
Finally, we will drop also this assumption, and we will cover the case when the coordinates of the input vectors may be correlated, and when multiple different classes exist.

5.4.1 Description of ASVM

SVM is a linear classification technique: it is based on a set of hyperplanes that split the space of the features in distinct regions. Points are classified according to the region they belong to.
Let $\boldsymbol{w_1}, \boldsymbol{w_3}, \boldsymbol{w_3}...\boldsymbol{w_c}$ the hyperplanes corresponding to c classes. So, a window sample $\boldsymbol{x}$ can be classified by computing the inner products $\boldsymbol{w_1 x}, \boldsymbol{w_2 x}, \boldsymbol{w_3 x}...\boldsymbol{w_c x}$ and taking the class with the biggest inner product. Our idea is to approximate

$$\boldsymbol{w_a x} = \sum_{i=1}^{n} w_{ai} x_i \approx \sum_{i=1}^{p} w_{ai} x_i$$

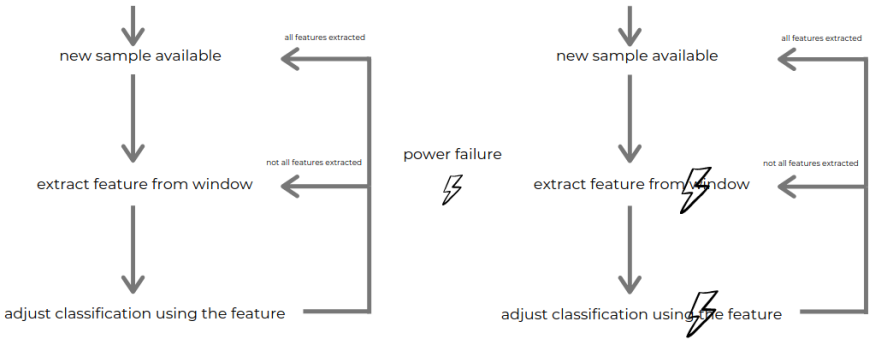


Figure 5.6: Inference in ASVM

with p≤n. In other words, we will use only a part of the features to compute the classification. The way classification is carried out in ASVM is shown in Figure 6.1. The idea of the anytime application is to compute the features one by one and to use the features to adjust the classification as soon as they are available. This kind of approach has the advantage that it becomes easy to cope with power failures: as a matter of fact, whenever a power failure happens there is already a partial result is available. Moreover, the partial result has made use of the highest possible amount of features that could be computed with the power available so far. This technique allows to arbitrarily cut down the costs of classification: however, since less information is included in the classification, the result will be less precise. In the following sections, we will estimate the precision loss due to this approximation.

5.4.2 Binary Classification, Independent Coefficients

Let $\{\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{x_3}...\boldsymbol{x_m}\}$ be a dataset composed by m vectors.
Let each sample

$$\boldsymbol{x_i} = [c_1 i, c_2 i, c_3 i...c_t i]$$

be a t-dimensional vector.
Let us assume that each coordinate c_i is drawn from a Gaussian distribution C_i

$$C_i \sim N(0, \sigma_i^2)$$

Let us assume that all the C_i are mutually independent.
Let

$$\boldsymbol{w} = [k_1, k_2, k_3...k_t]$$

be the hyperplane that is used by the linear model to classify the samples: for the sake of simplicity, let us assume that

$$k_j = 1 \; \forall j \in \{1..t\}$$

We will demonstrate that this assumption does not affect the generality of the analysis.
In linear models, the classification of a sample $\boldsymbol{x_i}$ is determined by the quantity

$$S_i = \boldsymbol{w}^T \boldsymbol{x_i}$$

where $\boldsymbol{w}^T \boldsymbol{x_i}$ is a normal inner product. Let us indicate as c_{ji} the j-th coefficient of the i-th sample which can be written as

$$S_i = \sum_{j \in \{1 \dots t\}} c_{ji} w_j$$

Let

$$S_{pi} = \sum_{j \in \{1 \dots p\}} c_{ji} w_j$$

when p=t, $S_{pi} = S_i$. In the other cases, S_{pi} is cheaper to compute than S_i. This consideration is particularly interesting when the features c_{ji} of point i are not immediately available, and involve extra computations to be extracted. We will come back to this point in the next chapter.
An approximate version of the classification of point P can be computed as

$$class_{pi} = sgn(\sum_{j \in \{1 \dots p\}} c_{ji} w_j)$$

that is to say using only the first p coordinates of the vector. We will call $class_{pi}$ the p-partial classification of the i-th vector. We are interested in computing the probability that a p-partial classification is coherent with its corresponding t-partial classification, i.e., with the exact classification, as a function of p.

$$P(p - coherence) = P(class_{pi} = class_i)$$

To compute $P(p - coherence)$, we will introduce an auxiliary variable, to easily represent the influence of the coordinates that are excluded from the p-partial classification.

$$R_{pi} = \sum_{j \in \{p+1 \dots t\}} c_{ji} w_j$$

p-coherence is enforced if the following event takes place

$$S_{pi} \geq -R_{pi}$$

After a few steps, listed in Chapter 9, we obtain

$$P(p - coherence) = 2 \int_{d=0}^{+\infty} f_{S_{pi}}(d)(1 - F_{R_{pi}}(d))dd$$

The distribution of $f_{S_{pi}}$ and $F_{R_{pi}}$ is computed in Chapter 9.

5.4.3 Multiclass Classification, Independent Coefficients

We will analyze the case of multiclass classification using the technique of One-versus-Rest (see Section 5.3.1). We recall that using this technique one can classify a vector $\mathbf{x}$ by measuring its signed distance from each of the hyperplances that correspond to each class, and assign it to the class whose hyperplane it is the furthest away.
Formally, let

$$\{C_1..C_c\}$$

be the classes and

$$H_l = [h_{l1}, h_{l2}, h_{l3}...h_{lt}]$$

the hyperplane corresponding to class l. To simplify the notation with respect to the previous case, let

$$\boldsymbol{x} = [c_1, c_2...c_t]$$

a sample, whose coordinates

$$c_i \sim N(0, \mu_i)$$

We are interested in the probability that a partial classification, performed using the first p coordinates, is coherent with the full classification performed using all the t coordinates.
Let C_p the class to which the point to classify belongs according to the evaluation done with the first p coordinates. Let C_f the final classification of the vector, performed using all the t coordinates.
In conclusion, we have that

$$P(C_p = C_i \wedge C_f = C_i) = \int_{\boldsymbol{adv}>\mathbf{0}} P(Ad_{ip} = \boldsymbol{adv}) P(Ac_{ep} >= -\boldsymbol{adv}) d\boldsymbol{adv} \tag{5.1}$$

Where Ad_{ip} and Ac_{ep} are random variables whose distribution is computed in Chapter 9, and varies as a function of p. This integral can be computed numerically.

5.4.4 Binary Classification, Correlated Coefficients

We will not analyze this case in detail: the analysis of this variant is so similar to the one in the next Section 9.2.1 that it would not add any theoretical value nor ease the introduction of the most general case.
Although this case is not so interesting from a theoretical point of view, it is a nice opportunity to visualize the practical effects of the equations derived so far. In particular, Figure 5.7 shows the probability of obtaining a coherent classification as a function of the number of features that are considered, for a random dataset. This probability has been computed using the model illustrated in the next Section 9.2.1. It is possible to notice the following aspects:

1. When zero features are used, the probability to obtain a coherent classification in 0.5. This makes sense because it is impossible to predict the

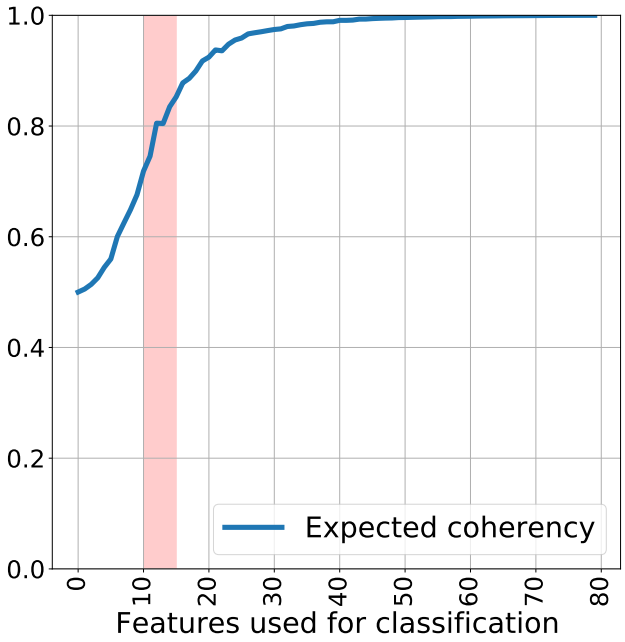


Figure 5.7: Expected coherency of binary classification of points with correlated features. An oscillation is highlighted.

outcome of a classification without any information better than a random guess.

2. The accuracy reaches one when more and more features are considered. At the end of the process, it becomes more difficult for the few remaining features that have not been considered yet to overcome the contribution of a high number of features.

3. The curve is increasing but for small random oscillations. When more and more features are considered, it becomes less likely for the remaining features to have an impact high enough to invert the sign of the partial classification. We will explain the random oscillations in Section 7.2.2.

5.4.5 Multiclass Classification, Correlated Coefficients

The last and most general case we are going to analyze is the situation when the coordinates of the samples to classify are correlated, and more than two classes are present.

The scheme of the demonstration is very similar to the one of the previous case, that is to say Section 9.2.
First of all, we are going to model the probability distribution of the signed distance of a sample $\boldsymbol{x}$ from the separating hyperplanes. Then, we will compute the probability distribution of the differences between the partial distances.
Finally, we will use these last probability distributions to model the probability that a classification remains coherent when more coordinates are used. Let $\boldsymbol{x}$ be the vector to classify.

$$\boldsymbol{x} = [c_1, c_2 ... c_t]$$

Let us assume that

$$\boldsymbol{x} \sim N(\boldsymbol{\mu_x}, \Sigma_x)$$

Where

$$\boldsymbol{\mu_x} = \mathbf{0}_t$$

And the entry of Σ_x on the i-th row and j-th column is indicated with Σ_x^{ij}.
Let

$$\{C_1, C_2 .. C_c\}$$

be the classes among which vectors can be classified. We need to compute the probability that a partial classification remains coherent when all the coordinates are used. Let

$$PC_p$$

be the partial classification computed using the first p coordinates. A partial classification is coherent if $PC_p = PC_t$. The probability of this event can be computed: the steps are explained in Chapter 9.

$$P(PC_p = PC_t) = \sum_{C_i \in \{C_1 .. C_c\}} \int_{\boldsymbol{v_e} > \mathbf{0_c}} \int_{\boldsymbol{v_c} > -\boldsymbol{v_e}} pdf_{Vi}([v_e|v_c]) dv_e dv_c$$

Where pdf_{Vi} is a probability density function, which is known given the number p of features that are being used.

5.5 Use Case: Human Activity Recognition

Once we have decided that we want to apply ASVM, and we have studied their properties, we need to find a reasonable use case to apply this algorithm. In particular, we need to find a scenario with the following characteristics:

1. The problem can be solved using low-end devices such as microcontrollers: this is necessary because we want to build a TPC application, and TPC applications cannot afford powerful, power-hungry devices

2. The scenario must offer the possibility to harvest energy from the environment: this is a basic assumption to apply TPC techniques

3. The problem in the scenario can be solved using SVM: this is necessary because we have decided to use ASVM for our use case.

In this section, we will prove that Human Activity Recognition (HAR) meets all these requirements. Human Activity Recognition is the ability to recognize a human's current activity based on information from various sensors such as physiological sensors, cameras, and RFID sensors [YLC11].
As for property 1, several works, such as Anguita's influential 2012 paper [Ang+12b], have demonstrated that it is possible to perform HAR on low-end devices. As for property 2, different scholars have successfully harvested energy from human movement using wearable kinetic harvesters. For example, Haroun et al. [HYW16] managed to extract 445 uW by using micro-electromagnetic vibration energy harvesters, while Romero et al. [RNW09] obtained a similar result, extracting 117 uW from a 2 cm^2 device worn by walking volunteers. Therefore, it is possible to apply energy harvesting techniques in HAR applications. Property 3 can be easily verified by considering that SVM perform well on HAR, as demonstrated by Anguita et al. [Ang+12b]. The work consist in classifying samples from six different activities, namely Walking, Walking Upstairs, Walking Downstairs, Standing, Sitting, Laying. Anguita's algorithm obtained an average precision of 0.909, while the average recall was 0.897. In conclusion, HAR is a case study that meets all the requirements we need for our proof of concept, and SVM are an algorithm that performs well for this problem. We have created an Anytime version of SVM: we will show its functioning and its properties in the next section, along with a brief overview of the existing Anytime solutions for SVM.

5.5.1 Data Processing

The raw input of HAR applications is typically constituted by a 6-dimensional trace of acceleration/angular velocity data.
These raw data have to be processed before they can be classified, and this processing phase constitutes the most relevant part of SVM-based HAR applications in terms of computational effort.
In this section we will describe how data have to be processed to obtain relevant features: we need to introduce the processing phase for two reasons:

1. The cost of the processing phase in terms of clock cycles is more than 90% of the total execution cost. In light of this consideration, we will conclude that our proposal to use only the affordable features and neglecting the others without even computing them has a major impact on the cost of executions.

2. The way the processing phase takes place made it necessary to apply some trick to the C code we have developed: the biggest practical consequence is that it made it necessary to use fixed point numbers. The fact that we use fixed point numbers made it necessary to check the correctness of the implementation and the precision loss during the evaluation of our solution.

In this section, we will describe how raw data have to be processed to extract the features, using the procedure introduced by Anguita et al. [Ang+12b].

Dataset Description

HAR consists in processing arbitrary data gathered from sensors that track the state of a person to understand what activity the person is performing.
We will focus our attention on data coming from accelerometer data.
Let us retrace the reasons why accelerometer data are a promising raw material to process using a combination of TPC and ACTs. First of all, the fact that an acceleration is being measured implies that something is moving, and so that it is possible to extract mechanical energy from his/her movement, using the energy harvesting techniques that we have listed in Section 2.5.1.
Moreover, Anguita et al. [Ang+12a] used SVM to process accelerometer data, with the objective of performing HAR. The original dataset that we have used has been gathered by Anguita et al. [Ang+13]. The entries of the dataset are composed by some data sensor, that we will describe in detail later, and a single label for each entry that describes the activity that was being carried out when the corresponding data were measured.
This dataset has been obtained using the sensors included in Samsung Galaxy S II smartphones. A group of 30 people aged from 19 to 48 were ordered to carry out specific activities, while a smartphone was monitoring them and gathering data.
In particular, the volunteers were asked to Stand up, Sit, Lay down, Walk, Walk upstairs and Walk downstairs. Meanwhile, an inertial accelerometer did measure six different physical quantities, i.e., the linear acceleration in the three dimensions and the angular velocity in the three dimensions.
More than 99% of the energy of the signal obtained by the accelerometer is below 20 Hz. Therefore, a good sampling rate should be more than twice this frequency.
As a consequence, Anguita's team chose a sampling frequency of 50 Hz, and used a 3^{rd} order low-pass Butterworth filter with a cutoff frequency of 20 Hz to remove the noise.
The data measured by the accelerometers contain a component due to the acceleration of the body and a component due to gravity acceleration. To distinguish between these components, in the original work it is assumed that the components related to gravity are characterized by lower frequencies than the ones due to body acceleration. Therefore, they used a second lowpass filter to divide these components.
Single accelerometer samples are very difficult to interpret, and so the scholars grouped them in sliding windows to obtain better insights from the data in longer time intervals. The length of the time interval has been chosen by taking into account several factors, such as the average walking cadence of healthy, elderly or disabled people, and constraints related to the domain of signal processing. In particular, data need to represent a full walking cycle to be more representative, and Anguita et al. proved that it is useful to transform them in

the domain of frequency using FFT.
FFT can process data in groups with a cardinality that is a power of 2, so Anguita and his colleagues chose the shortest interval that contained enough samples to cover a full walking cycle, and composed by a number of samples that is a power of 2: in particular, 128 samples.

Feature Extraction

For the reasons explained above in Section 5.5.1, the raw data measured by the accelerometer have been grouped in sliding windows composed by 128 samples each.
These raw data have been enriched by extracting various features from the signal, to gain better insights about its properties hence making HAR easier.
In particular, the authors extracted 17 measures from the signals in the domain of time and in the domain of frequency.
Anguita's team started from the work by Jhun-Ying et al. [YWC08], that designed an algorithm based on artificial neural networks to perform HAR. Jhun-Ying et al. used a set of basic metrics such as mean, standard deviation, minimum and maximum. Moreover, they computed Signal Magnitude Area and Average Energy of accelerometer signals in the domain of time or in the domain of frequency, and fed the result to a neural classifier.
Anguita et al. did further develop the idea by Jhun-Ying et al., computing other features of the signal. In particular, their contribution consisted in adding the energy distribution of the signal in the domain of frequency, the skewness of the signal and the angles between its components.
The final result of the signal processing was a 561-dimensional dataset, that is publicly available [20u].

5.6 Classification of Proof of Concept

In this section, we will classify the application that we are proposing using the criteria that we have shown in the previous parts of the work. In particular, we will discuss what are the challenges that we are tackling in TPC and AC, we will classify our solution according to the taxonomies for AC and TPC applications, and we will describe how we implemented the optimization principles to combine AC and TPC.

5.6.1 Addressed TPC Challenges

In Section 2.3 we have listed the most prominent challenges in TPC: our solution addresses a subset of the aforementioned problems. In particular:

1. Constrained resource: our application allows to use in a more efficient way the energy that is available.

2. Uncertain environment: our application is built to deal by design with the uncertainty of the environment, as it adapts its behaviour to energy availability.

3. Timely execution: the application is built to provide timely results. As power failures trigger the emission of outputs, it is impossible that a computation remains pending during a shutdown.

5.6.2 Classification as TPC solution

In Section 2.6, we have presented a simple taxonomy to classify TPC solutions. We will now apply that taxonomy to ASVM. Dimension by dimension, we have:

1. Demanding: implementing our solution implied a serious reasoning to find a clever way to mitigate the effect of power failures, that had a heavy impact on the code.

2. Adaptive: our application changes its behaviour according to the amount of energy that is available, and deals with the problem of emitting timely results without explicitly mentioning the problem of time.

5.6.3 Classification as ACT

In Section 3.6, we have presented a taxonomy to classify ACTs, that we will use now to classify our application.

1. Hardware or Software: our solution is a pure software technique, that does not require any kind of hardware customization.

2. Saved Resource: our application allows reduce the amount of energy consumed by a program.

3. Determinism: our solution is deterministic.

4. Flexibility: the minimum level of granularity at which ASVM work is the entire program.

5. Visible or Invisible: our solution is visible, because all the modifications that are applied to the normal workflow consist in instruction that are skipped. In other words, modifications happen by definition during instructions.

5.6.4 Control of ACT

In Section 3.6.3 we have presented a taxonomy for the type of control it is possible to perform on ACTs. Our application can be classified using that taxonomy:

1. Online, Offline: our technique works online. The behaviour of the application is not scheduled in advance but is determined by runtime conditions.

2. Manual, Automatic: our application is manual. We designed and coded it so support a specific ACT.

3. Degree of Approximation: our technique supports as many degrees of approximation as the number of features of te data. In this specific case, we are speaking of several hundreds of features.

5.7 Conclusion

In this chapter, we have described all the ingredients to implement an application that uses ACT in TPC context. In particular, we have

1. The ACT we want to use: Iterative Refinement, mainly because it has a very low runtime overhead.

2. The application to which we want to apply Iterative Refinement: the application is a novel version of SVM.

3. An estimate of the performance of the application as a function of the computational effort spent. This is very important because it allows us to estimate the performance of the application in a general way, independent from the randomness of the experiments carried out on specific datasets. In other words, it supports the claim that our technique is general and that it is not by chance that it produces the results we will see in the Evalutation part.

4. A concrete use case to apply the application: in particular, Human Activity Recognition.

In the next chapter, we will illustrate the C implementation we have developed to make these concepts concrete, and to measure on the field their behaviour.

Chapter 6

Prototyping

6.1 Abstract

In the previous Chapter 5, we have described the functional requirements of the Approximate Intermittent application we developed to demonstrate that it is fruitful to combine the domains of TPC and AC.
Our final goal is to prove on the field that our idea produces better results than a state-of-the-art solution. To do so, we need a functioning implementation of the concept we are proposing: the goal of this chapter is to describe how we carried out such implementation and how we tested its correctness. The deployment of our application in a realistic scenario works this way: first of all, it is necessary to train a model to classify accelerometer data. The training phase can take place on arbitrary machines.
Then, it is necessary to put the firmware on an embedded device, physically connected to the accelerometer that measures the data.
The need to use an embedded device to execute the application does seriously impact the details of the implementation: in particular, we had to represent numbers using fixed point to speed up the computation (>10x speedup with respect to floating point representation), and we had to use a simple linear model without kernel projection to reduce the memory consumption of the algorithm.
As we have highlighted in Section 5.5.1, raw accelerometer data have to be processed before being fed to the classifier. Also, this processing phase must take place on the embedded device because it is the only computer that can access the accelerometer.
After having processed raw data, the microcontroller can classify them using the classifier trained beforehand.
The training phase is not particularly interesting for the scope of this work, because we did not provide any contribution in this sense, as we have trained a model using a standard SVM Python library from the "scipy" package[20m].
Therefore, we will focus on the way we implemented and tested the firmware. In this section, we will describe the main aspects of the implementation: the

requirements in terms of memory and computational power, the commented pseudocode, and how we tested the firmware, that is to say comparing it to a Python program to perform the same task.
We compared the intermediate variables in the Python program and in the C program and we proved that the Mean Absolute Percentage Error (MAPE) between the corresponding variables is always less than 15%, and less than 1% for all the intermediate variables that impact the classification. Moreover, we have demonstrated that the approximation is Gaussian and its expected value is 0 for 3 classes of variables out of 5.
This chapter is organized this way: first of all, we will list the nonfunctional requirements of the implementation (Section 6.2), then we will display the details of the implementation (Section 6.3), and finally we will show how we tested the correctness of the implementation (Section6.4).

6.2 Nonfunctional Requirements

Figure 6.1 shows a high-level description of ASVM. We will now delve deep in the features that the algorithm must exhibit to correctly adhere to the behaviour that is shown.

- Memory consumption: the application must be deployed in a TPC context. As explained in Section 2.3.1, TPC devices are very poor performances in terms of computational power and memory availability. In this scenario, memory is the most problematic limit.
  Machine learning models, in general, can have remarkable sizes, and if the size of the model is bigger than the amount of storage that is available in the target device there is no way to implement it.
  SVM-based models can have huge sizes in case the classifier makes use of the kernel trick (see Section 5.3.1). In this case, the model is composed of one set of support vectors for each class. Therefore, the memory consumption is

  $$O(n_c n_{svc} n_d)$$

  Where n_c is the number of classes, n_{svc} is the number of support vectors that hold the hyperplane of class c, and n_d is the number of dimensions of the samples.
  In the present case, n_d is over 500, so the model could easily be composed of thousands of parameters. This is a problem because the novolatile memory available in microcontrollers that are typically used for TPC applications is in the order of kiloBytes [19o]. Therefore, it is a primary concern that one must design mechanisms for ASVM classification models to be compact.

- Computational cost: another important aspect that must be considered is the time needed to carry out the classification. The classification of samples cannot take an excessively long time, for two main reasons. First

of all, accelerometer data do continuously arrive in the microcontroller to be processed. As a consequence, the processing of a sample cannot take longer than the interval between the arrival of two consecutive samples. Moreover, every millisecond that passes when the microcontroller is turned on has a cost in terms of energy, that is the single scarce resource that must be saved during the operative life of the application.
In conclusion, there are both a hard constraint and a soft constraint on the maximum amount of time that can be spent by the algorithm to complete the computations triggered by the arrival of a new sample. This time cannot be longer than the inverse of the sampling rate, and should, in general, be maintained as low as possible to save energy.

6.3 Implementation

The goal of the work is to deploy a Human Activity Recognition (HAR) application in a TPC context, overcoming the limitations and the difficulties of TPC. Therefore, the application must be extremely light in terms of memory consumption and computational power needs, otherwise it may not fit the specifications of devices normally used in TPC.
This hard requirement concerns only the software that will be deployed on TPC devices: other preliminary procedures do not have any kind of constraint.
Starting from this consideration, we have split the algorithm into two parts: the training of the classification model and the inference.
The training phase can be carried out using arbitrary resources and machines, to it is not particularly problematic. On the other hand, the inference phase will take place on TPC devices, so it must fit their memory and it must satisfy other requirements in terms of timeliness, that we will analyze in the rest of the chapter.
This section is structured as follows: first of all, we will provide a high-level view of the algorithm, in the form of commented pseudocode. Then, we will show the most critical implementation choices that we have made to cope with the context where the application must be deployed.
Finally, we will delve deep in the sections of the algorithm, highlighting the ones that are the most problematic in terms of requirements and technical challenges, and we will present the solutions for their specific problems. Our goal is to deploy a HAR application on a TPC device. In particular, the application is an implementation of a novel algorithm, Anytime Support Vector Machines (ASVM), that is an anytime algorithm based on Support Vector Machines.
This algorithm needs a preliminary training phase to work properly: the goal of this phase is to find the optimal values for a set of parameters that will be used in the inference phase.
The training phase of ASVM is not different from the training of a normal SVM model: for further technical details about it, see Section 5.3.1. On the other hand, the inference phase is dissimilar from the inference on traditional SVM. We will analyzed the theoretical properties of the inference we are proposing in

Chapter 9.
The way classification is carried out in ASVM is shown in Figure 6.1. The idea of the algorithm is to compute the features one by one and to use the features to adjust the classification as soon as they are available. This kind of approach has the advantage that it becomes easy to cope with power failures: as a matter of fact, whenever a power failure happens there is already a partial result is available. Moreover, the partial result has made use of the highest possible amount of features that could be computed with the power available so far.

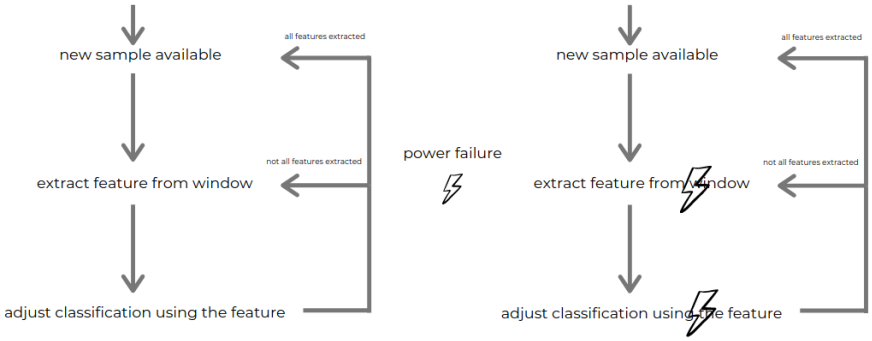


Figure 6.1: Inference in ASVM

6.3.1 Pseudocode

In this section we will show the pseudocode of the algorithm (Algorithm 1), to explain in further details how it works.
First of all, the device waits for the accelerometer to produce enough samples to fill an entire window, let ut say M.
Each raw sample contains N measures, that are the acceleration and optionally the angular velocities along the three axes.
The algorithm maintains a data structure to store NM numbers, that is to say the last M samples it received.
We will refer to this data structure as a window. When the first window has been filled, the very classification begins.
The algorithm iterates over the N dimensions of the raw data, and for each dimension it does extract a number of features. Each time a feature is extracted, it is immediately used to update the partial classification that is being computed. Let us have a look at the main functionalities of the algorithm. In rows 1 to 9, we declare the variables that will be used in the rest of the code.
The while loop from line 10 to line 13 is necessary to pile up enough samples when the algorithm starts: as a matter of fact, it is necessary to have at least one full window to extract the features from the raw samples measured during

Algorithm 1 Human Activity Recognition with ASVM

```
1: numberOfSamplesReceived ← 0
2: windowWithSamplesToProcess ← initializeWindowOfSamples()
3: currentWindow ← NULL
4: currentBodyWindow ← NULL
5: feature ← NULL
6: classification ← NULL
7: gradientOfBodyA ← NULL
8: bodyMagnitude ← NULL
9: gradientMagnitude ← NULL
10: while numberOfSamplesReceived < SIZE_OF_WINDOW do
11:   numberOfSamplesReceived ← numberOfSamplesReceived + 1
12:   receiveAndEnqueueSample(windowWithSamplesToProcess)
13: end while
14: while true do
15:   while ¬ newSampleAvailable() do
16:     wait()
17:   end while
18:   receiveAndEnqueueSample(windowWithSamplesToProcess)
19:   for all dimension in DimensionsOfRawSample do
20:     currentWindow ← windowWithSamplesToProcess[dimension]
21:     currentWindow ← applyMedianFilter(currentWindow)
22:     currentWindow ← applyLowPassFilter(currentWindow)
23:     currentBodyWindow ←separateBodyFromGravity(currentWindow)
24:     for all procedureToExtractFeature do
25:       feature ←procedureToExtractFeature(currentBodyWindow)
26:       classification ← updateClassification(classification, feature)
27:     end for
28:     gradientOfBodyA ←deriveWindow(currentBodyWindow)
29:     for all procedureToExtractFeature do
30:       feature ←procedureToExtractFeature(gradientOfBodyA)
31:       classification ← updateClassification(classification, feature)
32:     end for
33:   end for
34:   bodyMagnitude ←computeMagnitude(currentBodyWindow)
35:   for all procedureToExtractFeature do
36:     feature ←procedureToExtractFeature(bodyMagnitude)
37:     classification ← updateClassification(classification, feature)
38:   end for
39:   gradientMagnitude ←computeMagnitude(currentBodyWindow)
40:   for all procedureToExtractFeature do
41:     feature ←procedureToExtractFeature(gradientMagnitude)
42:     classification ← updateClassification(classification, feature)
43:   end for
44: end while
```

a meaningful time interval.
Each iteration of the while loop from line 14 to line 44 is used to process a new data window when a new sample arrives.
raw samples are multidimensional, as they are composed by acceleration on three axes. Loop at lie 19 is used to iterate over all the dimensions of raw samples.
Rows 20 to 23 are used to preprocess raw samples, by applying digital filters to remove noise and to separate the gravity acceleration from the body acceleration.
Loop at line 24 iterates over all the procedures to extract the features from a window (eg. min, max, average): for each procedure, the corresponding feature is extracted and immediately included in the classification. At row 28, the signal is derived, and in rows 29 to 31 the features of the derivative are computed.
At rows 34 to 38, the application extracts the features from the L2 norm of the multidimensional raw signal and uses them in the classification. At rows 39 to 43, the L2 norm is derived and the features of its derivative are extracted and used. The most important observation is that the goal of almost all the instructions is to extract new features and to use them. The fixed cost to pay independently from the number of features used is very low, so using ASVM potentially affects the majority of the instructions: in other words, ASVM is effective at its maximum in this code.

6.3.2 Implementation Details

In this section we will show the technical solutions that we have developed to address the requirements listed in Section 6.2. Each solution will be presented in detail, and we will discuss why it is useful to mitigate the effects of the corresponding problem.

- Fixed point representation of numbers: embedded devices do not always have a Floating Point Unit (FPU). Therefore, operations that involve numbers represented using a floating point notations take much longer to execute because they need to be emulated in software [20n]. Therefore, it is better to represent numbers using a fixed point notation. In the evaluation section we will present our experimental results, that demonstrate that using fixed point numbers it is possible to obtain a speedup greater than 8x. This optimization impacts the computation time needed to run the application, and it does take place in all the points of the code.

- No kernel trick: TPC devices have poor performances in terms of computational power and memory availability. As a consequence, it is necessary to cut down the memory consumption of the application as much as possible. Using a simple linear model, without projecting vectors in a higher-dimensional space using the kernel trick can greatly help in this sense.

As a matter of fact, the memory consumption of a linear model is

$$O(n_c n_d)$$

where n_c is the number of classes and n_d is the number of dimensions of the space of the samples. This is a major improvement with respect to the memory consumption of a model that makes use of the kernel trick, that is

$$O(n_{sv} n_c n_d)$$

where n_{sv} is the number of support vectors related to a specific class. Using a linear model does without projection does also help speeding up the inference phase: the time necessary to classify a sample is

$$O(n_c n_d)$$

compared to

$$O(n_{sv} n_c n_d)$$

when the kernel trick is used. In conclusion, not using the kernel trick helps solving both the problem of time and the problem of memory.

- Computation of features: when a new measure is gathered from the accelerometer, it becomes necessary to extract a set of features from the window composed by the last N samples (see Algorithm 1). The computation of the features may be very expensive, and scale with the dimension of the window. For example, the time T needed to compute the mean value of the window scales with the size N of the window as

  $$T = O(N)$$

  This behaviour can be problematic especially for some features to extract: for example, the standard deviation is more costly to extract than the mean. In particular, our tests, documented in the next chapters, show that trivial implementations may make impossible for the application to meet the time constraint declared in Section 6.2. Therefore, the algorithm carries out the computation of some features in an iterative way. Viz., the feature of the window defined by the arrival of sample T is computed starting from the same feature at time T-1. This feature helps in reducing the computation time of the application. For reasons of efficiency, we selected 140 features for the implementation among the 561 available.

6.4 Correctness Assessment

Some of the details of the implementation, listed in the previous Section 6.3.2, could hinder the correctness or the performance of the application. In particular, implementing from scratch digital filters and SVM is an error-prone

process. The same consideration is true for the utilization of fixed point numbers. Moreover, fixed point numbers could create a source of error other than the physiological approximation introduced by ASVM.
As a consequence, it is very important to

1. Test the correctness of the implementation

2. Measure the inaccuracy introduced by fixed point representation of numbers.

To carry out these tasks we created a Python program, functionally equivalent to the C firmware, and we compared the outputs of the two programs using different error metrics. In particular, we fed them with the same input and we measured the distance from their outputs.
We can safely assume that the Python implementation is correct because it was implemented using libraries from the widely used scipy package [20m]. Our implementation can be freely accessed [20p]. We wanted to discover how far are the results produced by our algorithm from the ones produced by the reference implementation.
To carry out this comparison, we have measured the distance between the variables in our algorithm with their corresponding doppelganger in the reference implementation. We wanted to discover two things: the maximum error that that the C implementation of our algorithm can make, and whether it produces results that are coherent with the Python implementation or not.
To understand the requirements of a meaningful distance we need to take into account the values of the variables themselves. Figure 6.2 shows the mean absolute value of some of the main intermediate variables of the algorithm.
The mean absolute values are distributed along more than three orders of mag-

variable name	mean absolute value	minimum	maximum	MAPE	p-value
body	0.353475	0.135254	0.644653	0.005741	2.14488e-30
derivative	0.094462	-0.157715	0.111084	0.004559	0.073307
bodymag	90.195567	81.560059	102.536499	0.005977	NA
jerk	0.088725	0.023315	0.314819	0.012017	NA
lowpassed	1.406280	-2.422241	1.993530	0.002208	0.415821

Figure 6.2: Variables and associated metrics

nitude, so we cannot choose an absolute distance: the metric that we choose has to be scaled according to the order of magnitude of the variables themselves.
Moreover, the sign of the variables is not constant, so we cannot use some class of metrics that rely on the assumption that the quantities to measure are concordant, such as Mean Percentage Error [Hil03].
We have used Mean Absolute Percentage Error [20r], a metric that is scaled and works independently from the sign of the points to measure.

Given a set of vectors

$$\boldsymbol{x_i} = [x_{i1}, x_{i2}..x_{in}]$$

MAPE is defined as

$$MAPE(\boldsymbol{x_i}, \boldsymbol{x_j}) = \frac{1}{n}\sum_{k=1}^{n}\frac{|x_{ik} - x_{jk}|}{|x_{ik}|}$$

We used this metric to detect the situations where the actual variable is far from the expected value.
However, pure MAPE can create errors when numbers are represented using fixed point notation. As a matter of fact, representing a number x using fixed point notation with I integer bits and D decimal bits can create an error ϵ bound by

$$|\epsilon| < 2^{-D} \; if \; |x| < 2^{I}$$

let us consider the situation when we have two numbers x_i and x_j, whose representations with fixed point notation are $x_i \pm \epsilon_i$ and $x_j \pm \epsilon_j$. Let us compute the error on the MAPE. MAPE is obtained from the original numbers by performing an addition and a division. Absolute errors sum up for additions, so the absolute error ϵ_{ij} of $x_i + x_j$ is

$$\epsilon_{ij} = \epsilon_i + \epsilon_j$$

for divisions and multiplications, relative errors sum up. So the relative error ϵ_{rMAPE} is

$$\epsilon_{rMAPE} = \frac{\epsilon_{ij}}{x_i - x_j} + \frac{\epsilon_i}{x_i}$$

ϵ_{rMAPE} can become huge, for example, when ϵ_i is comparable to x_i, that is to say when

$$|x_i| \approx \epsilon_i < 2^{-D}$$

In this situation MAPE becomes useless. So, we took into account this situation by ignoring all the quantities whose absolute value is below a threshold that we set to 2^{-D+3} after some experimental tuning.
Once we made this distinction, we can freely and consistently use MAPE to detect in our code the situations when a variable is too far away from its expected value.
Column "MAPE" of Figure 6.2 shows adjusted MAPE of the main variables in the code after the correction we have explained in the previous paragraph.
After having shown that the maximum error done by the C implementation compared with the Python implementation is small in absolute terms, we are also interested in discovering whether it is random or the values of C variables are consistently below or above their Python twins.
We carried out this test using the following steps.
First of all, we have computed the differences between the C value and the Pyehon value of the variables listed in Figure 6.2.
Let a_{it} be the actual value (the value in the C implementation) of variable i at the

time step t in our algorithm. Let e_{it} be its value in the Python implementation, that is to say its expected value in the algorithm. Let

$$d_{it} = a_{it} - e_{it}$$

be the difference between the actual value and the expected value. Let

$$\boldsymbol{d_i} = [d_{i1}, d_{i2}..d_{iT}]$$

be the vector of the values of the difference. Our thesis is that

$$\boldsymbol{d_i} \leftarrow N(\mathbf{0}, \sigma^2)$$

We have carried out this investigation in two steps: first of all we performed a Pearson test on all the $\boldsymbol{d_i}$, rejecting the null hypothesis of normally distributed samples when the p-value was below 0.01.
Then, we discarded the variables whose $\boldsymbol{d_i}$ is not normally distributed, and we applied a t-test in the remaining ones, testing the null hypothesis that the mean of the population is zero.
Column "p-value" of Figure 6.2 shows the p-values of this test, when applicable. For "bodymag" and "lowpassed" there is strong evidence to say that the mean of $\boldsymbol{d_i}$ is zero. This is not true in the case of "body". In conclusion, we can say that the results produced by our implementation are extremely close to the ones produced by the reference implementation. Moreover, we can say that for most variables the results of our implementation are consistently not below or above the corresponding number in the reference implementation.

6.4.1 Enforcing correctness

We have enforced the correctness of the code by running tests during the execution of the application.
In particular, we included a test stage during the pipeline to compile the firmware for MSP430: the stage consisted in generating and executing a special firmware, that included assertions to prevent us from testing bugged firmwares.
The tests were structured as follows: we included in the code the expected values of several intermediate variables extracted from the Python implementation. The firmware tested for each of these values that the corresponding variable computed by the C program was close enough (MAPE <15%).
These tests were removed in the version of the firmware we used to measure the performance of the application.

Chapter 7

Evaluation

7.1 Abstract

In this chapter, we will present how we measured on the field the performance of ASVM, our proof of concept application, shown in Chapter 6. We have demonstrated that it is correct, so we can now gather scientifically sound measures on its performance.
Our goal is to carry out a simulation of how ASVM behaves in realistic scenarios, in terms of timeliness, energy consumption and accuracy of the result. To simulate realistic energy conditions, we will use as input the voltage traces used for the evaluation of EPIC [Ahm+19].
The logical tools that we need to set up an experimental procedure are summarized in 7.1. They are:

1. A mapping from the energy that is available in the environment to the number of clock cycles that can be afforded consuming that energy. This mapping has been widely investigated in the literature, for example in EPIC [Ahm+19], so we will use an existing solution for this step.

2. A mapping from the number of clock cycles that can be invested in a computation to the number of features that can be used without exceeding that number. We constructed this mapping by emulating our application on an MSP430 board, using MSPSim [20o]. The details of this step are given in Section 7.3.2.

3. A mapping from the number of features used to the accuracy of the classifications that are emitted. We have computed analytically this mapping in the previous Chapter 5.

Each tool will enable one step towards the quantification of the performance of our application. We have measured the accuracy, the timeliness and the energy cost to pay, both on ASVM and, using the same experimental procedure, on an application that makes use of a state-of-the-art technique, namely checkpoints,

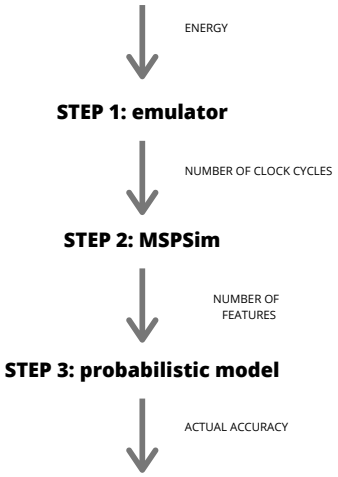


Figure 7.1: Experimental setup

to address the problems of TPC.
ASVM obtained energy savings of up to 41%, with accuracy losses always lower than 20%. This comparison allows us to understand that our solution emits results consistently in advance with respect to the checkpoint-based solution, up to 10s. Further details about this dimension are given in the rest of the section.
Operatively, the chapter is structured as follows. Step 1 requires a mapping from the energy available to the number of clock cycles that can be afforded. We computed this mapping by using a simulator developed by Bertani: the details of this first step are given in Section 7.3.2. Step 2 requires an implementation of our application for a microcontroller. We have developed such implementation in the form of a C program. This implementation introduces potential sources of error, both due to human errors and to several implementation details that are necessary to fit the application on a microcontroller. Therefore, we have thoroughly tested the program, as explained in the first part of Chapter 6.
We have also measured empirically the accuracy of the analysis of the algorithm presented in Section 9. At this point, we will have all the building blocks of the evaluation procedure. So, we will apply the evaluation procedure to understand the behaviour of our solution both in absolute terms and compared to a state-of-the-art solution. In the last part of the chapter, we will present the metrics obtained by our application and by the state-of-the-art application.

7.2 Accuracy of the Analysis of the Algorithm

In this section, we will explain how we measured the accuracy of the analysis of the algorithm that we carried out in Section 9. We will present different classification problems that we will solve using Anytime Support Vector Machines (ASVM). For each classification problem, we will compare the expected performance computed using the probabilistic model of the algorithm with the actual, experimentally measured performance.
We computed these experiments using two different datasets to make the test more sound from a scientific point of view. Had we used a single dataset, one could have objected that it is by chance or lucky guess that our prediction for the accuracy matches the real measured value: using two different datasets, this objection becomes much weaker.

7.2.1 Datasets

The first dataset that we have used is Human Activity Recognition Using Smartphones Data Set [Ang+13], from now on Dataset D1. For an accurate description of the features of the dataset, see Section 5.5.1: in this paragraph, we will summarize the main aspects of the dataset.
The dataset is the result of an important preprocessing operation performed on raw data gathered using smartphone accelerometers. Raw data have 6 dimensions, while the final dataset has 561. This is important because it means there are 561 precision levels in the algorithm. To have an idea of what it means, precision scaling has as many precision levels as the number of bits used to represent numbers, that is to say 16 or 32, on microcontrollers, while function perforation has as many precision levels as the number of function calls in an application (in the case of our algorithm, more than 5000). So, ASVM is a technique quite flexible, and this feature is very useful, as we have shown in Figure 5.1.
We have also used another dataset, from now on D2, namely WISDMActv1.1 [20s]. This dataset has been analyzed by Kwapisz et al. [KWM11], who used Long-Short Term Memory Recurrent Neural Networks to perform human activity recognition.
This dataset is composed by 46 features, that are computed from 6-dimensional data points sampled at a frequency of 20 Hz.
To obtain results that are comparable to the ones of the first dataset, we have used the raw data of the second dataset and applied the same preprocessing steps, obtaining again a dataset composed by 561 features.
As we mentioned in Chapter 6, we selected 140 features for the implementation.

7.2.2 Four Cases

In the following four sections, we will present the result of tests on the field of the accuracy of the analysis of ASVM that we will detail in Chapter 9. Each section is dedicated to one of the four variations of the analysis. All the sections

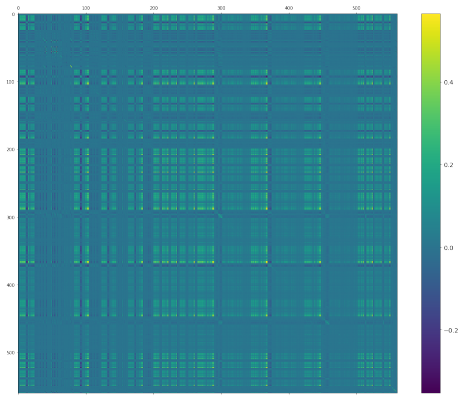


Figure 7.2: Covariance matrix of processed dataset N1.

are structured in a similar way. First of all, we have cleaned the data: in case of binary classification, we selected only samples from two classes, and excluded the other ones, while in the case of independent coefficients we projected the points using PCA, to prevent correlation from different features.
The results of the analyses are presented in the same way, by means of graphs. One such graph is shown in Figure 7.4. Each point on the graph represents a classification. Let P=(x,y) be a point on the curve. Its coordinate x represents the number of features used for the corresponding classification, while y represents the accuracy of the classification itself.
There are two curves, one labelled as "expected" and the other labelled as "measured". The former represents the accuracy expected by the probabilistic model of the algorithm, while the latter represents the accuracy of the model measured experimentally on a dataset, either D1 or D2 according to the situation.
The closer are the curves, the better is our probabilistic explanation. We have also computed the Mean Square Error between the curves, to summarize in a single number the goodness of our forecast.

Binary Classification, Independent Coefficients

The first test we carried out was the scenario when the data samples have to be classified in just two classes, and the coefficients of a sample are independent from one another.

Figure 7.2 shows that these hypotheses do not hold true for the datasets in question. The figure displays the covariance matrix of N1. The precise numeric values are not important in this context: it is enough to observe that the covariance between the features of the dataset is not zero even outside the main diagonal. Moreover, the dataset contains samples that are labelled in six different ways.
Therefore, we projected the points in the dataset using Principal Components Ananlysis (PCA) to uphold with the hypotheses of this version of the algorithm.

To obtain a binary classification problem, we compared the classes pairwise.
As for the independence of the coefficients, we have projected data using PCA [FRS01]. This technique allows adjusting the coordinate system of an arbitrary dataset into a coordinate system that offers interesting guarantees.
The most important property is that the variance of the points is maximized along the new dimensions, and the covariance between the dimensions becomes zero [20q]. Figure 7.3 shows graphically the effect of PCA on a simple dataset. We have applied our algorithm on the projected dataset N1, keeping only the

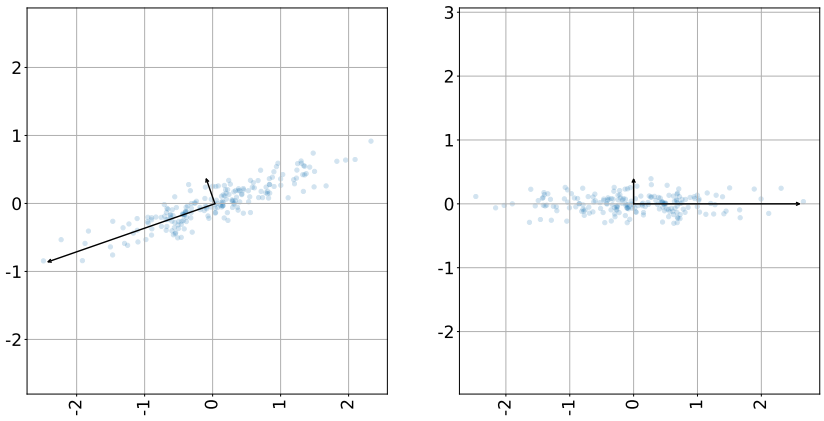


Figure 7.3: PCA applied to a simple dataset

samples labelled as running or walking, the two most numerous classes, obtaining the result in Figure 7.4 The Mean Square Error is 0.0017. From Figure 7.4 one can observe a very steep increase in the accuracy when the first features are used. As a matter of fact, walking and running can be easily distinguished. Therefore, we repeated the test classifying the activities WALKING UPSTAIRS and WALKING DOWNSTAIRS, that is the single pair where it is easier to make confusion [Moh16]. We carried out another test, using dataset D2. As we have explained before, using two distinct datasets makes our tests more solid. The output of the second test is shown in Figure 7.5. It is possible to observe that both the predicted accuracy and the actual accuracy struggle to reach a satisfying performance level.
Looking at the curves, we can observe that they are close to one another. To quantify their proximity, we computed the MSE between the corresponding point in the curves: MSE is 0.0022. This means that the RMSE is around 0.04, which is an estimate of the error of our forecast with respect to the actual behaviour.

Multiclass Classification, Independent Coefficients

In this section, we will present the outputs of the tests of the second version of the algorithm. It is different from the first one that we have tested in Section

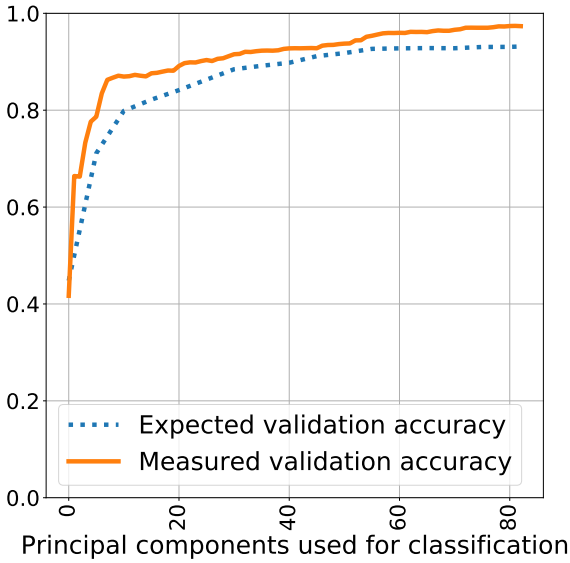


Figure 7.4: Accuracy of classification WALKING/RUNNING

7.2.2 because the data points have again independent coordinates, but they are classified along multiple classes.
One consequence of this aspect is that our model is more costly to compute. In particular, the expected accuracy can be computed using Equation 9.1, that is an N-dimensional integral, where N is the number of classes. Multidimensional integration is a hard problem, where it is extremely challenging to reach triple-digit accuracy when the number of dimensions is above 7, and where even the integration with 3 to 7 dimensions is very costly [Coo02].

As a consequence, we did not compute the expected accuracy for all the principal components, but only for 20 equally spaced points. We have observed that this number is representative of the shape and the value of the function.
Figure 7.6 shows the expected accuracy and the observed accuracy for the 6-class classification of the samples of Dataset D1. It is possible to notice that the prediction is less accurate than the previous case. This happens because of the numerical approximation of the integration that is involved in the computation, that we decided to perform using the trapezium rule on a sparse grid of points. This perception is confirmed by the MSE, that is higher than the previous case: in this case, it is 0.0026. In absolute terms the prediction remains good, as RMSE is 0.051, that is to say around 5% of the value to predict.

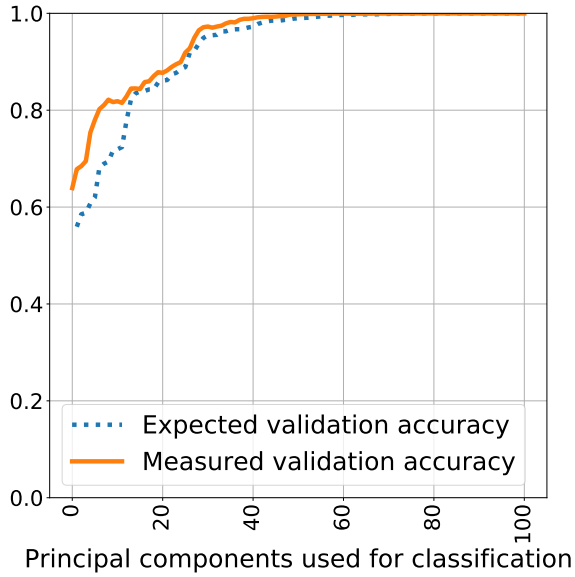


Figure 7.5: Accuracy of classification WALKING UPSTAIRS/WALKING DOWNSTAIRS

Binary Classification, Correlated Coefficients

As anticipated in Section 5.4.1, this specific version of the algorithm has not been analyzed from a theoretical point of view. The reason is that it is a less general version of the algorithm commented in Section 9.2.1 the situation when there are multiple classes, and the features are correlated.
Although the situation in question is not thrilling from a theoretical point of view, this scenario is a nice opportunity to visualize and explain some curious behaviours that we can observe experimentally.
In particular, we will plot the curve of the accuracy as a function of the number of features used, and explain some interesting details in the shape of the curve. We do not need a real dataset for this analysis: we generated a random covariance matrix to have control on its shape. The matrix is shown in Figure 7.8: the precise values of the covariance matrix are not important, it is sufficient to observe that it is not diagonal, and so the features of the dataset are correlated. We have used this covariance matrix in Figure 7.8to compute the expected accuracy as a function of the number of features used for the classification. This quantity is showed in Figure 7.9.

Figure 7.9 shows the probability that a partial classification is coherent with the final classification when the number of features used for the classification increases. It is possible to notice three interesting features of the function:

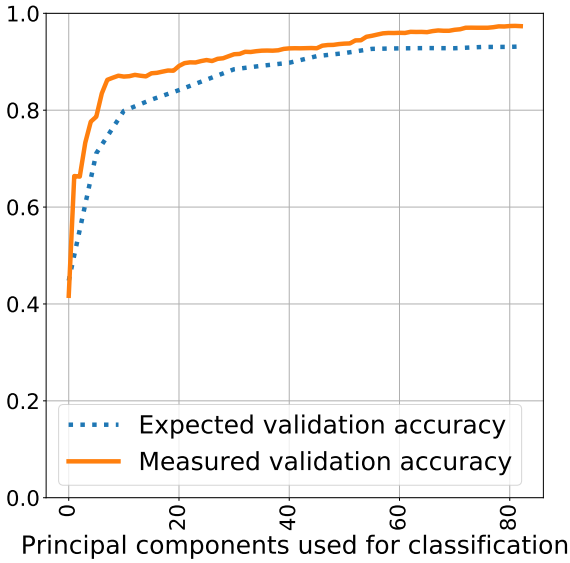


Figure 7.6: Accuracy of multiclass classification on D1

- The function's value is 0.5 when no feature is used: this makes sense because if a point is classified using no features it is necessary to purely guess at the class it belongs to. As there are only to classes, the probability of being right is 0.5
- The function's value is 1 when all the features are used: similarly, when a partial classification is computed using all the features it is by definition identical to the final classification. Therefore, the probability that these two numbers are identical is 1 as well.
- The function is generally increasing: when a feature is included in the computation of the partial classification, two things happen. First of all, it gives its contribution to the partial classification, either increasing or decreasing it. Second, the variability of the set of features that have not been used yet decreases. Intuitively, the information contained in the feature is moved from the possible future corrections to the deterministic present classification.

As a consequence of the third point, the probability that a partial classification remains coherent with the final one does increase as more and more features are used, apart for some random oscillation like the one highlighted by the circle. We will now provide an explanation for this kind of apparently incoherent behaviour: we will also provide a toy example to confirm our explanation. The

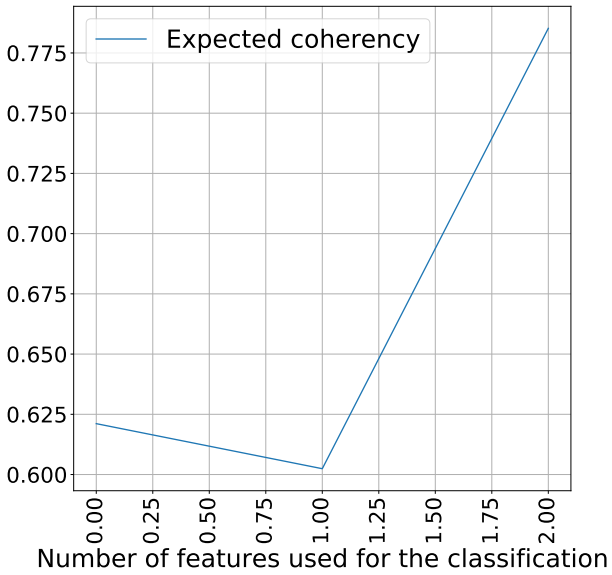


Figure 7.7: Expected coherency of classification in a peculiar case. The graph is zoomed from clarity

intuitive description of the situation is this one: there is a certain number N of features, where the first and the last $\frac{N}{3}$ ones contribute positively to the classification and the $\frac{N}{3}$ in the middle contribute negatively. Under these conditions, a typical sample will have its distance from the classification hyperplane initially increased, then decreased and finally increased again. Therefore, it will be more likely to have a positive distance from the hyperplane when the features in the middle have not given their contribution yet.
This situation can be quantified using the following covariance matrix.

$$\begin{bmatrix} 10 & -10 & 10 \\ -10 & 15 & -10 \\ 10 & -10 & 20 \end{bmatrix}$$

this matrix is symmetric and Positive Semidefinite, so it is a valid covariance matrix. In this situation, the probability of obtaining a coherent classification is shown in Figure 7.7 In conclusion, it is possible that the inclusion of an additional feature lowers the probability of obtaining a coherent classification. We have this way provided an explanation for the only counterintuitive phenomenon that it is possible to observe empirically in our experimental setup.

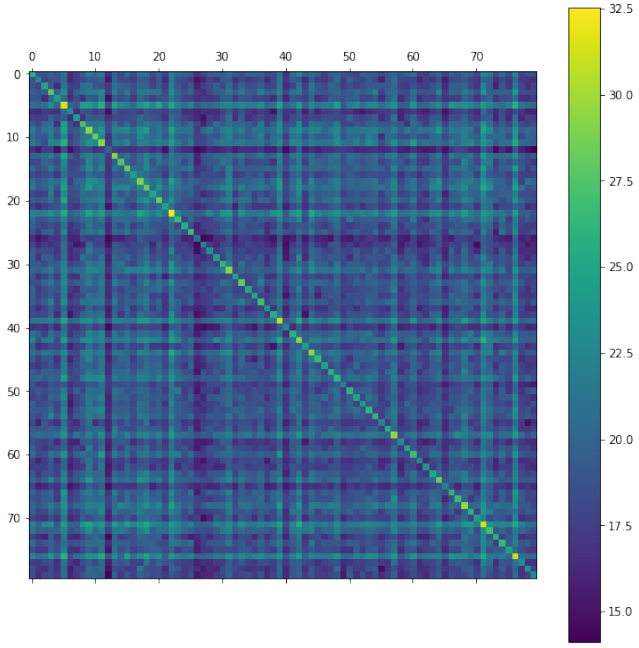


Figure 7.8: Random covariance matrix

Multiclass Classification, Correlated Coefficients

The testing of the version of the algorithm that tackles the situation of multiclass classification when the features are correlated will be carried out in a future work.
The scope of this work is to develop an application that makes use of AC techniques in a TPC context. We want to reach this goal in the leanest possible way: our objective is to demonstrate the feasibility of our project, without necessarily developing a complete and perfect system.
In the next section, we will measure on the field the performance of the algorithm.

7.3 Experimental Setup

In the previous Section 7.2, we have measured the accuracy with which our probabilistic description of ASVM provided in Section 9 describes the actual behaviour that can be observed with a real dataset.
Now, we can measure the performance of the entire application, to finally prove that ACTs can be applied fruitfully in TPC contexts.
The goal of this section is to describe what are the metrics that we have decided to measure, and the procedure we followed to measure them. In particular,

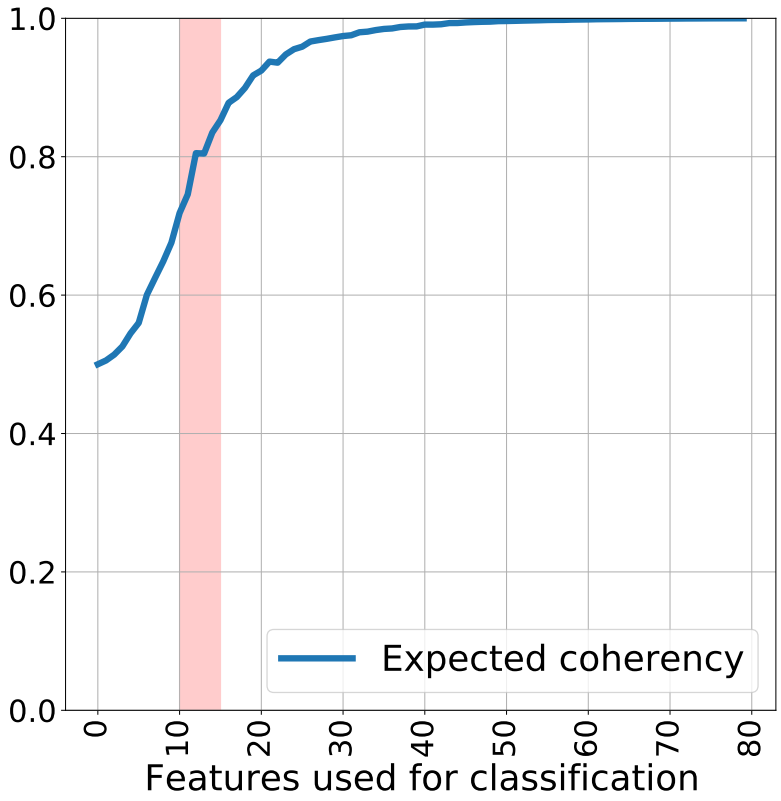


Figure 7.9: Copy of Figure 5.7. Expected coherency of binary classification of points with correlated features. Highlighted an oscillation.

we are interested in quantifying the timeliness, the power consumption and the accuracy of ASVM, and to compare ASVM's metrics with the metrics scored by a state-of-the-art application, based on checkpoints.
In Section 7.3.1, we will present the metrics that quantify the inputs and the outputs of the experiment, while in Section 7.3.2 we will describe the procedure that we used to measure these metrics. The procedure is composed by three steps: obtaining a mapping from the energy that is available to the affordable clock cycles, then a mapping from the available clock cycles to the number of features that can be used, and finally a mapping from the number of features to the expected accuracy of classifications.

7.3.1 Metrics

Our experiment consists in feeding an emulated MSP430 microcontroller with a voltage trace, to measure the number of clock cycles that are available for each classification, and from the number of clock cycles computing the accuracy of the classifications emitted using that particular voltage trace. We will also measure the timeliness of the classifications, compared to the timeliness of a checkpoint-based solution.
At the beginning of this Chapter 7, we have described the logical steps that compose the evaluation procedure. Now, we will describe the inputs and the outputs of this procedure, and then we will show the results that we have obtained when we have fed these inputs in the experiment.
The primary goal of Approximate Computing Techniques (ACTs) is to give developers the possibility to reduce the quality of the result produced by programs to lower the cost necessary to produce them.
In Chapter 4 we have discussed why the way approximate computing is traditionally applied does not meet the requirements of TPC contexts, and we have proposed a new paradigm. The goal of AC, when applied in TPC scenarios, must be to maximize the quality of the result produced by applications with the resources they can consume.
We have also highlighted that the key to follow this process is to formalize two functions, that is to say, an objective function that encompasses what is quality for the program that is being approximated, and a constraint function that represents the resources that can be consumed to obtain the result.
In this section we will propose the functions that represent cost and quality in the specific case of our algorithm ASVM.

Output 1: Accuracy

The primary goal of HAR is to correctly recognize the human activity that is being performed. In several influential papers in the field [YWC08][Ang+12a] accuracy is preferred to recall because in this domain the consequences of false negatives are not serious. We will follow this line of reasoning, using accuracy as well.
The goal of the application is to maximize the accuracy of the classification it produces. In Chapter 5 we have proven that the accuracy of the results produced by our algorithm is connected to the degree the computation is carried out, and we have quantified this bond. So, we have a precise method to map each state of the algorithm to the accuracy of the classification it is expected to produce: in other words, we can precisely measure this metric.

Output 2: Timeliness

An important aspect of TPC applications is that the concept of time works differently than in traditional, continuously powered scenarios. This problem has been highlighted for example in Mayfly [HSS17] and InK [Yun+18].

The risk of TPC application is that they may process old data without knowing that they have been gathered a long time ago, and so they may have become outdated and useless.
Therefore, it is important to understand how does a TPC application deal with the problem of time. Our algorithm tackles this issue in a very straightforward manner: it yields classifications as soon as they are ready, or when the device is going to shut down.
As a consequence, by definition, it does not process samples that have been gathered long ago in time. However, this qualitative description is not satisfying: we need to quantify it somehow.
This property is the key benefit of our work: our contribution is a technique to use the scarce resources available in TPC in a smart way, avoiding that computation batches pile up because producing accurate results is too costly. On the contrary, quickly emitting reasonable results allows our application to run smoothly.

Output 3: Consumed Energy

One of the primary goals of our work is to exploit all the available energy in an efficient way. Therefore, we are interested in discovering how much energy has been consumed by the application. However, in this context, we do not really need the amount in Joule, because this number would be highly dependent on the underlying hardware. To give a more precise and meaningful indication we will measure the number of clock cycles that are necessary to complete the execution of the application. This number is a proxy of the amount of energy consumed, but at the same time, it can be compared with other boards that need to share only the same instruction set and not the entire hardware setting.

Input: Available Energy

given that there are enough memory and computational power, in TPC contexts, the single scarce resource that limits the execution of applications is the amount of energy that is made available by the environment.
Therefore, the constraint our application is subject to is that is must not consume more energy than the one actually available.
We have enforced this constraint using a simulator that mimics the physical behaviour of msp430-x boards, including their energy consumption. So, we can make sure our algorithm does not exceed the consumption it can afford.
To simulate realistic behaviour, we have used real voltage traces measured by physical energy harvesters to power up the emulated device. The traces are taken from EPIC [Ahm+19].
Each trace represents the power produced by a different energy harvester: the traces have different variability and characteristic frequencies according to the power source they are gathered from.

7.3.2 Procedure

To measure the metrics listed in the previous Section 7.3, we have set up an experimental procedure to guarantees accuracy and repeatability. The experimental setup is shown in Figure 7.1. The steps are the ones that we introduced at the beginning of this Chapter 7: the first step consists in measuring the number of clock cycles that can be invested in a classification when a given voltage trace is available. The second one is mapping the number of clock cycles to the number of features that are available. The third and last step is mapping the number of available features to the accuracy of classifications. The next three sections are dedicated to these steps.

Step 1: Simulator

Our colleague Francesco Bertani developed a simulator that is able to compute the time intervals during which a MCU remains turned on when a given voltage trace is available. Moreover, it returns the number of clock cycles that are available during each time interval. So, we fed the voltage traces described in Section 7.3.1, and we measured the corresponding on intervals and clock cycles availability.
The Simulator models the MCU using a capacitor that stores energy, connected to a voltage generator that represents the energy harvester and to a current generator that drains current from the capacitor, which represents the microcontroller.
The microcontroller drains away current when there is some workload to perform. In particular, accelerometer samples arrive at a constant rate, and each one adds a constant amount of clock cycles to handle.
In our approximate and ASVM based simulation, the workload is brought to zero every time the device experiences a power failure: this behaviour represents the fact that computation is interrupted by power failures and classification are immediately emitted, coherently with the behaviour illustrated in Figure 6.1. Figure 7.10 (also in the introduction, copied here for clarity) shows how the simulation works. Sample 1 arrives to the application, and the device can spend all the time necessary to compute and use all the features of the sample. On the contrary, a power failure takes place while sample 2 is being processed. As a consequence, the application cannot invest in the classification of Sample 2 all the clock cycles that would be necessary to compute and use all its features.
We have decided to compare our solution with a solution based on checkpoints. For a technical description of checkpoint based solution see Section 2.6.1: the general idea is to store the state of the computation is case a power failure occurs. In these cases, a fixed amount of clock cycles is added to the workload: this represents the clock cycles spent to save the state of the computation to persistent memory.
This different behaviour has three main effects on the evaluation metrics. First of all, every sample is classified in the best possible way, computing and using all its feature: in other words, checkpoint-based solution reach the best possible

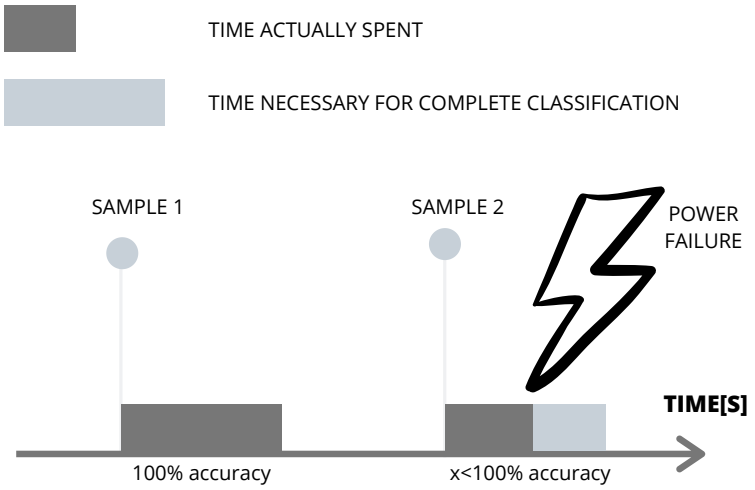


Figure 7.10: A complete classification and an incomplete classification

accuracy. Second, a certain amount of clock cycles is spent in saving and restoring the checkpoint, so this kind of solution has an overhead in terms of energy cost. Finally, the moment when classifications are emitted can be far away in time in case the board remains off for a long time interval.
We have measured the extra consumption and the delay paid by a checkpoint based solution by executing it in the same scenarios of our algorithm. These metrics will be shown in the following Section 7.3.3.

Step 2: MSPSim

The goal of the second step of the experiment is to understand how many features can be used when a given number of clock cycles is available. We have decided to perform this measurement using a simulation, for one main reason. The goal of this work is to demonstrate the feasibility and the usefulness of applying ACTs in a TPC context. To reach this objective, we do not need to craft a physical prototype: a simulated environment allows us to measure all the metrics we need, more precisely and with less interference than a physical experiment, and in a replicable way.
The simulation make it easy to measure arbitrary metrics about what is happening without interfering with the phenomena we are interested in. We carried out the simulation using MSPSim [20o]. MSPSim is a Java-based instruction level emulator of the MSP430 series microcontroller (MCU).
There are two motivations that make MSPSim particularly suited to our needs. First of all, MSP430 MCU are extremely low power [19o], and so they are often used in TPC applications. As a consequence, this simulator reproduces the behaviour of our algorithm on a device that could realistically be used for actual

deployment.
Moreover, MSPSim is open source and widely contributed. The fact it is open source makes it possible to detect its flaws and to objectively assess the correctness of our simulation, and this would be impossible with a proprietary simulator. The fact it is a participated project makes it real that people contribute and correct its bugs.
To use MSPSim, one must compile normal C programs using MSP430 toolchain, and feed the compiled program to the emulator. MSPSim runs it, and provides several metrics about the execution.
In particular, it measures the number of clock cycles spent by the application in each function and for the overall execution.
To obtain the map from the number of cycles to the number of features that can be used, we used the following procedure. Let us consider the situation when we need to discover the number N of features that can be used when C clock cycles are available. Initially, we make a guess on N, let us say N_1. Then, we run a modified version of the algorithm that is forced to terminate after it has used N_1 features, and we measure the number C_1 it took to execute this computation. We repeat this procedure for all the values N_i of the number of features to use, and we obtain the function that maps the number of features used into the cost in clock cycles to obtain them. At this point, we can invert this function to obtain the map from the number of clock cycles to the number of features. Figure 7.12 shows in a graphical way the procedure we have described here.
Now we have the tools to complete step 2 (see the beginning of this Chapter 7), that is to say, a mapping from the number of clock cycles to the number of features that are available.

Step 3: Probabilistic Model

The final goal of the experiment is to measure the accuracy of the classifications that can be produced by our algorithm in the situation when a certain amount of energy is available over time.
So far we have obtained a map that allows us to understand how many features will the algorithm be able to use given that a certain number of clock cycles can be invested. This map has been obtained experimentally, following the procedure described in the previous Section 7.3.2. Previously, we have obtained a map from the number of features to the accuracy: we have computed it theoretically in Section 5.4.1, and we have tested its correctness in Section 7.2. Now, we can stack the maps to obtain a function from the number of clock cycles invested to the expected accuracy of the classification that will be produced. The accuracy of a classification is computed every time classifications are emitted: when a classification is emitted because the algorithm has completely processed it. In particular, the emulator checks the number of clock cycles that have been spent for the classification, and infers the expected accuracy using the function that maps the number of clock cycles invested in a classification to the expected accuracy of the classification itself. The way we computed this

function has been explained in the previous Section 7.3.2. Therefore, the emitted classification will be based on the subset of features that have been computed: the number of features that can be computed can be inferred from the first step of the experimental environment explained in Section 7.3.2. The fact that the classification is based on a subset of the features does negatively affect the accuracy of the classification: in particular, the accuracy as a function of the number of features used has been computed in Section 5.4.1.
The behaviour of this solution is shown in Figure 7.11 When a sample arrives,

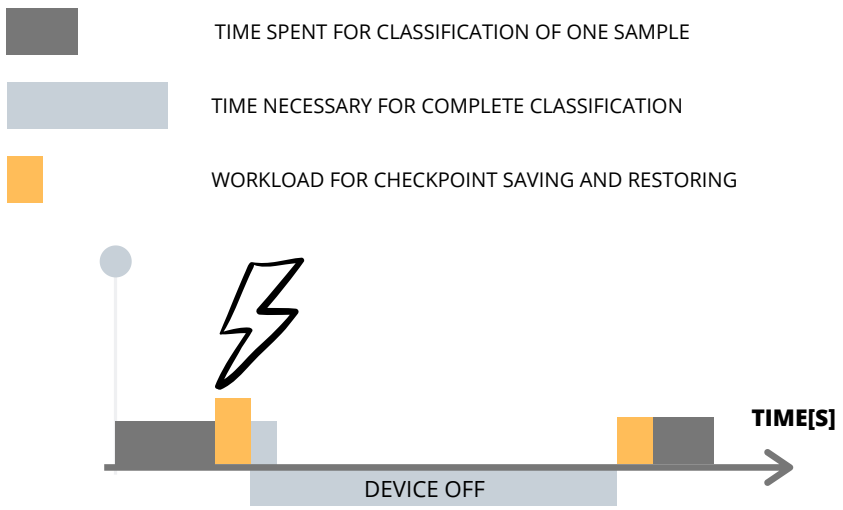


Figure 7.11: A checkpoint based solution

the algorithm starts to process it. When the energy available is too low, the algorithm is interrupted and the device stores the state of the computation into a checkpoint. After the checkpoint, the board remains off for a certain amount of time. When the device resurrects, it restores the state of the computation and then continues the classification. In the next section, we will illustrate the results of the experiments whose setup we have described in this section and in the previous three Sections.

7.3.3 Results

In this section, we will comment on the metrics that we measured executing our algorithm in different scenarios characterized by different energy availability.
As explained in the previous Section 7.3.2, we have designed a simulation environment where our application is executed in a way that is coherent with the energy conditions that we describe by means of a voltage trace, that has been measured once by a physical harvester and that we can arbitrarily replay.
This experimental setup allows us to measure the metrics listed in Section 7.3 relative to realistic executions.

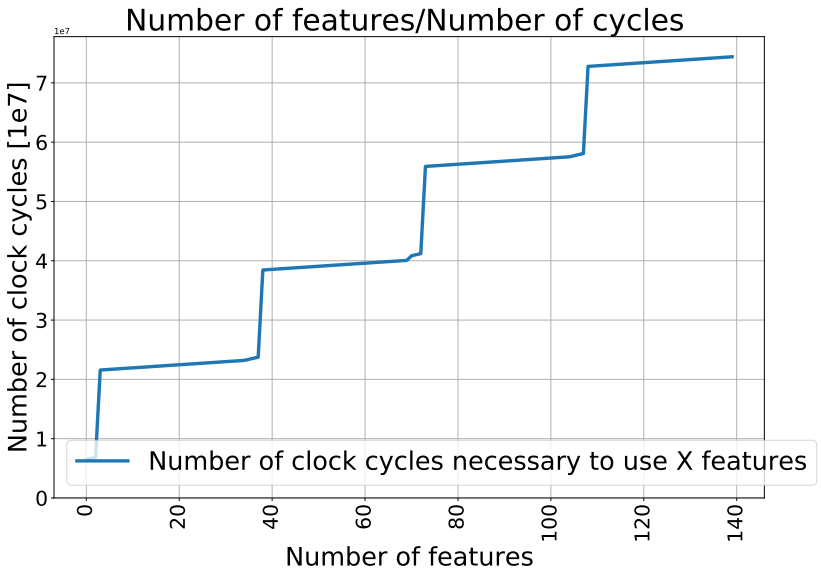


Figure 7.12: Clock cycles necessary to use a given number of features

Clock cycles to Accuracy

As explained in Section 7.3.2, a fundamental part of the experiment consisted in executing our algorithm using MSPSim, an emulator to reproduce the behaviour of embedded devices belonging to the MSP430 family [19o]. We used the simulator to compute the cost of computing and using a given number of features, for every number of features. The result we have measured is shown in Figure 7.12. It is possible to notice several steps in the function: this happens because one step of the algorithm is a Fast Fourier Transform (FFT) of the window, that is a very costly operation that makes available a great number of features immediately.
When it is necessary to compute the FFT of the window to obtain the subsequent feature, a huge number of clock cycles must be invested without immediate return. We can invert the function shown in Figure 7.12 to obtain the number of features that can be used for a classification as a function of the number of clock cycles that are invested in the classification. This function is shown in Figure 7.13. It is the function we called f in Section 7.3.2. This function can be combined with the map from the number of features used for a classification to the expected accuracy of the classification.
However, there is an implementation detail that is interesting to point out. As we have highlighted in Section 7.2.2, there are situations when including an

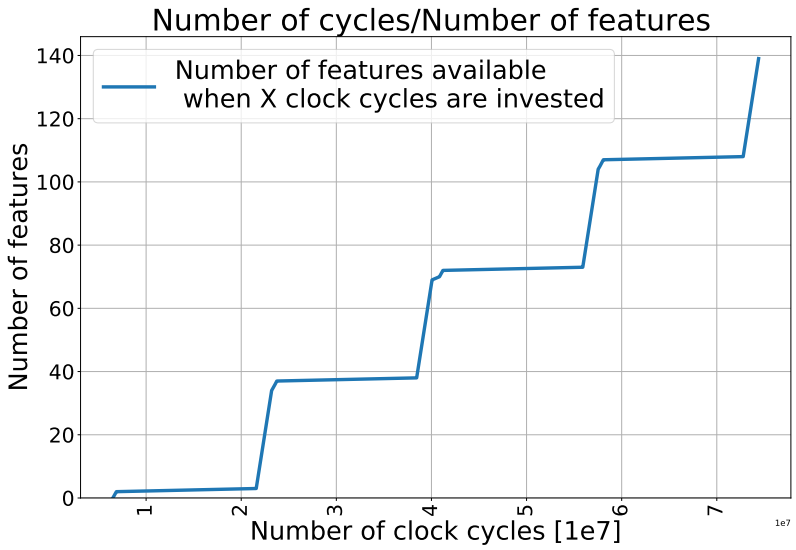


Figure 7.13: Features available as a function of the clock cycles invested

additional feature does not immediately improve the accuracy of a classification. In these cases we can mitigate the effect of counterproductive features by ignoring them. The practical effect of this choice is a transformation of the function from the number of features used to the expected accuracy. We have applied this correction in the test, as shown in Figure 7.14. This function and the function from the number of clock cycles to the number of features can be combined, to obtain the function from the number of clock cycles invested in a classification to the expected accuracy of the classification itself. This function is shown in Figure 7.15, for the case of a binary classification between points labeled as running and points labeled as walking. This function is the final output of the first step of the simulation: we can now use it to measure the performance of our algorithm in the second step of the simulation.

Performance Metrics

In this section we will combine the results obtained so far to measure in a realistic scenario the behaviour of our algorithm. As we have explained in Section 7.3.2, the experimental procedure consists in simulating the execution of the algorithm powering up the device used for the computation with a voltage trace measured by a physical harvester.

Figure 7.16 shows the results of the final step of the evaluation. Each point in the graph represents the result of an execution with a different voltage trace. Let E=(x,y) be a point in the graph: x represents the cost of the execution,

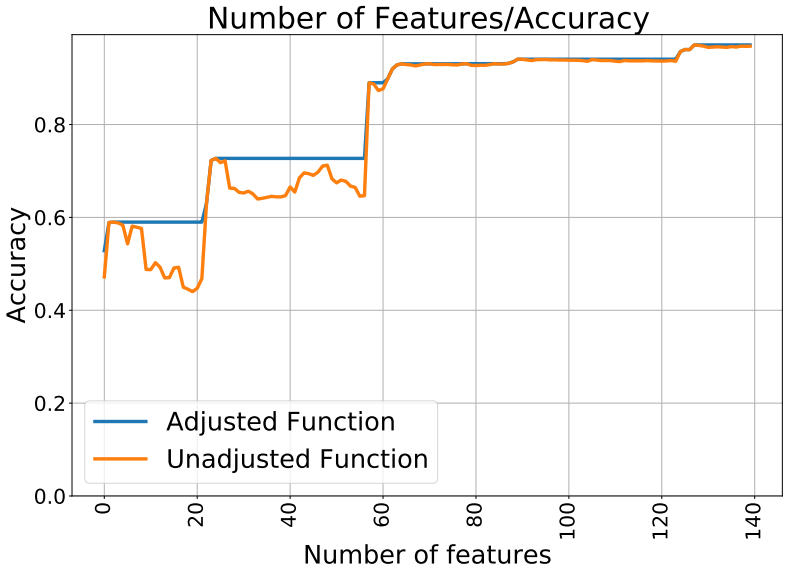


Figure 7.14: Expected accuracy as a function of the number of features used

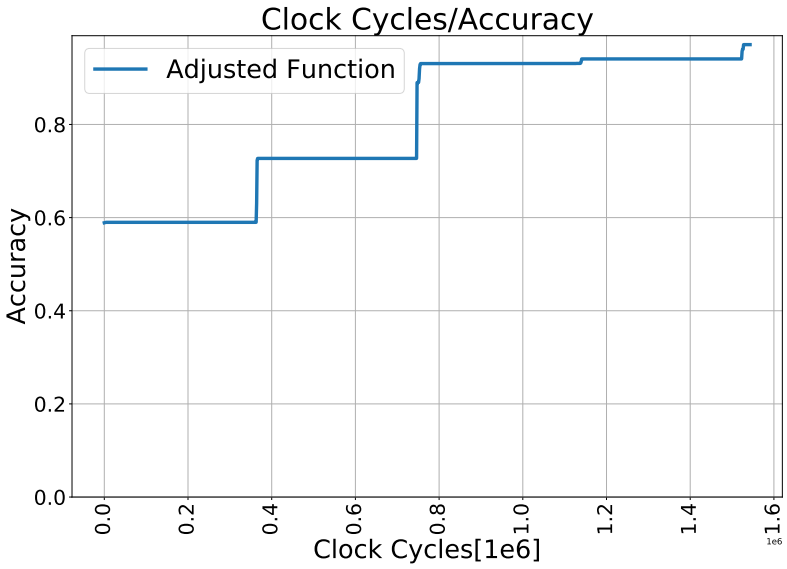


Figure 7.15: Expected accuracy as a function of the number of clock cycles invested

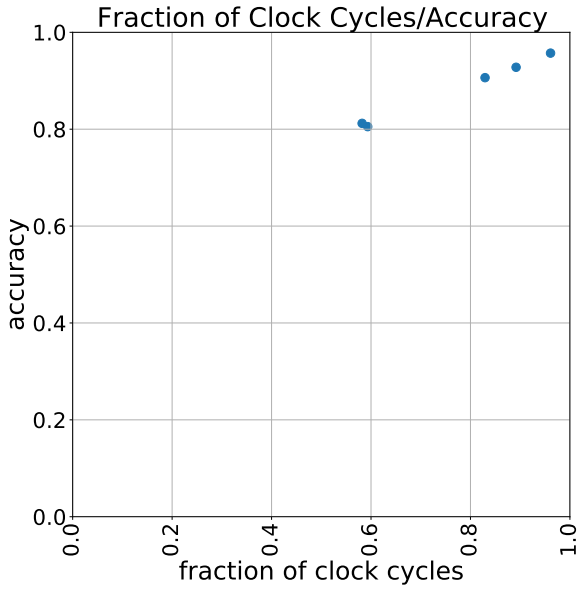


Figure 7.16: Cost/performance of ASVM

while y is the performance of the application during the execution.
The unit of measure of y is the average accuracy of the classifications it emitted during the execution. The cost of the execution x is computed in the following way:

$$x = \frac{C_A}{C_E}$$

where C_A is the number of clock cycles spent by the approximate version of the application and C_E is the number of clock cycles that would be spent by the application in case no power failure occurred at all, i.e., in the situation when all the classifications can be performed using all the features.
The interpretation of this graph is that the points on the top right of the graph describe executions where on average the classifications that were emitted were accurate and costly: when we move to the left, we meet cheaper classification, that have lower accuracy. We can observe that our algorithm can produce savings up to 41%, while the performance degradation remains below 20%.
Another important observation is that we can observe points in the entire region where $x \in (0.5, 1)$. This means that independently from the context the application was able to adapt to the amount of energy available, reaching the goal of consuming the available resources in an optimal way. This is very important because the very beginning of the work was to obtain a program that manages resources in an optimal way (Section 1.2).
After having analysed the accuracy and the cost of the executions of our ap-

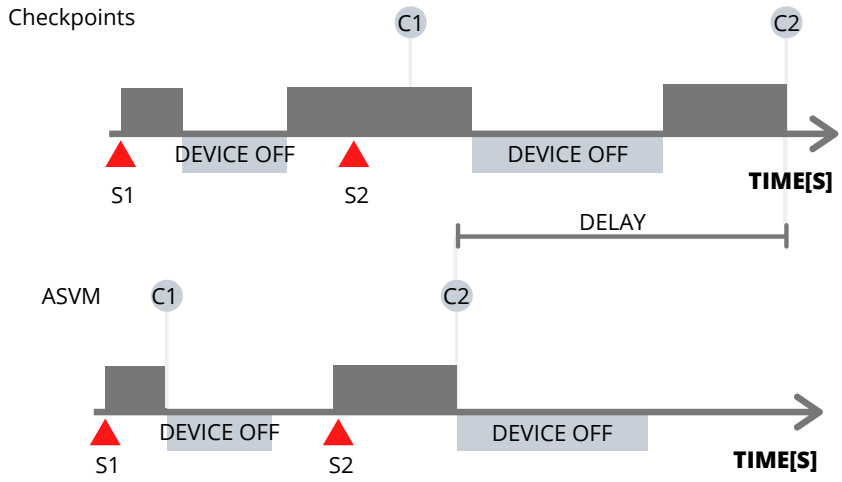


Figure 7.17: Delay in classification

plication, let us focus on its timeliness. Let us have a look at Figure 7.17: the two timelines show how our solution, labelled as ASVM, and the checkpoint-based solution react to the same voltage trace. ASVM emits C1 immediately before a power failure, and so it can immediately start processing S2 when it arrives. On the other hand, when S2 arrives the checkpoint-based solution is still processing S1. The fact that workloads propagate over power failures makes the checkpoint-based solution systematically late with respect to ASVM. Each classification emitted by the checkpoint-based solution has a certain delay (at least 0) with respect to the corresponding classification emitted by ASVM. We are interested in the distribution of the delays according to the voltage trace. The second graph that summarizes the results of our work is shown in Figure 7.18. Each subplot shows the distribution of the delays during an execution, associated to a specific voltage trace.
We measured the delay using the inverse of the sampling frequency as a unit of measure: in Figure 7.17, the delay would be slightly above 1.
Each graph is a histogram, that represents the distribution of the On the x axis there is the delay of classifications, while on y axis there is the number of classifications that suffered from that delay: for example, in trace 0 there are around 150 samples with a delay less than 100 units of measure.
There are two main observations that can be made: first of all, let us have a look at the absolute value of the delays: it ranges from tens to hundreds of samples, which means that ASVM is consistently and systematically far more timely than checkpoint-based solution.
The second observation we can make is that a long-lasting power failure can have a tremendous impact on all the subsequent classification. The most emblematic example is trace 2: a long power failure occurs at the beginning of the

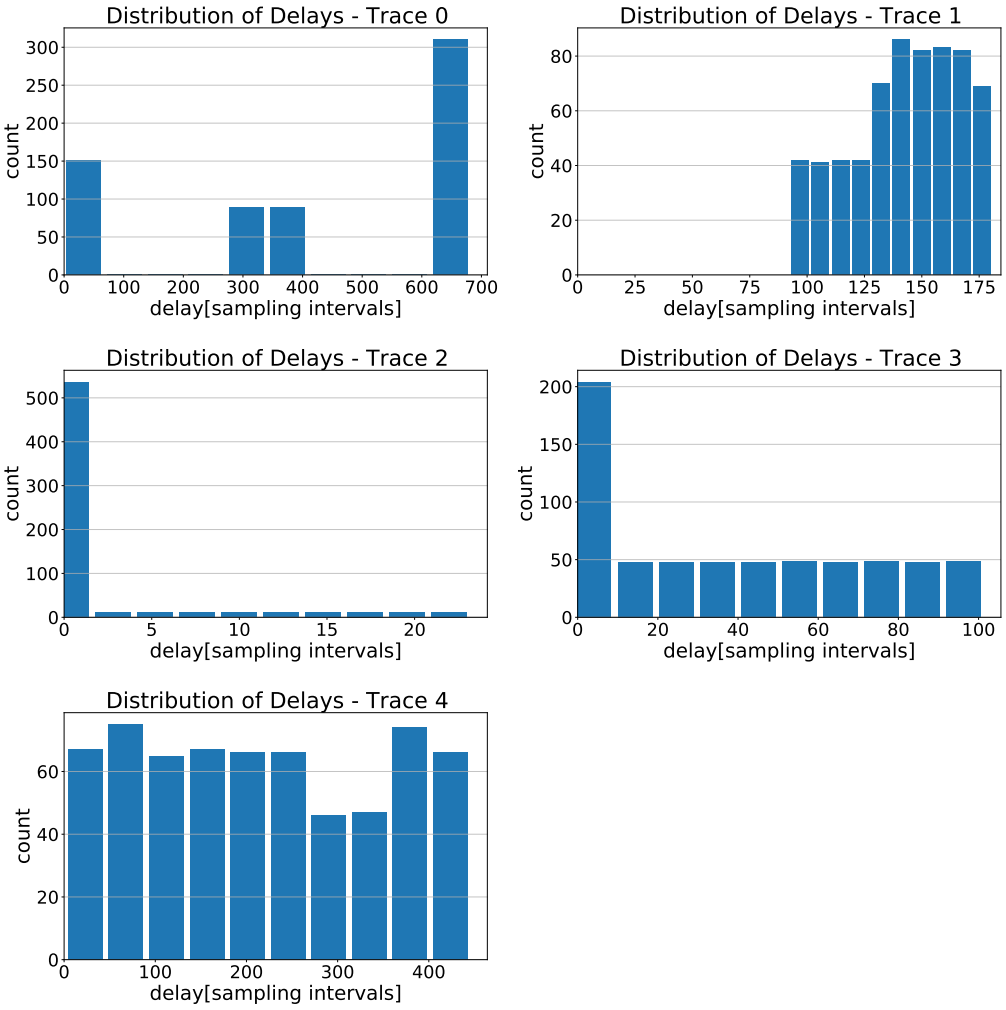


Figure 7.18: Distribution of delays

execution, causing a huge delay for all the samples. Figure 7.17 does also show that delay accumulates over time, and in case of numerous power failures, like in trace 4, the delay of the checkpoint-based solution increases more and more over time. The description of the results comes to its end. In the next chapter, we will describe the conclusions of our efforts and the future lines of research that we have opened up in this work.

Chapter 8

Conclusions and Future Works

The thesis we are defending in this work is that applying Approximate Computing (AC) in the domain of Transiently Powered Computing (TPC) brings huge advantages in terms of energy saved, quality and timeliness of the results produced by TPC applications.
To demonstrate this claim we have designed and implemented a complete application that uses ACT to perform Human Activity Recognition. We have simulated the deployment of the application in a TPC environment, and we have measured that our algorithm does consistently outperform a solution that makes use of a technique in the state of the art to address the problems related to the execution of the application in a TPC context in terms of timeliness in the production of results.
Moreover, we have measured that our solution can cut the computation time and energy necessary to execute the algorithm up to 50%, while lowering the accuracy of the classifications that are produced by 20%.
These advantages are key because the problem of saving energy and the problem of obtaining timely results are arguably the key issues of TPC ([RL14], [MCL17], [HSS17], [Yun+18]).
In Chapter 2 we have shown some of the reasons why Transiently Powered Computing can be a key instrument to enable a pervasive and sustainable adoption of the Internet of Things, and we have presented some of the most prominent challenges of this field.
In Chapter 3 we have argued how Approximate Computing can help to solve some of the most urgent issues of TPC.
In Chapter 4 we have formalized the characteristics that a fruitful combination of these domains should have: in particular, AC should be used to maximize the affordable performance of TPC applications.
In Chapter 5 we have described the characteristics of a proof of concept application that we used to prove our claim: the application uses the technique of

Iterative Refinement to address the problem of Human Activity Recognition.
In Chapter 7 we have measured the performance of the application, proving that it outperformed a state-of-the art solution in terms of timeliness and energy consumption.
In this chapter, we will show that the proof of concept application that we designed represents well the behaviour of all intermittent approximate applications. Moreover, we will highlight the most interesting and challenging research questions that opened up during the work but are too peripheral in the scope of the work itself to be extensively investigated here, and that should as a consequence be explored in future works.

8.1 Threats to Validity

The primary, broad objective from which we started was to demonstrate the feasibility and the usefulness of applying ACTs in the domain of TPC.
We have chosen to explore a single highly representative scenario that is general enough to prove that ACT and TPC go well together. We will now explain why the choices that we have made in this sense do not hinder the generality and the broadness of scope of the work.

8.1.1 Iterative Refinement

As for the technique to use, our decision to focus on Iterative Refinement has been the result of an analysis that considered several factors to assess the fitness of an ACT to the domain of TPC. As explained in Section 5.2, the metrics that we used to classify ACTS are meant to maximize the flexibility of ACT, intended as the variety of contexts when it can be used, to minimize the implementation and testing effort and to maximize the advantage it can provide in a TPC context, i.e., the executions it can unlock.
Iterative Refinement turned out to be one of the best techniques in all the metrics that we have used to quantify the fitness of ACTs, and absolutely the best when we considered the relative importance of the metrics themselves.
The only bias we have introduced is the numeric evaluation we gave to the single ACTs: one could argue about some of the scores we have assigned, but the process that we have followed is scientific and sound.

8.1.2 Energy Harvesting from Human Motion

The second decision we made was to focus on Human Activity Recognition. There are several reasons why HAR is a use case with a very broad scope, that does strongly suggest that AC and TPC work well together. Let us consider the very first constraint, that is to say, the power source that we used: Human Activity Recognition is typically performed using accelerometer data, which means

that where there is HAR there is also the possibility to harvest kinetic energy from human movement. Energy harvesting from human movement has been achieved in several works in the literature ([HYW16], [RNW09]), so we can qualitatively infer that HAR applications can be naturally deployed in TPC contexts without even investing the effort to find a power source.
Energy harvesting from human movement is not a particularly energy-intensive technique. Quantitatively, the throughput of a human-motion-based energy harvester is two orders of magnitude less than a solar harvester, and one order of magnitude less than vibrational harvesters [Bha+16]: the fact that we have been able to use this energy source implies that more power-intensive harvesting techniques can be used as well.

8.1.3 Machine Learning

Finally, the last consideration to make stems from the fact that we have addressed HAR using a machine learning algorithm. The point is that machine learning algorithms show many features that are necessary for algorithms to work well both in AC and TPC, so the evaluation of a TPC, AC machine learning application is representative of a majority of TPC and AC applications.
The main constraint of AC applications is that their output must be intrinsically fuzzy and this is true for machine learning applications (this is not the case for other applications like cryptography).
TPC applications must address some of the main problems of this field (listed in Section 2.3) to be useful. Machine learning algorithms do also address one of the most important issues, the need to save resources: they allow to compress data extracting relevant information, thus saving resources in terms of memory, computation and transmission time (a primary issue of TPC: see Section 2.3.1).
In conclusion, our algorithms has many characteristics of all approximate intermittent applications, and so it is a good example of their behaviour.

8.2 Results

We have shown that the behaviour of ASVM, our application, is a proxy of the behaviour of general AC and TPC applications. Let us summarize these behaviour using the performance metrics that we have measures, comparing our application with a state-of-the-art solution.
In terms of accuracy, ASVM has obtained a reduction below 20% with respect to a checkpoint-based solution for all the scenarios where we tested it, and often the reduction in terms of accuracy has remained below 10%.
This performance loss has been compensated by two key advantages: our application has saved up to 50% of the energetic cost with respect to an exact algorithm, and the results produced by ASVM are systematically more timely than the ones by the state-of-the-art application.
In particular, the absolute delay accumulated by the checkpoint-based applica-

tion solution accounted for up to 400 sampling periods.
In conclusion, we have proven that combining AC and TPC makes it possible to obtain cheap and timely results, at the cost of a moderate performance loss.

8.3 Future Works

In this section, we will explicitly list the open questions that we have briefly touched or mentioned during the previous chapters, and that we did not answer in this work. Some questions are related to making the results in this specific work more concrete and to improve the performance of this very algorithm, delving deep in the specific vertical scope of this thesis. Other questions take a cue from this work and range broader and more general fields.

8.3.1 Physical Implementation

It would be interesting to deploy the algorithm that we have designed and analyzed in this work into a physical device, able to perform HAR in the real world relying solely on energy harvested from human movement. This operation would create two distinct kinds of benefits: first of all it would be the final demonstration that the combination of AC and TPC is feasible, practical and useful.
Moreover, it would contribute to spreading the discipline of TPC in the general public, that is not yet fully aware of the huge potentialities of batteryless computing. In other words, implementing a physical product that makes use of the ideas that we have enunciated in this work will have both direct and indirect scientific and social consequences.

8.3.2 Improving the Algorithm

Although the metrics that measure the performance of the algorithm are already superior to the state of the art, there is still the possibility of speeding up the algorithm and further improving its effectiveness.
In particular, one major opportunity to make it better would be to change the shape of the map from the number of clock cycles invested to the number of features available, showed in Figure 7.15.

In the current implementation this function has several steps, which means that there are moments in which investing some extra clock cycle in the classification does not bring an immediate benefit, and to the energy spent may be lost. The reason why the function has this shape is the fact that one major step to process the data is to perform a Fast Fourier Transform of the window: therefore, every clock cycle invested in the FFT does not bring an immediate gain, but a huge number of features is unlocked when the computation of the FFT terminates.
his behaviour can be changed thanks to the Sliding Fast Fourier Transform [20a]. SFFT is a version of FFT especially fit for signals with a long duration in time.

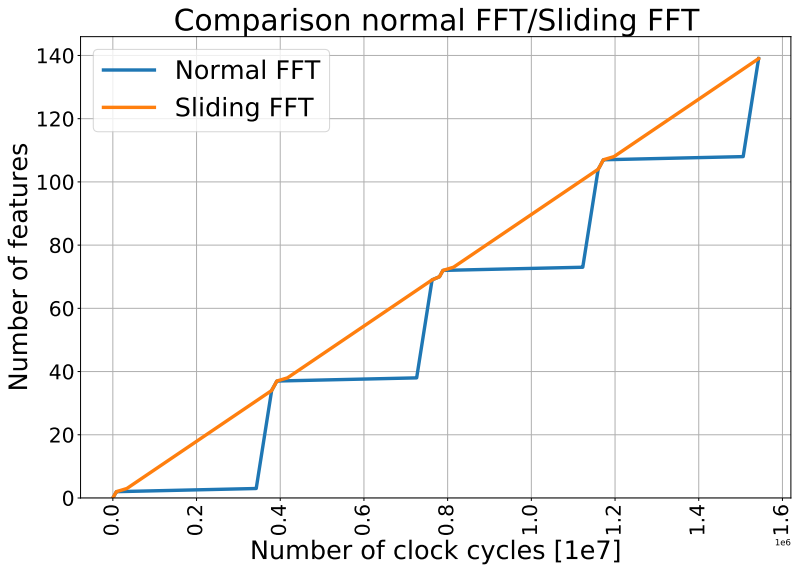


Figure 8.1: Features available as a function of the number of clock cycles invested

It has the property that the SFFT of the window from S_t to S_{t+T} can be cheaply computed from the SFFT of the window from S_{t-1} to S_{t+T-1}. Moreover, SFFT yields one coefficient at a time: this behaviour would impact the shape of the map from the number of clock cycles to the available number of features in the way showed in Figure 8.1. It is possible to observe that the number of features available using SFFT is always above the number of features that are available with a conventional FFT when the same number of clock cycles is invested.
As a consequence, the classifications emitted by the algorithm would be more accurate while maintaining the same cost.
While in principle sliding FFT could provide huge benefits, in practice it would require a substantial implementation effort.
In this work, we had the technical constraint to represent numbers using reduced precision (see Section 5.5): as a consequence, the computation of the vast majority of FFT coefficients becomes unstable.
Therefore, it is necessary to periodically retune the computation or to adapt the algorithm for a fixed-point utilization.

8.3.3 General-Purpose Performance Prediction

The theoretical analysis of the algorithm that we have carried out before the implementation is the crucial point of this work for one main reason, that is to say it provides probabilistic guarantees on the quality of the result that is produced by an approximated version of an algorithm. This is extremely important

because it makes it possible to deploy an application based on the algorithm being sure that it will not underperform and that it will produce results that are consistently good and predictable.
In other words, it is the only theoretical guarantee on the performance of the application. As a consequence, it would be extremely interesting to extend this kind of technique to other classes of algorithms.
This operation can be arbitrarily complex and general: while it is probably feasible to predict the accuracy of SVM that use the kernel trick, it would be far more complex to extend the analysis to nonlinear classifiers.
It would also be interesting to further extend the scope of the analysis by abandoning the field of machine learning algorithms no matter how general it is, and design a methodology to analyze the output of arbitrary applications from a probabilistic point of view.

Chapter 9

Appendix 1

In this Appendix we will provide the derivation of the mapping from number of used features to expected accuracy shown in Section . The derivations will be exposed in the same order as Section .

9.1 Binary Classification, Independent Coefficients

Let $\{\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{x_3}...\boldsymbol{x_m}\}$ be a dataset composed by m vectors.
Let each sample

$$\boldsymbol{x_i} = [c_1 i, c_2 i, c_3 i...c_t i]$$

be a t-dimensional vector.
Let us assume that each coordinate c_i is drawn from a Gaussian distribution C_i

$$C_i \sim N(0, \sigma_i^2)$$

Let us assume that all the C_i are mutually independent.
Let

$$\boldsymbol{w} = [k_1, k_2, k_3...k_t]$$

be the hyperplane that is used by the linear model to classify the samples: for the sake of simplicity, let us assume that

$$k_j = 1 \; \forall j \in \{1..t\}$$

We will demonstrate that this assumption does not affect the generality of the analysis.
In linear models, the classification of a sample $\boldsymbol{x_i}$ is determined by the quantity

$$S_i = \boldsymbol{w}^T \boldsymbol{x_i}$$

where $\boldsymbol{w}^T \boldsymbol{x_i}$ is a normal inner product. Let us indicate as c_{ji} the j-th coefficient of the i-th sample which can be written as

$$S_i = \sum_{j \in \{1...t\}} c_{ji} w_j$$

S_i is distributed as a Gaussian, because it is the sum of independent random normal variables.

$$S_i \sim N(a, b)$$

The parameters a and b of the distribution can be easily obtained by applying standard probability rules [20i]. In particular

$$a = \sum_{i \in \{1..t\}\}} 0 = 0$$

and

$$b = \sum_{j \in \{1..t\}} \sigma_j^2 k_j$$

This equality is the first reason why the value of the coefficients k_j is not important: as a matter of fact, they only appear multiplying C_j^2. Therefore, every coefficient k_j can be normalized to 1 if the variance of the corresponding C_j^2 is appropriately scaled.
Now that we know the distribution of S_i, we will produce an approximate version of this quantity, which is less costly to compute than the exact one, and we will study the probabilistic bound between the two quantities.
In particular, the approximate version of S_i is

$$S_{pi} = \sum_{j \in \{1...p\}} c_{ji} w_j$$

when p=t, $S_{pi} = S_i$. In the other cases, S_{pi} is cheaper to compute than S_i. This consideration is particularly interesting when the features c_{ji} of point i are not immediately available, and involve extra computations to be extracted. We will come back to this point in the next chapter.
The classification of the i-th vector can be computed as

$$class_i = sgn(\sum_{j \in \{1...t\}} c_{ji} w_j)$$

an approximate version of this classification can be computed as

$$class_{pi} = sgn(\sum_{j \in \{1...p\}} c_{ji} w_j)$$

that is to say using only the first p coordinates of the vector. We will call $class_{pi}$ the p-partial classification of the i-th vector. We are interested in computing

the probability that a p-partial classification is coherent with its corresponding t-partial classification, i.e., with the exact classification, as a function of p.

$$P(p-coherence) = P(class_{pi} = class_i)$$

To compute $P(p-coherence)$, we will introduce an auxiliary variable, to easily represent the influence of the coordinates that are excluded from the p-partial classification.

$$R_{pi} = \sum_{j \in \{p+1 \ldots t\}} c_{ji} w_j$$

p-coherence is enforced if the following event takes place

$$S_{pi} \geq -R_{pi}$$

Let

$$F_{S_{pi}}(z) = P(S_{pi} \leq z)$$

$$f_{S_{pi}}(z) = \frac{d}{dz} P(S_{pi} \leq z)$$

$$F_{R_{pi}}(z) = P(R_{pi} \leq z)$$

$$f_{R_{pi}}(z) = \frac{d}{dz} R(S_{pi} \leq z)$$

the probability of p-coherence can be written as

$$P(p{-}coherence) = \int_{d=-\infty}^{0} P(S_{pi} = d \wedge R_{pi} \geq -d) dd + \int_{d=0}^{+\infty} P(S_{pi} = d \wedge R_{pi} \leq -d) dd$$

this formulation is quite general, and it does not need any assumption on the shapes of S_{pi} or R_{pi} in order to work. However, under the assumption that c_{ij} are normally distributed, and independent it is possible to find a more explicit expression. Starting from the fact that the Normal distribution is symmetric, we have that

$$P(S_{pi} = d \wedge R_{pi} \geq -d) = P(S_{pi} = d \wedge R_{pi} \leq -d)$$

and so

$$P(p-coherence) = 2 \int_{d=0}^{+\infty} P(S_{pi} = d) P(R_{pi} \geq -d | S_{pi} = d) dd$$

Thanks to the independence of S_{pi} and R_{pi}

$$P(R_{pi} \leq d | S_{pi} = d) = P(R_{pi} \leq d)$$

so

$$P(p-coherence) = 2 \int_{d=0}^{+\infty} P(S_{pi} = d) P(R_{pi} \geq -d) dd =$$

$$= 2 \int_{d=0}^{+\infty} f_{S_{pi}}(d)(1 - F_{R_{pi}}(d))dd$$

This integral can be computed numerically, and it is very cheap to obtain: as a matter of fact, it is the integral along one single dimension of the product of two simple, unrelated functions.
We will test the correctness of this computation in the next chapter. In the next sections we will drop some of the assumptions we used, in order to obtain a more general result.

9.2 Multiclass Classification, Independent Coefficients

We will analyze the case of multiclass classification using the technique of One-versus-Rest 5.3.1. We recall that using this technique one can classify a vector $\mathbf{x}$ by measuring its signed distance from each of the hyperplances that correspond to each class, and assign it to the class whose hyperplane it is the furthest away. Formally, let

$$\{C_1..C_c\}$$

be the classes and

$$H_l = [h_{l1}, h_{l2}, h_{l3}...h_{lt}]$$

the hyperplane corresponding to class l. To simplify the notation with respect to the previous case, let

$$\boldsymbol{x} = [c_1, c_2...c_t]$$

a sample, whose coordinates

$$c_i \sim N(0, \mu_i)$$

Therefore, the distance of x_i from hyperplane H_l is

$$Fd_l = \sum_{j \in \{1..t\}} h_{lj} c_j$$

The index of the class to which the x_i belong is

$$class_i = argmax_{\ l}\ Fd_l$$

We will follow an idea similar to the one developed in Section 9.1: we will use the first p terms of the sum Fd_l instead of the full sum. Let

$$D_{lp} = \sum_{j \in \{1..p\}} h_{lj} c_j$$

the partial distance from vector $\boldsymbol{x}$ and hyperplane H_l using the first p coordinates.

Since it is the sum of independent gaussians, D_{lp} is gaussian as well. In particular,

$$D_{lp} \sim N(\mu_{lp}, \sigma^2_{lp})$$

$$\mu_{lp} = 0$$

$$\sigma^2_{lp} = \sum_{i=1}^{p} h^2_{lj} Var[c_i]$$

So, the vector

$$D_p = [D_{1p}, D_{2p}, D_{3p}..D_{cp}]$$

of the distances from x_i to hyperplanes 1 to c is a Gaussian vector.

$$D_p \sim N(\mu d_p, \Sigma d_p)$$

it is possible to easily compute that

$$\mu d_p = \mathbf{0}$$

As for Σd_p, the computation of the covariance matrix needs some extra step. We will compute the entries Σd^{ij}_p of the covariance matrix using the definition of covariance [20j].

$$\Sigma d^{ij}_p = \sum_{k=1}^{p} h_{ik} h_{jk} Var[c_k]$$

At this point, we know the distribution of the distances of a vector from each hyperplane, and the probabilistic relation that links the distances. Starting from this result, we will model the partial advantage of a classifier e, that is to say the difference from its distance and the distance of the other classifiers.

$$Ad_{ep} = [D_{ep} - D_{1p}, D_{ep} - D_{2p}..D_{ep} - D_{cp}]$$

Every entry of Ad_{ep} is gaussian because it is the difference of two gaussians. It holds true that

$$Ad_{ep} \sim N(\mu a_p, \Sigma a_p)$$

as the mean of both the two gaussians D_{ep} and D_{ip} is 0,

$$\mu a_p = \mathbf{0}$$

It is possible to compute the entries Σa^{ij}_p of the covariance matrix by using the definition of covariance.

$$\Sigma a^{ij}_{ep} = Var[D_{ep}] + Cov[D_{ip}, D_{jp}] - Cov[D_{ep}, D_{jp}] - Cov[D_{ip}, D_{ep}]$$

So, we now know everything about the probability that any classifier e is further from a sample with respect to other classifiers when only p coordinates are used. We are interested in the probability that a partial classification, performed using the first p coordinates, is coherent with the full classification performed using

all the t coordinates.
To compute this quantity, we need to compute the probability distribution of the correction, i.e, the partial distance computed using the last t-p coordinates. The steps to compute its distribution are similar to the steps to compute the partial advantage. Let C_{lp} the correction of the distance from classifier l, thanks to the last t-p coordinates

$$C_{lp} = \sum_{j \in \{p+1..t\}} h_{lj} c_j$$

C_{lp} is the sum of independent gaussians, so it is gaussian as well.

$$C_{lp} \sim N(\mu c_{lp}, \sigma c_{lp}^2)$$

$$\mu c_{lp} = 0$$

$$\sigma c_{lp}^2 = \sum_{i=p+1}^{t} h_{lj}^2 Var[c_i]$$

Let C_p be the vector of the corrections created by the last t-p coordinates.

$$C_p = [C_{1p}, C_{2p}, C_{3p}..C_{cp}]$$

C_p is a Gaussian vector.

$$C_p \sim N(\mu c_p, \Sigma c_p)$$

where

$$\mu c_p = \mathbf{0}$$

And the entries Σc_p^{ij} can be computed using the definition of covariance.

$$\Sigma c_p^{ij} = \sum_{k=p+1}^{t} h_{ik} h_{jk} Var[c_k]$$

Now, we know the properties of the corrections. We need to compute the probability distribution of the difference between the corrections to understand the probability that a specific classifier becomes the furthest away from the sample thanks to the corrections.

$$Ac_{ep} = [C_{ep} - C_{1p}, C_{ep} - C_{2p}..C_{ep} - C_{cp}]$$

$$Ac_{ep} \sim N(\mu_{A_c ep}, \Sigma_{A_c ep})$$

where

$$\mu_{A_c ep} = \mathbf{0}$$

the computations to perform to find the entries of $\Sigma_{A_c ep}$ are similar to the ones of Σa_{ep}. The result is

$$\Sigma_{A_c ep}^{ij} = Var[C_{ep}] + Cov[C_{ip}, C_{jp}] - Cov[C_{ep}, C_{jp}] - Cov[C_{ip}, C_{ep}]$$

Now we know all the elements that are necessary to compute the probability that a classification carried out using the first p coordinates is coherent with the classification that will derive using all the t coordinates.
Let C_p the class to which the point to classify belongs according to the evaluation done with the first p coordinates. Let C_f the final classification of the vector, performed using all the t coordinates.
We have that

$$P(C_p = C_f) = \sum_{i=1}^{c} P(C_p = C_i \wedge C_f = C_i)$$

Let us focus on one single term of the sum. The other ones can be computed by symmetry.

$$P(C_p = C_i \wedge C_f = C_i) = P(C_p = C_i)P(C_f = C_i | C_p = C_i)$$

We can compute this probability by considering the fact that a classification is maintained if and only if the partial advantage of the winning classifier is greater than the correction due to the last coordinates. Formally

$$P(C_p = C_i)P(C_f = C_i | C_p = C_i) =$$

$$= \int_{\boldsymbol{adv}>\mathbf{0}} P(Ad_{ip} = \boldsymbol{adv})P(Ac_{ep} >= -\boldsymbol{adv} | Ad_{ip} = \boldsymbol{adv}) d\boldsymbol{adv}$$

but Ac_{ep} and Ad_{ip} are independent, so

$$P(Ac_{ep} >= -\boldsymbol{adv} | Ad_{ip} = \boldsymbol{adv}) = P(Ac_{ep} >= -\boldsymbol{adv})$$

In conclusion, we have that

$$P(C_p = C_i \wedge C_f = C_i) = \int_{\boldsymbol{adv}>\mathbf{0}} P(Ad_{ip} = \boldsymbol{adv})P(Ac_{ep} >= -\boldsymbol{adv}) d\boldsymbol{adv} \tag{9.1}$$

This integral can be computed numerically, because the probability density function of Ad_{ip} and the cumulative density function of Ac_{ep} are known.

9.2.1 Multiclass Classification, Correlated Coefficients

The last and most general case we are going to analyze is the situation when the coordinates of the samples to classify are correlated, and more than two classes are present.
The scheme of the demonstration is very similar to the one of the previous case, that is to say Section 9.2.
First of all, we are going to model the probability distribution of the signed distance of a sample $\boldsymbol{x}$ from the separating hyperplanes. Then, we will compute the probability distribution of the differences between the partial distances.
Finally, we will use these last probability distributions to model the probability that a classification remains coherent when more coordinates are used. Let $\boldsymbol{x}$

be the vector to classify.

$$\boldsymbol{x} = [c_1, c_2 ... c_t]$$

Let us assume that

$$\boldsymbol{x} \sim N(\boldsymbol{\mu_x}, \Sigma_x)$$

Where

$$\boldsymbol{\mu_x} = \mathbf{0}_t$$

And the entry of Σ_x on the i-th row and j-th column is indicated with Σ_x^{ij}. Let

$$\{C_1, C_2 .. C_c\}$$

be the classes among which vectors can be classified. Let

$$H_l = [h_{l1}, h_{l2} .. h_{lt}]$$

be the classifier corresponding to the l-th class. Let

$$D_{lp} = \sum_{j=1}^{p} h_{lj} c_j$$

be the partial distance of $\mathbf{x}$ from the l-th hyperplane, computed using the first p coordinates. D_{lp} is a scalar, and it is the sum of gaussians. Therefore, it is gaussian as well.

$$D_{lp} \sim N(\mu_{lp}, \sigma_{lp}^2)$$

where

$$\mu_{lp} = 0$$

because $\boldsymbol{\mu_x} = \mathbf{0_t}$. The variance σ_{lp}^2 can be computed using the definition of variance for a random variable with 0 mean

$$\sigma_{lp}^2 = E[(\sum_{i=1}^{p} h_{li} c_i)(\sum_{j=1}^{p} h_{lj} c_j)] =$$

$$= E[\sum_{i=1}^{p} \sum_{j=1}^{p} h_{li} c_i h_{lj} c_j]$$

the expected value is a linear operator. Therefore, the expression above can be rewritten as

$$\sum_{i=1}^{p} \sum_{j=1}^{p} h_{lj} h_{li} E[c_i c_j]$$

but

$$E[c_i c_j] = Cov[c_i, c_j] = \Sigma_x^{ij}$$

in conclusion,

$$\sigma_{lp}^2 = \sum_{i=1}^{p} \sum_{j=1}^{p} h_{lj} h_{li} \Sigma_x^{ij}$$

that can be explicitly computed.
Let

$$D_p = [D_{1p}, D_{2p}..D_{cp}]$$

be the vector of the partial distances of the sample $\mathbf{x}$ from the hyperplanes.

$$D_p \sim N(\boldsymbol{\mu_p}, \Sigma_p)$$

where

$$\boldsymbol{\mu_p} = \mathbf{0_t}$$

and the entries Σ_p^{ij} of th i-th row and j-th column of the covariance matrix can be computed using the definition of covariance for random variables with 0 mean.

$$\Sigma_p^{ij} = Cov[D_{ip}, D_{jp}] = E[D_{ip}D_{jp}] =$$

$$= E[(\sum_{l=1}^{p} h_{il}c_l)(\sum_{m=1}^{p} h_{jm}c_m)] =$$

$$= E[\sum_{m=1}^{p}\sum_{l=1}^{p} h_{il}c_l h_{jm}c_m] =$$

$$= \sum_{m=1}^{p}\sum_{l=1}^{p} h_{il}h_{jm}E[c_l c_m] =$$

$$= \sum_{m=1}^{p}\sum_{l=1}^{p} h_{il}h_{jm}\Sigma_x^{lm}$$

Now, we know the joint probability distribution of the partial distances from all the classifiers, computed using the first p coordinates. From this probability we can obtain the probability distribution of the partial advantage of classifier e, that is to say the difference between the distance from classifier e and the others. Formally,

$$Ad_{ep} = [D_{ep} - D_{1p}, D_{ep} - D_{2p}..D_{ep} - D_{cp}]$$

Ad_{ep} is a gaussian vector,

$$Ad_{ep} \sim N(\mu_{Adep}, \Sigma_{Adep})$$

where

$$\mu_{Adep} = \mathbf{0}_c$$

and the entries of Σ_{Adep} can be computed using the definition of covariance for random variables with 0 mean

$$\Sigma_{Adep}^{ij} = Cov[D_{ep} - D_{jp}, D_{ep} - D_{ip}] =$$

$$= E[(D_{ep} - D_{jp})(D_{ep} - D_{ip})] =$$

$$= E[D_{jp}D_{ip}] - E[D_{jp}D_{ep}] - E[D_{ep}D_{ip}] + E[D_{ep}D_{ep}] =$$
$$= Cov[D_{jp}, D_{ip}] - Cov[D_{jp}, D_{ep}] - Cov[D_{ep}, D_{ip}] + Var[D_{ep}] =$$
$$= \Sigma_p^{ij} - \Sigma_p^{ej} - \Sigma_p^{ie} + \Sigma_p^{ee}$$

Now we need to compute the probability distribution of the partial distances computed using the last t-p coordinates of the vector. Let us call F_{lp} the distance from the l-th classifier, computed using the last t-p coordinates. We have that

$$F_{lp} = \sum_{i=p+1}^{t} h_{li}c_i$$

let

$$F_p = [F_{1p}, F_{2p}..F_{ct}]$$

be the vector of all the partial distances.

$$F_p \sim N(\boldsymbol{\mu_{Fp}}, \Sigma_{Fp})$$

where

$$\boldsymbol{\mu_{Fp}} = \mathbf{0_c}$$

and

$$\Sigma_{Fp}^{ij} = Cov[F_{ip}, F_{jp}] =$$

$$= E[F_{ip}, F_{jp}] = E[(\sum_{l=p+1}^{t} h_{jl}c_l)(\sum_{m=p+1}^{t} h_{im}c_m)] =$$

$$= E[\sum_{l=p+1}^{t}\sum_{m=p+1}^{t} h_{jl}h_{im}c_m c_l] =$$

$$= \sum_{l=p+1}^{t}\sum_{m=p+1}^{t} h_{jl}h_{im}E[c_m c_l] =$$

$$= \sum_{l=p+1}^{t}\sum_{m=p+1}^{t} h_{jl}h_{im}\Sigma_x^{lm}$$

we have to compute the probability distribution of the advantage gained by classifiers thanks to the last t-p coordinates. Let

$$A_{cep}$$

be the vector that represents the advantage of classifier e. Formally:

$$A_{cep} = [F_{ep} - F_{1p}, F_{ep} - F_{2p}..F_{ep} - F_{cp}] = [A_{cep1}, A_{cep2}..A_{cepc}]$$

it is a gaussian vector:

$$A_{cep} \sim N(\boldsymbol{\mu_{Acep}}, \Sigma_{Acep})$$

where

$$\boldsymbol{\mu_{Acep}} = \mathbf{0_c}$$

and the entries of Σ_{Acep} can be computed as follows:

$$\Sigma^{ij}_{Acep} = Cov[A_{cepi}, A_{cepj}] =$$

$$= Cov[F_{ep} - F_{ip}, F_{ep} - F_{jp}] =$$

$$= E[(F_{ep} - F_{ip})(F_{ep} - F_{jp})] =$$

$$= E[F_{ep}F_{ep}] - E[F_{ep}F_{jp}] - E[F_{ep}F_{ip}] + E[F_{ip}F_{jp}] =$$

$$= Cov[F_{ep}, F_{ep}] - Cov[F_{ep}, F_{jp}] - Cov[F_{ep}, F_{ip}] + Cov[F_{ip}, F_{jp}] =$$

$$\Sigma^{ee}_{Fp} - \Sigma^{ej}_{Fp} - \Sigma^{ei}_{Fp} + \Sigma^{ij}_{Fp}$$

At this point we have the distribution of the partial distance from an arbitrary classifier in two cases: when it is computed using the first p components, and the last t-p components.
We will now focus on the link between them to understand how likely is it that a classification computed using the first p coordinates remains coherent when all the other coordinates are included as well.
To compute this computation, we will concatenate the initial partial advantages and the final partial advantages.

$$V_{ep} = [\boldsymbol{A_{dep}}, \boldsymbol{A_{cep}}] =$$

$$= [A_{dep1}, A_{dep2}..A_{depc}, A_{cep1}, A_{cep2}..A_{cepc}]$$

this vector is gaussian.

$$V_{ep} \sim N(\boldsymbol{\mu_{Vep}}, \Sigma_{Vep})$$

where

$$\boldsymbol{\mu_{Vep}} = \mathbf{0_{2c}}$$

and the entries Σ^{ij}_{Vep} of the covariance matrix are either quantities that we have already computed, or that we can compute from scratch. As a matter of fact, it holds that

$$\Sigma^{ij}_{Vep} = \Sigma^{ij}_{Adep} \quad when \quad i \leq c, j \leq c$$

because these entries represent the covariances between two partial initial distances, that we have computed above.
Similarly,

$$\Sigma^{ij}_{Vfp} = \Sigma^{ij}_{Acfp} \quad when \quad i > c, j > c$$

because the entries with these indexes are the covariances between partial final distances.
We are now interested in computing the mixed entries, i.e., the correlations between A_{depi} and A_{cfpj}. We will once again use the definition of covariance for random variables with zero mean.

$$Cov[A_{depi}, A_{cfpj}] = E[A_{depi}A_{cfpj}] =$$

$$= E[(F_{ep} - F_{ip})(D_{fp} - D_{jp})] =$$
$$= E[F_{ep}D_{fp}] - E[F_{ep}D_{jp}] - E[F_{ip}D_{fp}] + E[F_{ip}D_{jp}] =$$
$$= Cov[F_{ep}, D_{fp}] - Cov[F_{ep}, D_{jp}] - Cov[F_{ip}, D_{fp}] + Cov[F_{ip}, D_{jp}]$$

the four terms of the sum are in the form $Cov[F_{ap}, D_{bp}]$. We will now compute this general term using the definition of covariance for random variables with 0 mean.

$$Cov[F_{ap}, D_{bp}] = E[F_{ap}D_{bp}] =$$
$$= E[\sum_{i=p+i}^{t} h_{ai}c_i \sum_{j=1}^{p} h_{bj}c_j] =$$
$$\sum_{i=p+i}^{t}\sum_{j=1}^{p} h_{ai}h_{bj}E[c_j c_i] =$$
$$= \sum_{i=p+i}^{t}\sum_{j=1}^{p} h_{ai}h_{bj}\Sigma_x^{ij}$$

this computation allows us to compute all the missing values of Σ_{Vfp}^{ij} In particular, we have that

$$Cov[F_{ep}, D_{fp}] = \sum_{i=p+i}^{t}\sum_{j=1}^{p} h_{ei}h_{fj}\Sigma_x^{ij}$$

$$Cov[F_{ep}, D_{jp}] = \sum_{i=p+i}^{t}\sum_{k=1}^{p} h_{ei}h_{jk}\Sigma_x^{ik}$$

$$Cov[F_{ip}, D_{fp}] = \sum_{k=p+i}^{t}\sum_{j=1}^{p} h_{ik}h_{fj}\Sigma_x^{kj}$$

$$Cov[F_{ip}, D_{jp}] = \sum_{k=p+i}^{t}\sum_{l=1}^{p} h_{ik}h_{jl}\Sigma_x^{kl}$$

In conclusion, we have the complete joint distribution of the partial initial distances and the partial final distances.
We need to compute the probability that a partial classification remains coherent when all the coordinates are used. Let

$$PC_p$$

the partial classification computed using the first p coordinates. The probability that a partial classification remains coherent can be computed as:

$$\sum_{C_i \in \{C_1..C_c\}} P(PC_p = C_i \wedge PC_t = C_i)$$

Let us compute one single term of the sum: the other ones are similar, and we will derive them using the same steps.
Let

$$ad_{ep} = [ad_{ep1}, ad_{ep2}..ad_{epc}]$$

be a c-dimensional vector, that represents the values of the partial initial advantages of classifier e computed using the first p coordinates. Let

$$ac_{ep} = [ac_{ep1}, ac_{ep2}..ac_{epc}]$$

be a c-dimensional vector, that represents the values of the partial final distances from classifier e, computed using the last t-p coordinates.
The event $PC_p = C_e$ is equivalent to

$$ad_{epi} > 0 \ \forall i \in 1..c$$

because it means that classifier e is further away than all the other ones. Similarly, $PC_t = C_e$ holds true when the partial final distances are not big enough to take over the advantage gained using the first p coordinates. Formally, the final classification remains coherent if:

$$ad_{epi} > -ac_{ep1} \ \forall i \in 1..c$$

this inequality defines the region where the event of a coherent classification is verified. To compute the probability of this event we need to integrate the probability density function of the joint distribution of the partial initial and final distances over the region. Let $pdf_{Vi}(\boldsymbol{x})$ be the probability density function of V_i evaluated in $\mathbf{x}$. We have that

$$P(PC_p = C_i \wedge PC_t = C_i) = \int_{\boldsymbol{v_e} > \mathbf{0_c}} \int_{\boldsymbol{v_c} > -\boldsymbol{v_e}} pdf_{Vi}([v_e|v_c])dv_e dv_c$$

we can use this result to compute the overall probability that a partial classification is coherent

$$P(PC_p = PC_t) = \sum_{C_i \in \{C_1..C_c\}} \int_{\boldsymbol{v_e} > \mathbf{0_c}} \int_{\boldsymbol{v_c} > -\boldsymbol{v_e}} pdf_{Vi}([v_e|v_c])dv_e dv_c$$

Since pdf_{Vi} is known, this expression can be computed. We have computed the probability to obtain a coherent classification using only a part of the coordinates of a vector.

Bibliography

[FRS01] Karl Pearson F.R.S. "LIII. On lines and planes of closest fit to systems of points in space". In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (1901), pp. 559–572. DOI: 10.1080/14786440109462720. eprint: https://doi.org/10.1080/14786440109462720. URL: https://doi.org/10.1080/14786440109462720.

[Ros58] F. Rosenblatt. "The perceptron: A probabilistic model for information storage and organization in the brain." In: *Psychological Review* 65.6 (1958), pp. 386–408. ISSN: 0033-295X. DOI: 10.1037/h0042519. URL: http://dx.doi.org/10.1037/h0042519.

[BGV92] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. "A Training Algorithm for Optimal Margin Classifiers". In: *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*. COLT '92. Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 1992, pp. 144–152. ISBN: 089791497X. DOI: 10.1145/130385.130401. URL: https://doi.org/10.1145/130385.130401.

[KM99] Kenneth A. Kaufman and Ryszard S. Michalski. "Learning from Inconsistent and Noisy Data: The AQ18 Approach". In: *ISMIS*. 1999.

[Coo02] Ronald Cools. "Advances in multidimensional integration". In: *Journal of Computational and Applied Mathematics* 149.1 (2002). Scientific and Engineering Computations for the 21st Century - Methodologies and Applications Proceedings of the 15th Toyota Conference, pp. 1–12. ISSN: 0377-0427. DOI: https://doi.org/10.1016/S0377-0427(02)00517-4. URL: http://www.sciencedirect.com/science/article/pii/S0377042702005174.

[Dec02] Dennis Decoste. "Anytime Interval-Valued Outputs for Kernel Machines: Fast Support Vector Machine Classification via Distance Geometry". In: (May 2002).

[PLI02] Joanne Peng, Kuk Lee, and Gary Ingersoll. "An Introduction to Logistic Regression Analysis and Reporting". In: *Journal of Educational Research - J EDUC RES* 96 (Sept. 2002), pp. 3–14. DOI: 10.1080/00220670209598786.

[Hil03] W. Hildreth. *Case Studies in Public Budgeting and Financial Management*. Jan. 2003. ISBN: 0-8247-0888-1.

[XMM03a] Fen Xie, Margaret Martonosi, and Sharad Malik. "Compile-Time Dynamic Voltage Scaling Settings: Opportunities and Limits". In: *SIGPLAN Not.* 38.5 (May 2003), pp. 49–62. ISSN: 0362-1340. DOI: 10.1145/780822.781138. URL: https://doi.org/10.1145/780822.781138.

[XMM03b] Fen Xie, Margaret Martonosi, and Sharad Malik. "Compile-time Dynamic Voltage Scaling Settings: Opportunities and Limits". In: *SIGPLAN Not.* 38.5 (May 2003), pp. 49–62. ISSN: 0362-1340. DOI: 10.1145/780822.781138. URL: http://doi.acm.org/10.1145/780822.781138.

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN: 0387310738.

[Pop+07] P. Pop et al. "Scheduling and voltage scaling for energy/reliability trade-offs in fault-tolerant time-triggered embedded systems". In: *2007 5th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. Sept. 2007, pp. 233–238. DOI: 10.1145/1289816.1289873.

[Yeh+07] 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.

[Sch+08] Fabio Schreiber et al. "PERLA: a Data Language for Pervasive Systems". In: Apr. 2008, pp. 282–287. ISBN: 978-0-7695-3113-7. DOI: 10.1109/PERCOM.2008.30.

[YWC08] Jhun-Ying Yang, Jeen-Shing Wang, and Yen-Ping Chen. "Using acceleration measurements for activity recognition: An effective learning algorithm for constructing neural classifiers". In: *Pattern Recognition Letters* 29.16 (2008), pp. 2213–2220. ISSN: 0167-8655. DOI: https://doi.org/10.1016/j.patrec.2008.08.002. URL: http://www.sciencedirect.com/science/article/pii/S0167865508002560.

[RNW09] Edwar Romero, MR Neuman, and RO Warrington. "Kinetic energy harvester for body motion". In: Jan. 2009.

[Saz+09] E. Sazonov et al. "Self-Powered Sensors for Monitoring of Highway Bridges". In: *IEEE Sensors Journal* 9.11 (Nov. 2009), pp. 1422–1429. ISSN: 2379-9153. DOI: 10.1109/JSEN.2009.2019333.

[Wag+09] Kiri L. Wagstaff et al. "Progressive refinement for support vector machines". In: *Data Mining and Knowledge Discovery* 20.1 (Oct. 2009), p. 53. ISSN: 1573-756X. DOI: 10.1007/s10618-009-0149-y. URL: https://doi.org/10.1007/s10618-009-0149-y.

[Yeh+09] Thomas Yeh et al. "Fool Me Twice: Exploring and Exploiting Error Tolerance in Physics-Based Animation". In: *ACM Trans. Graph.* 29 (Jan. 2009).

[Cho11] Mihir Choudhury. "Approximate Logic Circuits: Theory and Applications". PhD thesis. USA, 2011. ISBN: 9781124801759.

[KGE11] P. Kulkarni, P. Gupta, and M. Ercegovac. "Trading Accuracy for Power with an Underdesigned Multiplier Architecture". In: *2011 24th Internatioal Conference on VLSI Design*. Jan. 2011, pp. 346–351. DOI: 10.1109/VLSID.2011.51.

[KWM11] Jennifer R. Kwapisz, Gary M. Weiss, and Samuel A. Moore. "Activity Recognition Using Cell Phone Accelerometers". In: *SIGKDD Explor. Newsl.* 12.2 (Mar. 2011), pp. 74–82. ISSN: 1931-0145. DOI: 10.1145/1964897.1964918. URL: https://doi.org/10.1145/1964897.1964918.

[MS11] R. Mangharam and A. A. Saba. "Anytime Algorithms for GPU Architectures". In: *2011 IEEE 32nd Real-Time Systems Symposium*. Nov. 2011, pp. 47–56. DOI: 10.1109/RTSS.2011.41.

[Par+11] Seonyeong Park et al. "A comprehensive study of energy efficiency and performance of flash-based SSD". In: *Journal of Systems Architecture* 57.4 (2011), pp. 354–365. ISSN: 1383-7621. DOI: https://doi.org/10.1016/j.sysarc.2011.01.005. URL: http://www.sciencedirect.com/science/article/pii/S1383762111000178.

[RSF11] Benjamin Ransford, Jacob Sorber, and Kevin Fu. "Mementos: System Support for Long-running Computation on RFID-scale Devices". In: *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS XVI. Newport Beach, California, USA: ACM, 2011, pp. 159–170. ISBN: 978-1-4503-0266-1. DOI: 10.1145/1950365.1950386. URL: http://doi.acm.org/10.1145/1950365.1950386.

[Sam+11] Adrian Sampson et al. "EnerJ: Approximate Data Types for Safe and General Low-power Computation". In: *Proceedings of the 32Nd ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI '11. San Jose, California, USA: ACM, 2011, pp. 164–174. ISBN: 978-1-4503-0663-8. DOI: 10.1145/1993498.1993518. URL: http://doi.acm.org/10.1145/1993498.1993518.

[Sid+11] 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. Szeged, Hungary: ACM, 2011, pp. 124–134. ISBN: 978-1-4503-0443-6. DOI: 10.1145/2025113.2025133. URL: http://doi.acm.org/10.1145/2025113.2025133.

[YLC11] Jaeyoung Yang, Joonwhan Lee, and Joongmin Choi. "Activity Recognition Based on RFID Object Usage for Smart Mobile Devices". In: *Journal of Computer Science and Technology* 26 (Mar. 2011), pp. 239–246. DOI: 10.1007/s11390-011-9430-9.

[Ang+12a] Davide Anguita et al. "Human Activity Recognition on Smartphones Using a Multiclass Hardware-Friendly Support Vector Machine". In: vol. 7657. Dec. 2012, pp. 216–223. DOI: 10.1007/978-3-642-35395-6_30.

[Ang+12b] Davide Anguita et al. "Human Activity Recognition on Smartphones Using a Multiclass Hardware-Friendly Support Vector Machine". In: *Ambient Assisted Living and Home Care*. Ed. by José Bravo, Ramón Hervás, and Marcela Rodríguez. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 216–223. ISBN: 978-3-642-35395-6.

[Ben+12] S. Benecke et al. "Energy harvesting on its way to a reliable and green micro energy source". In: *2012 Electronics Goes Green 2012+*. Sept. 2012, pp. 1–8.

[Esm+12] H. Esmaeilzadeh et al. "Neural Acceleration for General-Purpose Approximate Programs". In: *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. Dec. 2012, pp. 449–460. DOI: 10.1109/MICRO.2012.48.

[KK12] A. B. Kahng and S. Kang. "Accuracy-configurable adder for approximate arithmetic designs". In: *DAC Design Automation Conference 2012*. June 2012, pp. 820–825. DOI: 10.1145/2228360.2228509.

[Rin+12] Michael F. Ringenburg et al. "Quality of Service Profiling and Autotuning for Energy-Aware Approximate Programming". In: 2012.

[Ven+12] S. Venkataramani et al. "SALSA: Systematic logic synthesis of approximate circuits". In: *DAC Design Automation Conference 2012*. June 2012, pp. 796–801. DOI: 10.1145/2228360.2228504.

[Ang+13] Davide Anguita et al. "A Public Domain Dataset for Human Activity Recognition using Smartphones". In: Jan. 2013.

[CMR13] Michael Carbin, Sasa Misailovic, and Martin C. Rinard. "Verifying Quantitative Reliability for Programs That Execute on Unreliable Hardware". In: *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications*. OOPSLA '13. Indianapolis, Indiana, USA: ACM, 2013, pp. 33–52. ISBN: 978-1-4503-2374-1. DOI: 10.1145/2509136.2509546. URL: http://doi.acm.org/10.1145/2509136.2509546.

[HSD13] Cho-Jui Hsieh, Si Si, and Inderjit Dhillon. "A Divide-and-Conquer Solver for Kernel Support Vector Machines". In: *31st International Conference on Machine Learning, ICML 2014* 1 (Nov. 2013).

[Riz+13] Luca Rizzon et al. "Wireless Sensor Networks for Environmental Monitoring Powered by Microprocessors Heat Dissipation". In: Nov. 2013, 8:1–8:6. ISBN: 9781450324328. DOI: 10.1145/2534208.2534216.

[Sam+13] M. Samadi et al. "SAGE: Self-tuning approximation for graphics engines". In: *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Dec. 2013, pp. 13–24.

[FMS14] Fayçal Ait Aouda, K. Marquet, and G. Salagnac. "Incremental checkpointing of program state to NVRAM for transiently-powered systems". In: *2014 9th International Symposium on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*. May 2014, pp. 1–4. DOI: 10.1109/ReCoSoC.2014.6861359.

[GR14] B. Grigorian and G. Reinman. "Dynamically adaptive and reliable approximate computing using light-weight error analysis". In: *2014 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. July 2014, pp. 248–255. DOI: 10.1109/AHS.2014.6880184.

[Gut+14] Joaquin Gutierrez et al. "Automated Irrigation System Using a Wireless Sensor Network and GPRS Module". In: *Instrumentation and Measurement, IEEE Transactions on* 63 (Jan. 2014), pp. 166–176. DOI: 10.1109/TIM.2013.2276487.

[HSS14] Josiah Hester, Timothy Scott, and Jacob Sorber. "Ekho: Realistic and Repeatable Experimentation for Tiny Energy-Harvesting Sensors". In: *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*. SenSys '14. Memphis, Tennessee: Association for Computing Machinery, 2014, pp. 1–15. ISBN: 9781450331432. DOI: 10.1145/2668332.2668336. URL: https://doi.org/10.1145/2668332.2668336.

[MKS14] Reza Madankan, M. Amin Karami, and Puneet Singla. "Uncertainty Analysis of Energy Harvesting Systems". In: Aug. 2014, V006T10A066. DOI: 10.1115/DETC2014-35480.

[MBJ14] J. S. Miguel, M. Badr, and N. E. Jerger. "Load Value Approximation". In: *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. Dec. 2014, pp. 127–139. DOI: 10.1109/MICRO.2014.22.

[Mit14] Sparsh Mittal. "A survey of architectural techniques for improving cache power efficiency". In: *Sustainable Computing: Informatics and Systems* 4.1 (2014), pp. 33–43. ISSN: 2210-5379. DOI: https://doi.org/10.1016/j.suscom.2013.11.001. URL: http://www.sciencedirect.com/science/article/pii/S2210537913000516.

[Pie+14] Andrea Pietrelli et al. "Wireless Sensor Network Powered by a Terrestrial Microbial Fuel Cell as a Sustainable Land Monitoring Energy System". In: *Sustainability* 6 (Oct. 2014), pp. 7263–7275. DOI: 10.3390/su6107263.

[RL14] Benjamin Ransford and Brandon Lucia. "Nonvolatile Memory is a Broken Time Machine". In: *Proceedings of the Workshop on Memory Systems Performance and Correctness*. MSPC '14. Edinburgh, United Kingdom: ACM, 2014, 5:1–5:3. ISBN: 978-1-4503-2917-0. DOI: 10.1145/2618128.2618136. URL: http://doi.acm.org/10.1145/2618128.2618136.

[Roy+14] Pooja Roy et al. "ASAC: Automatic Sensitivity Analysis for Approximate Computing". In: *Proceedings of the 2014 SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems*. LCTES '14. Edinburgh, United Kingdom: ACM, 2014, pp. 95–104. ISBN: 978-1-4503-2877-7. DOI: 10.1145/2597809.2597812. URL: http://doi.acm.org/10.1145/2597809.2597812.

[Sam+14] Mehrzad Samadi et al. "Paraprox: Pattern-based Approximation for Data Parallel Applications". In: *SIGPLAN Not.* 49.4 (Feb. 2014), pp. 35–50. ISSN: 0362-1340. DOI: 10.1145/2644865.2541948. URL: http://doi.acm.org/10.1145/2644865.2541948.

[Thw+14] B. Thwaites et al. "Rollback-free value prediction with approximate loads". In: *2014 23rd International Conference on Parallel Architecture and Compilation Techniques (PACT)*. Aug. 2014, pp. 493–494. DOI: 10.1145/2628071.2628110.

[Zor+14] Davide Zordan et al. "On the Performance of Lossy Compression Schemes for Energy Constrained Sensor Networking". In: *ACM Trans. Sen. Netw.* 11.1 (Aug. 2014), 15:1–15:34. ISSN: 1550-4859. DOI: 10.1145/2629660. URL: http://doi.acm.org/10.1145/2629660.

[Bal+15] 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: 1943-0671. DOI: 10.1109/LES.2014.2371494.

[Gan+15] S. Ganapathy et al. "Mitigating the impact of faults in unreliable memories for error-resilient applications". In: *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. June 2015, pp. 1–6. DOI: 10.1145/2744769.2744871.

[Goi+15] Ínhigo Goiri et al. "ApproxHadoop: Bringing Approximations to MapReduce Frameworks". In: *Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. Mar. 2015. URL: https://www.microsoft.com/en-us/research/publication/approxhadoop-bringing-approximations-to-mapreduce-frameworks/.

[Khu+15] D. S. Khudia et al. "Rumba: An online quality management system for approximate computing". In: *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*. June 2015, pp. 554–566. DOI: 10.1145/2749469.2750371.

[LR15] Brandon Lucia and Benjamin Ransford. "A Simpler, Safer Programming and Execution Model for Intermittent Systems". In: *SIGPLAN Not.* 50.6 (June 2015), pp. 575–585. ISSN: 0362-1340. DOI: 10.1145/2813885.2737978. URL: http://doi.acm.org/10.1145/2813885.2737978.

[Rah+15a] A. Raha et al. "Quality configurable reduce-and-rank for energy efficient approximate computing". In: *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*. Mar. 2015, pp. 665–670. DOI: 10.7873/DATE.2015.0569.

[Rah+15b] A. Rahimi et al. "Approximate associative memristive memory for energy-efficient GPUs". In: *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*. Mar. 2015, pp. 1497–1502. DOI: 10.7873/DATE.2015.0579.

[Rin+15] Michael Ringenburg et al. "Monitoring and Debugging the Quality of Results in Approximate Programs". In: *SIGPLAN Not.* 50.4 (Mar. 2015), pp. 399–411. ISSN: 0362-1340. DOI: 10.1145/2775054.2694365. URL: http://doi.acm.org/10.1145/2775054.2694365.

[RR15] K. Roy and A. Raghunathan. "Approximate Computing: An Energy-Efficient Computing Technique for Error Resilient Applications". In: *2015 IEEE Computer Society Annual Symposium on VLSI*. July 2015, pp. 473–475. DOI: 10.1109/ISVLSI.2015.130.

[Sch+15] David Schwalb et al. "nvm malloc: Memory Allocation for NVRAM". In: *ADMS@VLDB*. 2015.

[SSE15] Mark Sutherland, Joshua San Miguel, and Natalie Enright Jerger. "Texture Cache Approximation on GPUs". In: June 2015.

[Tal+15] Vamsi Talla et al. "Powering the Next Billion Devices with Wi-Fi". In: *CoRR* abs/1505.06815 (2015). arXiv: 1505.06815. URL: http://arxiv.org/abs/1505.06815.

[Yaz+15] A. Yazdanbakhsh et al. "Axilog: Language support for approximate hardware design". In: *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*. Mar. 2015, pp. 812–817. DOI: 10.7873/DATE.2015.0513.

[AMP16] Henko Aantjes, Amjad Y. Majid, and Przemysław Pawełczak. *A Testbed for Transiently Powered Computers*. 2016. arXiv: 1606.07623 [cs.ET].

[Agr+16] A. Agrawal et al. "Approximate computing: Challenges and opportunities". In: *2016 IEEE International Conference on Rebooting Computing (ICRC)*. Oct. 2016, pp. 1–8. DOI: 10.1109/ICRC.2016.7738674.

[Bha+16] Naveed Anwar Bhatti et al. "Energy Harvesting and Wireless Transfer in Sensor Network Applications: Concepts and Experiences". In: *ACM Trans. Sen. Netw.* 12.3 (Aug. 2016), 24:1–24:40. ISSN: 1550-4859. DOI: 10.1145/2915918. URL: http://doi.acm.org/10.1145/2915918.

[HYW16] Ahmed Haroun, Ichiro Yamada, and S. Warisawa. "Investigation of Kinetic Energy Harvesting from Human Body Motion Activities using Free/Impact Based Micro Electromagnetic Generator". In: *Journal of Diabetes and Cholesterol Metabolism (DCM)* 1 (Nov. 2016), pp. 12–16.

[Hes+16] Josiah Hester et al. "Persistent Clocks for Batteryless Sensing Devices". In: *ACM Trans. Embed. Comput. Syst.* 15.4 (Aug. 2016), 77:1–77:28. ISSN: 1539-9087. DOI: 10.1145/2903140. URL: http://doi.acm.org/10.1145/2903140.

[Khu+16] D. S. Khudia et al. "Quality Control for Approximate Accelerators by Error Prediction". In: *IEEE Design Test* 33.1 (Feb. 2016), pp. 43–50. ISSN: 2168-2364. DOI: 10.1109/MDAT.2015.2501306.

[MS16] D. May and W. Stechele. "Voltage over-scaling in sequential circuits for approximate computing". In: *2016 International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS)*. Apr. 2016, pp. 1–6. DOI: 10.1109/DTIS.2016.7483887.

[Mit16] Sparsh Mittal. "A Survey of Techniques for Approximate Computing". In: *ACM Comput. Surv.* 48.4 (Mar. 2016), 62:1–62:33. ISSN: 0360-0300. DOI: 10.1145/2893356. URL: http://doi.acm.org/10.1145/2893356.

[Moh16] Muhammad Sufyian Mohd Azmi. "Accelerator-Based Human Activity Recognition Using Voting Technique with NBTree and MLP Classifiers". In: vol. 7. Sept. 2016. DOI: 10.18517/ijaseit.7.1.1790.

[San+16] Joshua San Miguel et al. *A Systolic Approach to Deriving Anytime Algorithms for Approximate Computing*. Apr. 2016.

[Vas+16] V. Vassiliadis et al. "Towards automatic significance analysis for approximate computing". In: *2016 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. Mar. 2016, pp. 182–193.

[VRR16] S. Venkataramani, K. Roy, and A. Raghunathan. "Approximate Computing". In: *2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID)*. Jan. 2016, pp. 3–4. DOI: 10.1109/VLSID.2016.128.

[ADL17] Beshr Al Nahas, Simon Duquennoy, and Olaf Landsiedel. "Network Bootstrapping and Leader Election Utilizing the Capture Effect in Low-power Wireless Networks". In: Nov. 2017, pp. 1–2. DOI: 10.1145/3131672.3137002.

[BM17] Naveed Anwar Bhatti and Luca Mottola. "HarvOS: Efficient Code Instrumentation for Transiently-Powered Embedded Sensing". In: *Proceedings of the 16th ACM/IEEE International Conference on Information Processing in Sensor Networks*. IPSN '17. Pittsburgh, Pennsylvania: Association for Computing Machinery, 2017, pp. 209–219. ISBN: 9781450348904. DOI: 10.1145/3055031.3055082. URL: https://doi.org/10.1145/3055031.3055082.

[Gom+17] Andres Gomez et al. "Efficient, Long-Term Logging of Rich Data Sensors Using Transient Sensor Nodes". In: *ACM Trans. Embed. Comput. Syst.* 17.1 (Sept. 2017), 4:1–4:23. ISSN: 1539-9087. DOI: 10.1145/3047499. URL: http://doi.acm.org/10.1145/3047499.

[Hao+17] Han Hao et al. "GHG Emissions from the Production of Lithium-Ion Batteries for Electric Vehicles in China". In: *Sustainability* 9.4 (2017). ISSN: 2071-1050. DOI: 10.3390/su9040504. URL: https://www.mdpi.com/2071-1050/9/4/504.

[HS17] Josiah Hester and Jacob Sorber. "The Future of Sensing is Batteryless, Intermittent, and Awesome". In: *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*. SenSys '17. Delft, Netherlands: ACM, 2017, 21:1–21:6. ISBN: 978-1-4503-5459-2. DOI: 10.1145/3131672.3131699. URL: http://doi.acm.org/10.1145/3131672.3131699.

[HSS17] Josiah Hester, Kevin Storer, and Jacob Sorber. "Timely Execution on Intermittently Powered Batteryless Sensors". In: *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*. SenSys '17. Delft, Netherlands: ACM, 2017, 17:1–17:13. ISBN: 978-1-4503-5459-2. DOI: 10.1145/3131672.3131673. URL: http://doi.acm.org/10.1145/3131672.3131673.

[Luc+17] Brandon Lucia et al. "Intermittent Computing: Challenges and Opportunities". In: *SNAPL*. 2017.

[MCL17] Kiwan Maeng, Alexei Colin, and Brandon Lucia. "Alpaca: Intermittent Execution Without Checkpoints". In: *Proc. ACM Program. Lang.* 1.OOPSLA (Oct. 2017), 96:1–96:30. ISSN: 2475-1421. DOI: 10.1145/3133920. URL: http://doi.acm.org/10.1145/3133920.

[Sen+17] Uvis Senkans et al. "Applications of Energy-Driven Computing: A Transiently-Powered Wireless Cycle Computer". In: Nov. 2017. DOI: 10.1145/3142992.3142993.

[VBM17] Theodoros D. Verykios, Domenico Balsamo, and Geoff V. Merrett. "Exploring energy efficient state retention in transiently-powered computing systems". In: *IDEA League Doctoral School on Transiently Powered Computing (10/11/17)*. 2017. URL: https://eprints.soton.ac.uk/414786/.

[AS18] Samira Ataei and James E. Stine. "A 64 kB Approximate SRAM Architecture for Low-Power Video Applications". In: *IEEE Embed. Syst. Lett.* 10.1 (Mar. 2018), pp. 10–13. ISSN: 1943-0663. DOI: 10.1109/LES.2017.2750140. URL: https://doi.org/10.1109/LES.2017.2750140.

[CL18] Alexei Colin and Brandon Lucia. "Termination Checking and Task Decomposition for Task-Based Intermittent Programs". In: *Proceedings of the 27th International Conference on Compiler Construction*. CC 2018. Vienna, Austria: Association for Computing Machinery, 2018, pp. 116–127. ISBN: 9781450356442. DOI: 10.1145/3178372.3179525. URL: https://doi.org/10.1145/3178372.3179525.

[18] "Diffusion of solar photovoltaic systems and electric vehicles among Dutch consumers: Implications for the energy transition". In: *Energy Research & Social Science* 46 (2018), pp. 68–85. ISSN: 2214-6296. DOI: https://doi.org/10.1016/j.erss.2018.06.003. URL: http://www.sciencedirect.com/science/article/pii/S2214629618305875%22,%20author%20=%20%22M.J.%20van%20der%20Kam%20and%20A.A.H.%20Meelen%20and%20W.G.J.H.M.%20van%20Sark%20and%20F.%20Alkemade.

[Ion18] Adrian Ionescu. "Chapter 8 - Beyond CMOS: Steep-Slope Devices and Energy Efficient Nanoelectronics". In: *High Mobility Materials for CMOS Applications*. Ed. by Nadine Collaert. Woodhead Publishing Series in Electronic and Optical Materials. Woodhead Publishing, 2018, pp. 281–305. ISBN: 978-0-08-102061-6. DOI: https://doi.org/10.1016/B978-0-08-102061-6.00008-2. URL: http://www.sciencedirect.com/science/article/pii/B9780081020616000082.

[JAD18a] Neal Jackson, Joshua Adkins, and Prabal Dutta. "Reconsidering Batteries in Energy Harvesting Sensing". In: *Proceedings of the 6th International Workshop on Energy Harvesting & Energy-Neutral Sensing Systems*. ENSsys '18. Shenzhen, China: ACM, 2018, pp. 14–18. ISBN: 978-1-4503-6047-0. DOI: 10.1145/3279755.3279757. URL: http://doi.acm.org/10.1145/3279755.3279757.

[JAD18b] Neal Jackson, Joshua Adkins, and Prabal Dutta. "Reconsidering Batteries in Energy Harvesting Sensing". In: *Proceedings of the 6th International Workshop on Energy Harvesting & Energy-Neutral Sensing Systems*. ENSsys '18. Shenzhen, China: ACM, 2018, pp. 14–18. ISBN: 978-1-4503-6047-0. DOI: 10.1145/3279755.3279757. URL: http://doi.acm.org/10.1145/3279755.3279757.

[ML18] Kiwan Maeng and Brandon Lucia. "Adaptive Dynamic Checkpointing for Safe Efficient Intermittent Computing". In: *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*. OSDI'18. Carlsbad, CA, USA: USENIX Association, 2018, pp. 129–144. ISBN: 978-1-931971-47-8. URL: http://dl.acm.org/citation.cfm?id=3291168.3291178.

[Mor+18] T. Moreau et al. "A Taxonomy of General Purpose Approximate Computing Techniques". In: *IEEE Embedded Systems Letters* 10.1 (Mar. 2018), pp. 2–5. ISSN: 1943-0671. DOI: 10.1109/LES.2017.2758679.

[RAP18] Francesco Regazzoni, Cesare Alippi, and Ilia Polian. "Security: The Dark Side of Approximate Computing?" In: *Proceedings of the International Conference on Computer-Aided Design*. ICCAD '18. San Diego, California: Association for Computing Machinery, 2018. ISBN: 9781450359504. DOI: 10.1145/3240765.3243497. URL: https://doi.org/10.1145/3240765.3243497.

[XS18] Siyuan Xu and Benjamin Schäfer. "Toward Self-Tunable Approximate Computing". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* PP (Dec. 2018), pp. 1–12. DOI: 10.1109/TVLSI.2018.2884848.

[YAK18] S. Yesil, I. Akturk, and U. R. Karpuzcu. "Toward Dynamic Precision Scaling". In: *IEEE Micro* 38.4 (July 2018), pp. 30–39. ISSN: 1937-4143. DOI: 10.1109/MM.2018.043191123.

[Yun+18] Kasundefinedm Sinan Yundefinedldundefinedrundefinedm et al. "InK: Reactive Kernel for Tiny Batteryless Sensors". In: *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*. SenSys '18. Shenzhen, China: Association for Computing Machinery, 2018, pp. 41–53. ISBN: 9781450359528. DOI: 10.1145/3274783.3274837. URL: https://doi.org/10.1145/3274783.3274837.

[Ahm+19] Saad Ahmed et al. "The Betrayal of Constant Power x Time: Finding the Missing Joules of Transiently-powered Computers". In: *Proceedings of the 20th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems*. LCTES 2019. Phoenix, AZ, USA: ACM, 2019, pp. 97–109. ISBN: 978-1-4503-6724-0. DOI: 10.1145/3316482.3326348. URL: http://doi.acm.org/10.1145/3316482.3326348.

[Bal+19] D. Balsamo et al. "Energy Harvesting Meets IoT: Fuelling Adoption of Transient Computing in Embedded Systems". In: *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*. Apr. 2019, pp. 413–417. DOI: 10.1109/WF-IoT.2019.8767302.

[Che+19] S. Cherubin et al. "TAFFO: Tuning Assistant for Floating to Fixed point Optimization". In: *IEEE Embedded Systems Letters* (2019), pp. 1–1. ISSN: 1943-0671. DOI: 10.1109/LES.2019.2913774.

[Den+19] B. W. Denkinger et al. "Impact of Memory Voltage Scaling on Accuracy and Resilience of Deep Learning Based Edge Devices". In: *IEEE Design Test* (2019), pp. 1–1. ISSN: 2168-2364. DOI: 10.1109/MDAT.2019.2947282.

[GD19] Shubhangi K. Gawali and Mukund K. Deshmukh. "Energy Autonomy in IoT Technologies". In: *Energy Procedia* 156 (2019). 5th International Conference on Power and Energy Systems Engineering (CPESE 2018), pp. 222–226. ISSN: 1876-6102. DOI: https://doi.org/10.1016/j.egypro.2018.11.132. URL: http://www.sciencedirect.com/science/article/pii/S1876610218310920.

[Ima+19] M. Imani et al. "Resistive CAM Acceleration for Tunable Approximate Computing". In: *IEEE Transactions on Emerging Topics in Computing* 7.2 (Apr. 2019), pp. 271–280. ISSN: 2376-4562. DOI: 10.1109/TETC.2016.2642057.

[ILN19] Bashima Islam, Yubo Luo, and Shahriar Nirjon. *Zygarde: Time-Sensitive On-Device Deep Intelligence on Intermittently-Powered Systems*. May 2019.

[Kri+19] A. S. Krishnan et al. "Secure Intermittent Computing Protocol: Protecting State Across Power Loss". In: *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*. Mar. 2019, pp. 734–739. DOI: 10.23919/DATE.2019.8714997.

[Lan+19] L. Lan et al. "Scaling Up Kernel SVM on Limited Resources: A Low-Rank Linearization Approach". In: *IEEE Transactions on Neural Networks and Learning Systems* 30.2 (Feb. 2019), pp. 369–378. ISSN: 2162-2388. DOI: 10.1109/TNNLS.2018.2838140.

[ML19] Kiwan Maeng and Brandon Lucia. "Supporting Peripherals in Intermittent Systems with Just-in-time Checkpoints". In: *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI 2019. Phoenix, AZ, USA: ACM, 2019, pp. 1101–1116. ISBN: 978-1-4503-6712-7. DOI: 10.1145/3314221.3314613. URL: http://doi.acm.org/10.1145/3314221.3314613.

[Mai+19] Andrea Maioli et al. "On Intermittence Bugs in the Battery-less Internet of Things (WIP Paper)". In: *Proceedings of the 20th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems*. LCTES 2019. Phoenix, AZ, USA: ACM, 2019, pp. 203–207. ISBN: 978-1-4503-6724-0. DOI: 10.1145/3316482.3326346. URL: http://doi.acm.org/10.1145/3316482.3326346.

[OU19a] Y. Ono and K. Usami. "Approximate Computing Technique Using Memoization and Simplified Multiplication". In: *2019 34th International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC)*. June 2019, pp. 1–4. DOI: 10.1109/ITC-CSCC.2019.8793369.

[OU19b] Y. Ono and K. Usami. "Approximate Computing Technique Using Memoization and Simplified Multiplication". In: *2019 34th International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC)*. June 2019, pp. 1–4. DOI: 10.1109/ITC-CSCC.2019.8793369.

[20a] https://www.comm.utoronto.ca/~dimitris/ece431/slidingdft.pdf. 2005 (accessed March 6, 2020).

[20b] https://www.electronicdesign.com/power-management/article/21796369/energy-harvesting-and-wireless-sensor-networks-drive-industrial-applications. 2013 (accessed january 6, 2020).

[20c] https://web.stanford.edu/group/sisl/k12/optimization/MO-unit3-pdfs/3.1introandgraphical.pdf. 2013 (accessed january 6, 2020).

[20d] https://www.machinedesign.com/mechanical-motion-systems/article/21829396/sensor-noise-limits-resolution-when-monitoring-motion. 2009 (accessed january 6, 2020).

[20e] http://ipl.physics.harvard.edu/wp-uploads/2013/03/PS3_Error_Propagation_sp13.pdf. 2009 (accessed january 6, 2020).

[19a] https://eu.mouser.com/applications/energy-harvesting-new-applications/. 2015 (accessed October 12, 2019).

[19b] https://homes.cs.washington.edu/~wysem/publications/react-wax15.pdf. 2015 (accessed October 12, 2019).

[19c] https://www.mckinsey.it/file/7259/download?token=B-JxyzW7. 2017 (accessed October 12, 2019).

[19d] https://www.sagentia.com/files/2015/12/Energy-Harvesting.pdf. 2011 (accessed December 4, 2019).

[19e] https://lora-alliance.org/about-lorawan. 2015 (accessed December 1, 2019).

[19f] http://www.mit.edu/~turitsyn/assets/pubs/Hosseinloo2016hf.pdf. 2016 (accessed December 1, 2019).

[19g] https://www.st.com/en/microcontrollers-microprocessors/stm32h7-series.html. 2016 (accessed December 2, 2019).

[19h] https://corporate.delltechnologies.com/en-us/newsroom/announcements/2011/06/20110628-01.htm. 2011 (accessed December 7, 2019).

[19i] https://www.britannica.com/technology/Moores-law. 2011 (accessed December 7, 2019).

[19j] https://www.mckinsey.com/industries/technology-media-and-telecommunications/our-insights/new-demand-new-markets-what-edge-computing-means-for-hardware-companies. 2017 (accessed December 2, 2019).

[19k] https://www.mckinsey.com/industries/semiconductors/our-insights/the-internet-of-things-sizing-up-the-opportunity. 2015 (accessed December 4, 2019).

[19l] https://www.digikey.com/en/articles/techzone/2012/aug/powering-microcontrollers-with-scavenged-energy. 2018 (accessed December 2, 2019).

[19m] https://www.ey.com/Publication/vwLUAssets/EY_-_Future_of_IoT/$FILE/EY-future-of-lot.pdf. 2017 (accessed December 3, 2019).

[19n] https://www.tudelft.nl/en/technology-transfer/development-innovation/research-exhibition-projects/energy-harvesting/. 2017 (accessed December 3, 2019).

[19o] http://www.ti.com/microcontrollers/msp430-ultra-low-power-mcus/overview.html. 2019 (accessed December 2, 2019).

[19p] http://www.diva-portal.org/smash/get/diva2:1333932/FULLTEXT01.pdf. 2019 (accessed December 3, 2019).

[19q] https://sci.esa.int/web/sci-ft/-/50124-technology-readiness-level. 2019 (accessed December 3, 2019).

[19r] https://www.mckinsey.com/featured-insights/artificial-intelligence/notes-from-the-ai-frontier-applications-and-value-of-deep-learning. 2018 (accessed December 7, 2019).

[19s] https://www.fisica.uniroma2.it/~solare/en/wp-content/uploads/2018/12/Lez_12_Noises.pdf. 2018 (accessed December 7, 2019).

[19t] https://www.tutorialspoint.com/digital_communication/digital_communication_quantization.htm. 2018 (accessed December 7, 2019).

[19u] https://www.weforum.org/agenda/2019/04/how-much-data-is-generated-each-day-cf4bddf29f/. 2019 (accessed December 7, 2019).

[19v] https://hbr.org/resources/pdfs/comm/siemens/Acceleratingtheiot.pdf. 2019 (accessed December 7, 2019).

[19w] https://www.fierceelectronics.com/sensors/what-a-sensor. 2019 (accessed December 7, 2019).

[19x] https://www.bloomberg.com/press-releases/2019-06-28/mobile-robotics-market-expected-to-attain-39-58-billion-globally-by-2026-at-21-5-cagr-allied-market-research. 2019 (accessed December 7, 2019).

[19y] https://www.arm.com/why-arm/custom-socs. 2019 (accessed December 8, 2019).

[20f] https://www.iis.fraunhofer.de/en/ff/kom/iot/embedded-ml.html. 2020 (accessed january 24, 2020).

[20g] https://www.quora.com/What-is-the-relationship-between-machine-learning-and-probability-theory. 2015 (accessed january 24, 2020).

[19z] https://stackoverflow.com/questions/3038999/what-is-the-cost-of-memory-access. 2014 (accessed December 12, 2019).

[19aa] https://wapco.e-ce.uth.gr/2015/papers/SESSION3/WAPCO_3_3.pdf. 2014 (accessed December 12, 2019).

[19ab] https://parsec.cs.princeton.edu/. 2010 (accessed December 26, 2019).

[19ac] https://www.microsoft.com/en-us/research/wp-content/uploads/2016/12/green.pdf. 2016 (accessed December 12, 2019).

[19ad] https://web.archive.org/web/20140305080324/http://glossary.computing.society.informs.org/index.php?page=nature.html. 2006 (accessed December 25, 2019).

[20h] https://www.projectrhea.org/rhea/index.php/Lecture_12_-_Support_Vector_Machine_and_Quadratic_Optimization_Problem_OldKiwi. 2011 (accessed february 2, 2020).

[20i] https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf. 2012 (accessed february 4, 2020).

[20j] http://mathworld.wolfram.com/Covariance.html. 2020 (accessed february 8, 2020).

[20k] https://scikit-learn.org/stable/modules/generated/sklearn.multiclass.OneVsRestClassifier.html. 2017 (accessed february 8, 2020).

[20l] https://scikit-learn.org/stable/modules/generated/sklearn.multiclass.OneVsOneClassifier.html. 2017 (accessed february 8, 2020).

[20m] https://www.scipy.org/. 2020 (accessed February 23, 2020).

[20n] https://processors.wiki.ti.com/index.php/Floating_Point_Optimization. 2018 (accessed February 20, 2020).

[19ae] http://apps.webofknowledge.com/RAMore.do?product=WOS&search_mode=GeneralSearch&SID=D6SkDp21hSVXAGAIBBq&qid=1&ra_mode=more&ra_name=PublicationYear&colName=WOS&viewType=raMore. 2010 (accessed November 15, 2019).

[20o] https://github.com/contiki-ng/mspsim. 2017 (accessed February 24, 2020).

[19af] https://spqr.eecs.umich.edu/papers/ransford-thesis.pdf. 2013 (accessed November 15, 2019).

[20p] https://github.com/fbambusi/thesis. 2019 (accessed February 24, 2020).

[20q] https://stats.stackexchange.com/questions/81715/prove-that-covariance-matrix-is-diagonal-after-pca-transformation. 2014 (accessed February 29, 2020).

[19ag] https://www.enocean.com/fileadmin/redaktion/pdf/white_paper/White_Paper_EnOcean_Cost_of_Batteries.pdf. 2015 (accessed November 14, 2019).

[20r] https://arxiv.org/pdf/1605.02541.pdf. 2017 (accessed February 28, 2020).

[20s] https://www.kaggle.com/machinoai/datasets-hra-lstm. 2017 (accessed February 29, 2020).

[19ah] https://my.eng.utah.edu/~cs7810/pres/14-7810-02.pdf. 2016 (accessed November 16, 2019).

[19ai] https://www.samsung.com/it/smartphones/galaxy-s10/specs/. 2016 (accessed November 16, 2019).

[19aj] https://www.ineltro.ch/media/downloads/SAAItem/45/45958/36e3e7f3-2049-4adb-a2a7-79c654d92915.pdf. 2016 (accessed November 16, 2019).

[19ak] https://eu.mouser.com/ProductDetail/Texas-Instruments/MSP-EXP430G2. 2016 (accessed November 16, 2019).

[19al] `https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf`. 2018 (accessed November 24, 2019).

[20t] `https://parsec.cs.princeton.edu/download.htm`. 2020 (accessed february 13, 2020).

[20u] `archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones`. 2013 (accessed September 10, 2020).

[GBL14] "Saïcha Gerbinet, Sandra Belboom, and Angélique Léonard". ""Life Cycle Analysis (LCA) of photovoltaic panels: A review"". In: *"Renewable and Sustainable Energy Reviews"* "38" ("2014"), "747–753". ISSN: "1364-0321". DOI: "https://doi.org/10.1016/j.rser.2014.07.043". URL: http://www.sciencedirect.com/science/article/pii/S136403211400495X.