Universidade Estadual de Campinas
Instituto de Computação

Roberto Alejandro Hidalgo Castro

A Framework for Context-Aware Approximate Computing

Um framework para computação aproximada sensível ao contexto

CAMPINAS
2018

Roberto Alejandro Hidalgo Castro

A Framework for Context-Aware Approximate Computing

Um framework para computação aproximada sensível ao contexto

Dissertação apresentada ao Instituto de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Thesis presented to the Institute of Computing of the University of Campinas in partial fulfillment of the requirements for the degree of Master in Computer Science.

Supervisor/Orientador: Prof. Dr. Lucas Francisco Wanner

Este exemplar corresponde à versão final da Dissertação defendida por Roberto Alejandro Hidalgo Castro e orientada pelo Prof. Dr. Lucas Francisco Wanner.

CAMPINAS
2018

Agência(s) de fomento e nº(s) de processo(s): Não se aplica.

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

H53f

Hidalgo Castro, Roberto Alejandro, 1992-
A framework for context-aware approximate computing / Roberto Alejandro Hidalgo Castro. – Campinas, SP : [s.n.], 2018.

Orientador: Lucas Francisco Wanner.
Dissertação (mestrado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Computação consciente de energia. 2. Computação aproximada. 3. Framework (Programa de computador). I. Wanner, Lucas Francisco, 1981-. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: Um framework para computação aproximada sensível ao contexto
Palavras-chave em inglês:
Energy-aware computing
Approximate computing
Framework (Computer program)
Área de concentração: Ciência da Computação
Titulação: Mestre em Ciência da Computação
Banca examinadora:
Lucas Francisco Wanner [Orientador]
Rodolfo Jardim de Azevedo
Lucas Mello Schnorr
Data de defesa: 09-05-2018
Programa de Pós-Graduação: Ciência da Computação

Universidade Estadual de Campinas
Instituto de Computação

Roberto Alejandro Hidalgo Castro

A Framework for Context-Aware Approximate Computing

Um framework para computação aproximada sensível ao contexto

Campinas, 09 de maio de 2018

Agradecimentos

I would like to thank my advisor, Lucas Wanner for guiding and supporting me over development of my thesis. For being an excellent researcher, mentor and instructor. Also, I would especially like to thank my family for all the support and constant help that you gave me during my work; in particular, I would like to thank a lot my father Alejandro, my mother Olinda and my sister Maria. Finally, I would like to really thank to my girlfriend Stacy for all the constant encouragement and support during this important step in my life.

Abstract

Approximate computing can considerably improve energy efficiency in applications where an approximate result is enough or by relaxing the need for fully precise operations. However, approximate computing applications typically aren't able to take advantage of the computer context dynamically. By improving the computer's access to context in real-time, approximate applications can get information about the current computer power consumption, take decisions according to previously fixed rules, and use this information to produce a more suitable approximation for the current context.

We built a library that includes a series of functions with different implementations wherein each implementation has a different precision result, and a system service that monitors the computer context, including energy consumption, and according to this context (using specified rules), changes the library implementations used by applications in real-time. Applications using the library can therefore save energy when necessary, without compromising quality of results.

We evaluate our context-aware approximate computing library with applications that are suitable for approximations. For each of these applications, we measured the energy consumption of the computer when they are run using the highest precision implementations of the library (that most energy intensive implementations). Knowing this value, we were able to fix an goal energy consumption value (a percentage of the value previously calculated), and using rules around this value, increase or decrease the precision of the implementations used by an application.

Our results show that in our case studies we are able to trade-off at most of 4% degradation in application quality for up to 62% savings in energy consumption. Furthermore, we fix an energy consumption goal for each application, and the applications were able to adapt at run-time to this goal very closely.

Resumo

Computação aproximada pode melhorar consideravelmente a eficiência energética em aplicações onde um resultado aproximado é suficiente. Neste trabalho, construímos bibliotecas de funções padrão que incluem uma série de funções com diferentes implementações, onde cada implementação tem um resultado de precisão diferente. Desenvolvemos ainda um serviço de sistema que monitora o contexto do computador, incluindo o consumo de energia e, de acordo com esse contexto (usando regras especificadas), altera as implementações de biblioteca usadas pelos aplicativos em tempo real. Dessa forma, o aplicativo produz resultados aproximados, mas aceitáveis, ao mesmo tempo que limita o consumo de energia.

O sistema desenvolvido foi testado com aplicativos que são adequados para aproximações. Para cada uma das aplicações, medimos o consumo de energia do computador quando elas são executadas usando as implementações de maior precisão da biblioteca (as implementações mais consumidoras de energia). Conhecendo esse valor, conseguimos fixar um valor de consumo de energia de meta (uma porcentagem do valor calculado anteriormente) e desenvolvemos regras em torno desse valor, aumentando ou diminuindo a precisão das implementações usadas por um aplicativo.

Os resultados mostram que, em nossos estudos de caso, podemos limitar a degradação máxima de 4% na qualidade de resultados das aplicações para obter até 62% de economia no consumo de energia. Além disso, fixamos uma meta de consumo de energia para cada aplicativo, e os aplicativos foram capazes de se adaptar em tempo de execução a essa meta.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

Approximate computing has emerged as a promising approach to the energy-efficient implementation of digital systems. Approximate computing relies on the ability of many systems to tolerate some loss of quality in the computed result. By relaxing the need for fully precise operations or where an approximate result is sufficient, approximate computing techniques allow considerably improved energy efficiency.

While precision is important for some operations, many applications are fundamentally approximate. Perfect answers are not necessary or even impossible in some domains such as machine learning, speech recognition, computer vision, graphics and physical simulation. In many situations systems waste time, energy, and complexity to provide uniformly precise operation for applications that do not require it. However, This is not a permission for computers to abandon predictability in favor of errors. Abstractions are needed that incorporate approximate operation orderly, these abstractions should lead to treat accuracy as a resource and trading it for resources such as time, space, or energy.

In this work, we apply approximate computing to applications for reducing their quality while obtaining reduced energy consumption. We develop a library with functions that can be used in other applications but we include many implementations of the same function, each of these implementations has certain level of quality and certain level of precision. So, the objective is that a certain application uses these implementations and trades between them, sacrificing some quality but reducing the energy consumption of the application. Our library contains only math function but the idea can be applied to any kind of functions. We made this library to be included in any application as automatically as possible through code parsing, but giving also the opportunity that the user chooses which functions wants to use. We have evaluated this library with applications, getting in most cases good results, up to 62% saving in energy consumption for at most 4 % degradation in application quality.

In addition to this library, we have built a monitoring system that constantly measures the energy consumption of the computer while the application that uses the library is running. This system has a energy goal, and depending whether the computer energy consumption is lower or higher that this value, it sends signals to the application to reduce or increase the precision of the functions used in the library. When the energy is lower than the goal, it means that we can increase the energy consumption, so we are able to have more precision in our functions and a signal is sent to increase the precision (increase the quality). Similarly, if the energy is higher than the goal, we should decrease our energy consumption and to manage that we should decrease the precision (decrease the quality). Using this method, we have evaluated this work with many applications, and verified that the energy consumption of the computer when the applications are running oscillates around the energy goal value. This energy goal value is calculated using the highest value of the energy consumption for a period of time, which is when the application is running using the best quality (the highest possible precision). So, the energy goal is percentage of the highest energy consumption for a period of time and system monitoring measures the energy for the same period of time and compares with it. This percentage can be specified by the user.

This document is organized as follows. In the second chapter, we will make a review of the related works for our work, describing works that were made in the field of approximate computing and context-aware computing. Then in the third chapter, we will describe the system design of the developed library and how this library was implemented together with the monitoring system. applications. In fourth chapter, we will talk about the experiments that were made to evaluate our work. We talk about the experimental setup together with the applications that were evaluated for this work in conjunction with the modifications made. Also, we will present the results that were obtained for the all these applications using this library. And in the last chapter, we will give the conclusions of this work.

Chapter 2

Related work

We have explore related works in two main topics: approximate computing and context-aware computing. In this chapter, we will discuss works that made a contribution in these two topics and are related to our work.

2.1 Approximate Computing

There has been an increased interest in the field of approximate computing. Many of the following works have been using different approaches according to the system in discussion. Some of them focus on a language or compiler level, others use the operating system to make energy trade-off and another works build special hardware architectures.

2.1.1 Compiler Techniques

There is a series of works that use compiler techniques to make the approximation. Some of them focus in creating a special programming language that exploits language constructs and features. EnerJ [21] uses type qualifiers to tell which data can be subject to approximate computation. By making use of these types, approximate variables can be assigned to low-power operations, low-power storage and energy efficient algorithms provided by the user can be used with these types of data. EnerJ was tested with many applications showing its effectiveness, with significant potential energy savings (10%–50%) and very little lost of accuracy.

PetaBricks [2] is a parallel language and compiler, based in having many implementations of multiple algorithms to solve a certain problem. In this language uses algorithmic choice including different automatic parallelization techniques, data distributions, algorithmic parameters, transformations, and blocking. When choosing between methods, the compiler is able to tune a program so it gets optimal efficiency for a desired level of accuracy.

Finally, we have Neural Acceleration for General-Purpose Approximate Programs [7], that uses a learning approach to accelerate approximate programs using an algorithm transformation called the Parrot transformation, these algorithm selects and trains a neural network to mimic a region of imperative code. Since neural networks produce approximate results, the neural acceleration makes use of a programming model that allows programmers to identify code regions suitable for approximation that can produce imprecise but acceptable results. Approximate code regions are faster and more energy efficient than executing the original code. The results show that NPU acceleration provides a speedup of 2.3 and energy savings of 3.0 on average with quality loss of at most 9.6%.

2.1.2 Runtime Techniques

Approximate computing with a programming language may also rely on runtime level mechanism. Green [3] for instance, is a system that enables the approximation expensive functions and loops and operates in two phases. The first phase is the calibration phase that builds a model of the quality of service loss produced by the approximation. This model is used later in the second phase, the operational phase, to make approximation decisions based on constraints specified by the user. This phase also includes an adaptation function that monitors the runtime behavior and changes the approximation decisions. The effectiveness of Green was evaluated implementing a system and language extensions using the Phoenix compiler framework. The experiments touch domains such as graphics, machine learning and signal processing, and show significant improvements in performance and energy consumption.

X-Ware [20] is a computing framework which components used by a certain application can take different characteristics and forms across its lifetime with the goal of adjusting to dynamic requirements. X-Ware explores dynamic choice of implementations for certain system components, having different trade-offs between quality and resource usage; and the change in non-functional characteristics of these components, like speed and energy consumption, due to process and environmental variations.

Eon [24] is a runtime system and programming language for supporting the development of perpetual systems and allows programmers compose programs from components written in C. Eon is an energy-aware programming language, that annotates paths through the program with different energy states. Eon dynamically adapts these states according to predicted energy levels, choosing the flow of execution and adjusting it, maximizing the quality of service under energy constraints.

JouleGuard [12] is a runtime control system that manages approximate applications with system resource usage to provide control guarantees of energy consumption, while maximizing accuracy. JouleGuard was implemented and tested it on three different platforms (mobile, tablet, and server) with eight different approximate applications created from two different frameworks. JouleGuard provides near optimal accuracy, adapts to phases in application workload, and provides better outcomes than application approximation alone. Another advantages is that JouleGuard is general with respect to the applications and systems that it manages, which makes it suitable for a good amount of approximate computing frameworks.

In [23], is proposed a method for moving selection of processing platform in continuous run-time choice. The idea is to detect illegally parked vehicles in urban scenes. For each detector power, time, and accuracy information is measure constantly, so an anomaly measure is assigned based on its trajectory and location compared with learned movement patterns. The scenes with high behavioral anomalies are processed with faster but more power-consumption implementations, but routine periods are processed with power-optimized and less accurate slower versions.

ViRUS [27] is an application runtime support system that swaps between blocks of code of equivalent functionality but with different quality with the goal of energy efficiency. In ViRUS, the operating system adapts quality according to certain energy-aware policies making use of techniques for algorithmic choice in runtime, adapting with minimal intervention and exposing quality information, that can be configured by the developers. The application of ViRUS using polymorphic math library showed upwards of 4% degradation in application quality for a band of upwards of 50% savings in energy. ViRUS is tested for applications such us the Whetstone benchmark, and Blackscholes and Swaptions from the Parsec suite [4], for which library functions represent a significant fraction of application energy and time cost. In this work, we extend the idea of ViRUS to handle context-aware energy trade-offs, and explore it with several approximate applications. Unlike ViRUS, our framework presents a simplified adaptation mechanism with energy goal and is able to run with minimal user intervention. Also, our experiments present a much larger set of apps.

2.1.3 Hardware Design Techniques

Another set of implementations in approximate computing relies on hardware systems that can decrease the application's execution time, as well as save power. Because our work focuses solely on software-level approximations, we will not cover this class of work here.

Various modern processors have large last level caches that consume a good amount of energy. Many identical or similar data values might be cached across different blocks simultaneously in multi-megabyte last level caches which wastes cache capacity. Doppelanger [15] tries to reduce this redundancy effectively, it exploits the fact that a large fraction of cache values exhibit approximate similarity, in other words, values across cache blocks are not identical but are similar. Doppelganger designed a cache: the Doppelganger cache that associates the tags of multiple similar blocks with a single data array to reduce the amount of data stored. The cache achieve reductions of 1.55x in LLC area, 2.55x in dynamic energy and 1.41x leakage energy without harming performance.

Gottscho et al. [8], proposed a fault-tolerant cache architecture that utilizes insights gained from a SOI test chip, this architecture contains static and dynamic variants of power/capacity scaling. The mechanism combines multi-level voltage scaling with power gating of blocks that become faulty at each voltage level. The results showed that the average energy saved by the static variant is 55%, while the dynamic variant saves an average of 69% of energy with respect to caches at 1V.

Finally, DRUM [9], which is a multiplier with an unbiased error distribution that produces lower computational errors in real applications. This is achieved because errors cancel each other out rather than accumulate, as the multiplier is used repeatedly in the computation. The approximate multiplier also enables designers to parameterize it depending on their accuracy and power targets. The work shows power savings of up to 58% when the design is used in applications.

2.2 Context-aware Computing

A system can be called context-aware if it uses context to provide relevant information or services to the user, where relevancy depends on the user's task. In this section, we present applications that use context-aware computing and frameworks that support the context-awareness for other applications.

2.2.1 Practical applications

Context-aware computing include applications in mobile and pervasive applications. One of the most common contexts is user geolocation. For example, we have the Active Badge Location System [28], a system for the location of people in an office environment. The system operates as follows: the staff members use badges that transmit signals to a centralized service giving information about their location. Also, Mobile phones represent useful agents for their owners by detecting and reporting situations that are of interest. An application can be found in PeopleTones [11]. PeopleTones is an application for mobile phones, that contains an algorithm for detecting proximity and techniques for reducing sensor noise and power consumption.

Another application that can fit in user's overall activities is the Digital Family Portrait system [17] that introduces the concept of a digital portrait that gives qualitative visualizations of a family member's daily life. So for example, member families are able to check up on other member families without having to explicitly bother them. In the Community Bar system [25], researchers implemented an awareness tool that periodically took screen-shots of the user's display, so that user's co-workers could know what documents their teammates were working on, and provided a common frame of reference. Madhavapeddy et al. [14], propose audio networking, sound hardware like speakers and microphones for wireless networking. This work demonstrates that context-aware applications can be based on existing hardware using audio networking techniques.

In computer science, authors in [1] formulate an energy-aware real-time task scheduling for DVS-enabled multiprocessor systems as a combinatorial optimization problem. A genetic algorithm is proposed in conjunction with a stochastic evolution algorithm to schedule real-time tasks. A simulation study has been done using real benchmark data to evaluate the performance of the Hybrid Genetic Algorithm in terms of solution quality and efficiency.

In medicine, Tung-Hung Lu at al., [12] has proposed a motion-sensing based management system for smart context awareness rehabilitation health-care by the integration of physiological sensing and feedback coaching. The system does not only provide aspects like the exercise coaching instruction or the balance stability analysis, but also simultaneously shows the user image, exercise skeleton streaming, center of pressure, center of gravity and physiological information.

2.2.2 Application Support Frameworks

Several systems focus in supporting the development of context-aware applications. It's important to make the distinction between different kinds of software support for building context-aware applications. In general, software support for applications can be classified as libraries, frameworks or toolkits. A library is a generalized set of related algorithms. Examples include code for manipulating strings and for performing complex mathematical calculations. Libraries focus exclusively on code reuse. On the other hand, frameworks concentrate more on design reuse by providing a basic structure for a certain class of applications. Frameworks shoulder the central responsibilities in an application but provide ways to customize the framework for specific needs. Toolkits build on frameworks by also offering a large number of reusable components for common functionality.

Schilit presented a system architecture that supported context-aware mobile computing [22]. Schilit's work was focused on making context-aware computing applications possible to build. Schilit's architecture supported the gathering of context about devices and users with three main components in the system: device agents that maintain status and capabilities of devices; user agents that maintain user preferences; and, active maps that maintain the location information of devices and users.

Then, Stick-e notes system, a general framework for supporting people that is not a programmer in building context-aware applications [19]. This work focuses on how to support application designers in actually using the context to perform context-aware behaviors. It provides a general mechanism for indicating what context an application designer wants to use and provides simple semantics for writing rules, that specify what actions to take when a particular combination of context is realized. A group of notes or rules are collected together to form a stick-e document. A context-aware application consists of the document and the stick-e note architecture.

Another framework is CoolTown [5], an infrastructure that supports context-aware applications by representing real world objects, like people, places and devices, with a web page. Each web page dynamically updates itself when it gathers new information about the entity that it represents. This infrastructure supports applications that display context and services to end users. For example, as a user moves throughout an environment, they will see a list of available services for this environment and can execute one of the services.

CALAIS [18], the Context And Location Aware Information Service, is another architecture that was designed to support context-aware applications. This work was performed to solve two problems: the arranged nature of sensor use and the lack of a accurate location information management system. An abstraction was developed to hide the details of sensors from context-aware applications. However, similar to Schilit's architecture, there was very little support to aid developers in adding new sensors to the architecture.

Some applications tried to apply context-aware computing in runtime. CAreDroid [6] is a framework that is designed to disconnect the application logic from the complex adaptation decisions in Android context-aware applications with the same idea as ViRUS, using methods that are sensitive to certain contexts along with the permissible operating ranges under those contexts. The results show that the applications using CAreDroid have at least half lines of code fewer and are 10 times more efficient in execution time.

PowerDial [10] is a system that aids dynamically to adapt the application behavior to execute successfully in response of power fluctuations. PowerDial transforms static configuration parameters from the application into dynamic knobs that can be manipulated to trade-off dynamically the accuracy of the results for reductions in the computational resources. As a result, the application obtains performance improvements and power savings. Experimental results show that PowerDial is able to allow applications to execute dynamically in response of power variations, reducing also the required machines, power consumption and capital costs.

Context-aware computing is a promising approach for exploiting the characteristics of mobile computing like mobility, communication and portability. For example, by combining machine learning and context-aware computing, it is possible to provide services based on the users' usage patterns on the mobile device combined with the context where the user and the mobile interacts. CAMF [26] has designed a Context-Aware Machine learning Framework for Android platform. The framework was demonstrated trough the implementation of an application that monitors applications on an Android device along with its environmental context. The goal of this monitoring to dynamically launch Android applications when then context is appropriate.

In software engineering, there is a concept called context-aware personal agent (CPA) that adapts to the changing contexts of its user. Platys [16] is a software engineering framework that was developed to support the development and execution of CPAs. The modelling of a CPA in the framework is done through cognitive steps to simply the development, and delegates the concerns of context transmission from the users and acquisition from sensors to a middleware. By this way, Platys improves the reusability and experience when the user employs CPAs.

Compared to related work, our context monitoring strategy will focus on parameters for energy management such us power consumption.

2.3 Summary

Table 2.1 shows a classification of the selected Approximate Computing Frameworks. Each framework is classified according to the type of adaptation. We have basically three types: adaptations where the approximation is done at compile level, at runtime level and at hardware level. About the techniques, there is a good variety depending how the approximation is handled with the application. The Programming Language Technique consists in developing a programming language that enables to include approximate implementations inside an application efficiently. The Multiple Code Path Technique consists in using multiple path of approximate implementations to be used in the application, these path can be chosen by compilation parameters. The Functional Adaptation Technique consists in replacing the current function implementations of the application by approximate implementations of the same functions. The Dynamic Choice Technique consists of changing the implementations of functions inside the application dynamically depending of certain rules and the context.

Also, for each of framework the technique applied to make the approximation is presented. Table 2.2 presents a brief description of the context-aware computing frameworks. As we can see context-aware frameworks can cover many study fields and can be applied to many problems. Some of these works have obtained progress in energy saving by cost of quality as well; Virus [27] managed to get saving in energy by 40% for the application

Table 2.1: Approximate Computing Frameworks

Paper	Type of adaptation	Techniques
EnerJ [21]	Compiler	Programming Language and Data Types
Petabricks [2]	Compiler	Programming Language and Multiple Code Path
NPU [7]	Compiler	Functional Adaption
Green [3]	Runtime	Multiple Code Path and Parameters
ViRUS [27]	Runtime	Functional Adaption
X-Ware [20]	Runtime, Hardware	Dynamic Choice
Eon [24]	Runtime, Compiler	Multiple Code Path
JouleGuard [12]	Runtime	Multiple Code Path
Doppelanger [15]	Hardware	Memorization
Gottscho et al. [8]	Hardware	Memory Capacity Scaling
Drum [9]	Hardware	Approximate Multiplier

Table 2.2: Context-Aware Computing Frameworks

Paper	Description
PowerDial [10]	Transforms static configuration parameters into dynamic knobs
CAreDroid [6]	Decouple the application logic from the complex adaptation decisions
Stick-e [19]	Support application designers in actually using the context
Platys [16]	Framework that supports the development and execution of CPAs
CALAIS [18]	Context and location aware information service
CAMF [26]	Context-aware machine learning framework for android platform

Blackscholes and 50% for the application Swaptions just for a cost of at most 4% degradation in quality; Green [3] obtained an energy improvement of 28% by the lost of quality of less than 1% in the application Blackscholes.

Our work fit into the categories of compile time and run time adaptation. Our approximate computing technique is similar to ViRUS [27] and CAreDroid [6], we develop a functional adaptation technique through multiple code paths, but our focus is on automated generation of approximate applications and dynamic control monitoring for variable cost and quality objectives.

Chapter 3

System design and implementation

In this section, we propose an application support system for applications that can handle approximate results with the benefit of receiving savings in energy consumption. In this chapter, we explain how this system has been designed and implemented.

3.1 Broad Overview

Our application support system allows applications to dynamically trade function implementations inside the application and with this trading increasing or decreasing the quality of the results but with the advantage of decreasing or increasing the energy consumption of the application when is being run at the operating system. To allow this change in the function implementations of the target application, our system has a library that has been built containing many implementations of the same function. When possible, this library is linked automatically to the application and the system is able to check the performance of the application at runtime, measure the current energy consumption and depending on this, to trigger instructions to change the function implementations. The goal of this changing is to increase or decrease the energy consumption of the whole application but at the cost of the least lost of quality as possible. In the following parts, we explain how this library is built and linked to the application and how the system controls the performance of the application and alters the quality at runtime.

3.2 Variable quality library design

This library was built with in C language, using C tools like macros to allow a multiple declaration of functions and to change implementations of the functions dynamically. It is important to point that the C Macros were used to implement the library but won't be used directly by the user. As we said, certain function has multiple implementations and to distinguish between each of them, we are using levels. With this library, we should be able to change these levels inside the applications (each level represents a certain grade of quality reduction with purpose of decreasing the energy consumption). To easily change the level of a function, we are using helpers to get/set the quality level. This library will be linked at compilation time to the target application. The code in the figure 3.1

shows how functions with one argument are declared in the library, functions with two arguments are declared in a similar way.

```
#define F_ONE( name, level , rettype , argtype , f ) \
rettype __var__ ##name ##level (argtype arg) { return f(arg); }

#define F_HELPERS( name ) \
void v_ ##name ## _set_qlevel(int lvl); \
int v_ ##name ## _get_qlevel(void); \
int v_ ##name ## _avl_qlevel(void);

#define F_DECLARATIONS_ONE( name, rettype , argtype ) \
extern rettype (*v_##name)( argtype arg ); \
F_HELPERS( name )
```

Figure 3.1: One argument function declaration

Given a set of multiple implementations f_1, f_2, f_3, f_4 of the same function, our objective is to present a wrapper function v_f to the applications. We do this by declaring the function v_f as a function pointer and then dynamically assigning the pointer of the address of one of the function implementations (lets say f_3 for example, 3 is the level of the function) . Now, the function pointer f is lined to one of the implementations. So, if the function pointer is assigned to f_3, when the application calls to v_f, it is translated to the implementation f_3. This process is done by calling to the macro *F_ONE*. Beside this macro, all the declarations needed to use the assigned function are done in the *F_DECLARATIONS_ONE*. Inside this macro, the macro *F_HELPERS* sets the functions to handle level currently assigned (in our example is 3). In the next image 3.2, we can see the macro *F_HELPERS* expanded and what each function inside does. *_set_qlevel* sets the level of the function, changing also the implementation at the line 10. So, each time we want to change the level of the function, we can also call directly to the this function. *_get_qlevel* returns the currently function level and *_avl_qlevel* returns the maximum level permitted for the function.

```
 1 void v_ ##name ## _set_qlevel(int lvl) \
 2 { \
 3         if (lvl < 1) { \
 4                 lvl = 1; \
 5         }; \
 6         if (lvl > max_level) { \
 7                 lvl = max_level; \
 8         }; \
 9         __v_ ##name ## _curr_level = lvl; \
10         v_ ##name =__v_##name[lvl]; \
11 } \
12 int v_ ##name ## _get_qlevel(void) { return __v_ ##name ## _curr_level;} \
13 int v_ ##name ## _avl_qlevel(void) { return max_level; }
```

Figure 3.2: Function helpers

And in the next image 3.3, we can see how this macro looks for the cosine function. So, every function in the library has all these functions adapted for each of them.

```c
void v_cos _set_qlevel(int lvl) \
{ \
        if (lvl < 0) { \
                lvl = 0; \
        }; \
        if (lvl > max_level) { \
                lvl = max_level; \
        }; \
        __v_cos_curr_level = lvl; \
        v_cos =__v_cos[lvl]; \
} \
int v_cos_get_qlevel(void) { return __v_cos_curr_level;} \
int v_cos_avl_qlevel(void) { return max_level; }
```

Figure 3.3: Function helpers for the cosine function

3.2.1 Functions supported

This library will support C math functions but it is not restricted to this kind of functions, the library can also be applied for other functions that are suitable to approximation. These functions contain four basic implementations in the library. These implementations are divided in four main versions for each function: double, float, approximate and faster approximate. We use levels to easily refer to each of these implementations. The following code 3.4 shows how the four implementations of the sin function are assigned to the wrapper function *v_sin* using the previously mentioned macro *F_ONE*. The numbers for each function represent the quality level, the higher the level is the higher the quality is. As these wrappers call directly to each of the functions depending on which level we are, using these wrappers don't represent any overhead.

```c
double (*v_sin)( double arg );

F_ONE(sin, 4, double, double, sin)
F_ONE(sin, 3, double, double, sinf)
F_ONE(sin, 2, double, double, fastsin)
F_ONE(sin, 1, double, double, fastersin)

double (*__v_sin[])(double arg) = {__var__sin4, __var__sin3, __var__sin2,
    __var__sin1};
```

Figure 3.4: Function wrappers of the sin function

The macro *F_ONE* defines the function implementation for each level. We can see this more clearly by expanding these macros in the next code 3.5.

The double approximation is the double precision approximation in the results of the function and it is the one that represents the highest quality implementation. The float approximation is similar to the double approximation but reduces the precision inside all the operations of the function to a float precision. The fast approximation contains a specific algorithm inside the implementation of each function and this algorithm reduces the precision of the result considerably. And finally the faster approximation, similarly to the previous approximation contains a specific algorithm in the implementation but

```c
double (*v_sin)( double arg );

double __var__sin4 (double arg) { return sin(arg); }
double __var__sin3 (double arg) { return sinf(arg); }
double __var__sin2 (double arg) { return fastsin(arg); }
double __var__sin1 (double arg) { return fastersin(arg); }

double (*__v_sin[])(double arg) = {__var__sin4, __var__sin3, __var__sin2,
    __var__sin1};
```

Figure 3.5: Macros expanded for the function wrappers of the sin function

Table 3.1: Functions currently supported by the library

Function	Description	Implementation	Level	Time consumption (Cycles)	Code Memory Size (bytes)
v_sin(x)	Sine function of x	sin	4	772342	6008
		sinf	3	751460	6008
		fastsin	2	430260	808
		fastersin	1	393920	728
v_cos(x)	Cosine function of x	cos	4	505600	5992
		cosf	3	454980	5992
		fastcos	2	442120	1112
		fastercos	1	387970	712
v_tan(x)	Tangent function of x	tan	4	558370	6040
		tanf	3	480180	6040
		fasttan	2	415140	856
		fastertan	1	384270	856
v_exp(x)	Exponential function of x	exp	4	479470	27024
		expf	3	459430	15800
		fastexp	2	433810	1240
		fasterexp	1	403000	1016
v_log(x)	Logarithm function of x	log	4	610820	15832
		logf	3	438740	15832
		fastlog	2	413570	1240
		fasterlog	1	379490	648
v_pow(x)	x raised to the power of y	pow	4	488520	38416
		powf	3	436330	18040
		fastpow	2	419540	1368
		fasterpow	1	391400	1032

reducing the precision of the results even more compared with the previous approximation.

Table 3.1 shows the total time consumption and the code memory size of each of the function implementations that are in the library. The time for each function was measure in cycle in an arm architecture using VarEMU [13]. We can see that for each function, when the implementation has a lower level (lower level of quality) it decreases the time consumption and decreases also the code memory size usage in general. Also, we can see clearly that for each of the functions there is a considerable difference between the first two implementations and the last two, the implementations with special algorithms, both in time consumption and in code memory.

Figure 3.6 shows how the execution time per function is reduced in percentage with respect to the normal execution (the highest quality execution).

3.2.2 Memory Usage

When the presented library linked to a certain target application is being used and a certain level of approximation in the function implementations is assigned, the code memory

Figure 3.6: Relative execution time per function

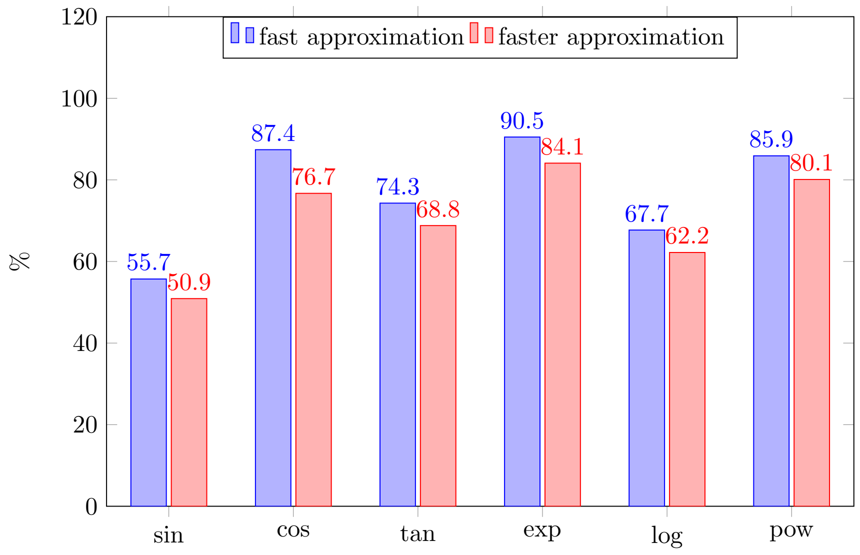


Table 3.2: Memory usage using the library

Application	Code Size (in bytes)
Base Application	400044
Original Application	447188
Heavy Application	452996

size of the source code of the original application changes. Generally, this represents an increase in the source code memory usage. In the table 3.2, we present how expensive is this increase in the memory in general for any application that uses this library in bytes.

In the table, base application represents just a generic application without including any of the function from the library, our original application represents an application that use all the functions inside the library but in their original form and our heavy application represents an application that uses all the function in the library in all their four implementations several times. So, we can clearly see that there is a difference in code size between the base application and the original application when including the library; on the other hand, the difference in code size between the heavy application and original application is not too big compared to the previous difference. As a conclusion, we can say that including the library in the application represents a considerable higher cost when using the function implementations in their two basic forms, the implementations just ind double and float precision, than when using the two approximate implementations.

3.3 Dynamic Adaptation

Now we need a way to modify dynamically this library implementations depending on the situation we are. To achieve this, we need a way to say this to the main program and we need something that manages this changes. In the following parts, we explain how this dynamic adaptation is done.

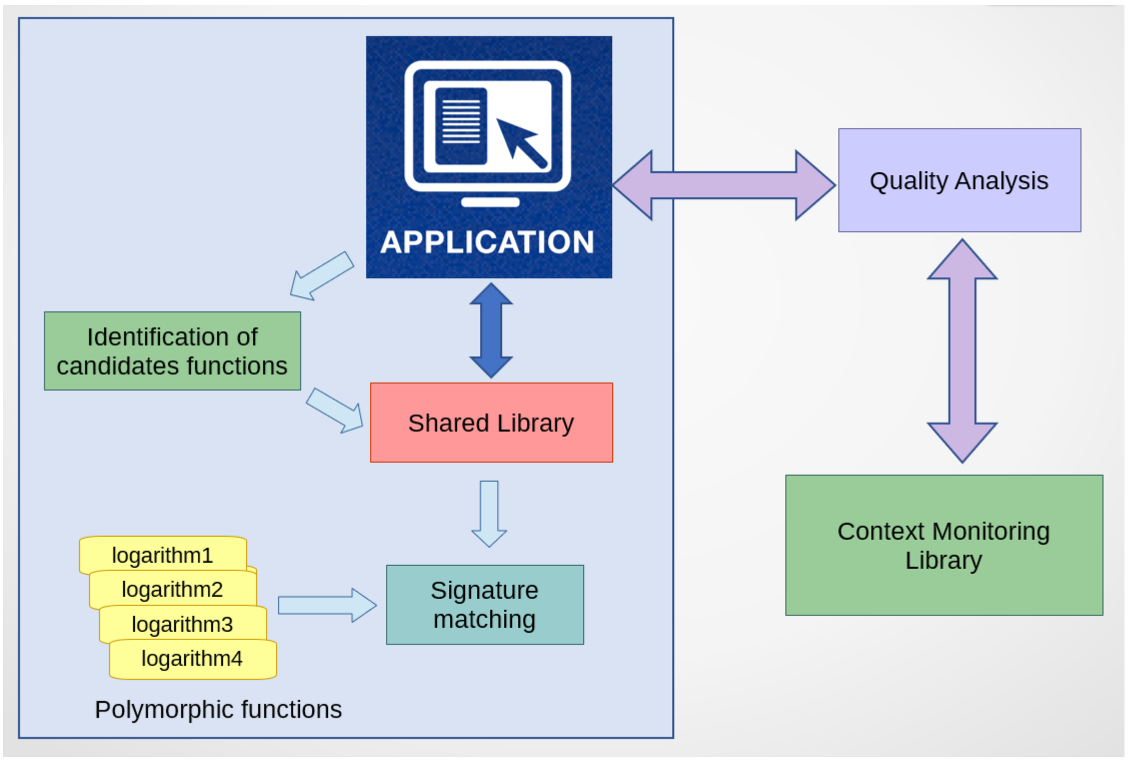


Figure 3.7: System interaction with the target application

3.3.1 Signal Handling

To be able to swap between the function implementations of the library, we use Signal Handling along with the function helpers *_set_qlevel* and *_get_qlevel* that are available for each of the library functions. With these functions, we can define our own policies depending on what we want. We will show an example of how a policy can be made. We can have two basic type of signals depending on how we want to modify the function implementations. In one will increase the quality of the current implementations and in the other one will decrease it (if we are not able to decrease or increase, it will just stay the same). The code 3.8 shows a function that deals with this signal handling, emitting the signal SIGUSR1 to decrease the function level and emitting the signal SIGUSR2 to increase it.

In the image, we can see how to use the functions *_set_qlevel* and *_get_qlevel* to increase the level of all the functions at the same time, we can create more specific rules and changing just some of them depending of our policy.

3.3.2 Automatic Adaptation

Our work provides a script to replace automatically all the functions that can be suitable for an approximation inside the target application. These function will be shown along

```c
void sig_handler(int signo){

  if (signo == SIGUSR1){
    printf("received signal decrease level\n");
    v_pow_set_qlevel(v_pow_get_qlevel() - 1);
    v_exp_set_qlevel(v_exp_get_qlevel() - 1);
    v_log_set_qlevel(v_log_get_qlevel() - 1);
    v_cos_set_qlevel(v_cos_get_qlevel() - 1);
    v_sin_set_qlevel(v_sin_get_qlevel() - 1);
  }

  if (signo == SIGUSR2){
    printf("received signal increase level\n");
    v_pow_set_qlevel(v_pow_get_qlevel() + 1);
    v_exp_set_qlevel(v_exp_get_qlevel() + 1);
    v_log_set_qlevel(v_log_get_qlevel() + 1);
    v_cos_set_qlevel(v_cos_get_qlevel() + 1);
    v_sin_set_qlevel(v_sin_get_qlevel() + 1);
    }
}
```

Figure 3.8: Signal handling function

with their frequency in the application, so that the user can decided to avoid replacing any of them. This script was developed using the tools: Grep and Sed.

3.3.3 Monitoring System

Besides the target application and the library implemented, we have another program that monitors the performance of the application and depending on this, it triggers signals to the application to reduce or increase the quality output, and with this reduce or increase the energy consumption. The monitoring system has an energy goal percentage based on the energy consumption at the highest quality (the default quality of the application). This energy at the highest quality was previously calculated by the monitoring system running the application at the highest quality several times and obtaining the average energy consumption. This system measures the energy consumption and depending on difference between the current energy consumption and the energy consumption goal, it sends signals to the application to change the library function implementations that are used by the application at the moment, changing also the performance of the program. We are working with the signals: SIGUSR1 and SIGUSR2. This step is done periodically by the monitoring system, trying to reach the energy goal at each iteration. This system uses energy counters to estimate the energy consumption.

The Figure 3.3 presents a global review of how the dynamic adaption is done. The shared library is linked by the target application through the signature matching and at run-time when the application is running there is a constant quality measure by the context monitoring application changing the quality of the target application if its needed.

Chapter 4

Experiments

In this chapter, we talk about the methodology that we use for the evaluation of the developed library in the target applications and the results obtained in this work.

4.1 Experimental Setup

In this section, we talk about the environment and parameters for our cases studies that were used to evaluate our proposed system.

4.1.1 VarEMU

VarEMU [13] is framework that is useful for the evaluation of variability-aware software techniques. This framework allows us to emulate variations in power consumption and to adapt to these variations in software in an ARM architecture. Also, users can create virtual machines that feature static and dynamic variations in power consumption through the use of parameters in a power model, this parameters can also be changed dynamically. So, with this framework we are able to quantify and to evaluate the effects of variations on each of our target applications. The virtual machine created trough VareEMU interacts with it through memory mapped registers and VarEMU allows us to measure the energy and execution time by creating a checkpoint for all VarEMU registers.

4.1.2 Parameters for power model

VarEMU supports configurable models for static and dynamic power. For this work we used the default model which is fitted to a Cortex M class ARM chip [13]. In our experiments, we report energy consumption and execution time savings relative to the standard app with no modifications. Energy is given in Joules and execution time in cycles. These numbers are normalized in our experiments.

4.2 Target Applications

In this work, we will modify certain applications, transforming an application into another one that trades the accuracy of the computation in return for reductions in the computational resources. These reductions will translate directly into a reduction in time execution and energy consumption. The applications chosen for our work were selected because all of them present a heavy usage of math functions which is crucial for us, and some of them were also used in the evaluation of other similar works.

4.2.1 Bodytrack

This application is an Intel workload whose goal is tracking a human body with many cameras through an image sequence. This application does an annealed filter for tracking, using the foreground silhouette and edges of the images as features.

Changes

This computer vision application works with an annealed particle filter which uses repetitively sinusoidal functions. Also, this application contains a good amount of logarithm and exponential functions. We trade all these functions with implementations that saves us computer consumption for a reduced quality.

Quality

This application has as an output an image that we are using to measure the quality of the program. We do a root-mean-square deviation over each of the pixels of the image to measure the quality.

Related results

This application has been approximated by PowerDial [10] obtaining a Speedup 0,99 by a loss of quality of 0.839% and also was approximated by JouleGuard [12], obtaining a Speedup 7.38 by a loss of quality of 14.4%. PowerDial identified two parameters (number of annealing layers and number of particles) that were manipulated dynamically for the trade-off and JouleGuard uses Powerdial approximation implementation coordinated with resource usage.

Results

This application has been approximated in the current work and obtained a 16.22% in energy saving and 16.41% in time savings for a quality cost of 1.86% in the metric NRMSE (normalized root mean square error) using the fast approximation (medium quality). On the other side, we obtained 17.71% in energy saving and 17.54% in time saving for a quality cost of 2.14% in the metric NRMSE using the faster approximation, which give us a sightly better result but with a higher quality loss. Figure 4.7 shows the precise image output compared with the approximate image output and we can see that the quality loss is so small that is practically imperceptible.

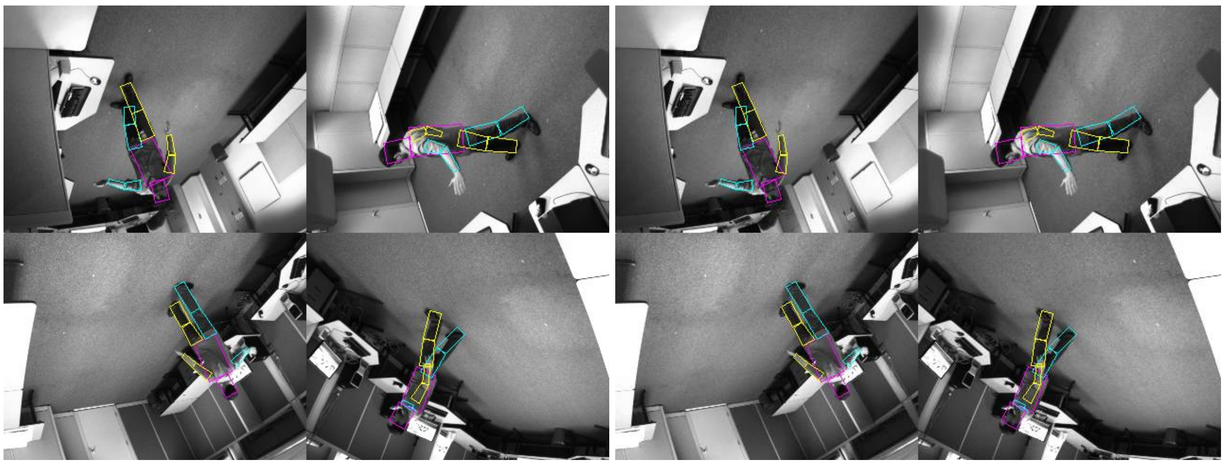


Figure 4.1: Bodytrack: Image precise output vs approximate output

4.2.2 Facesim

This Intel application computes a realistic animation of a modeled face, taking the modeled face with a set of muscle activations and then simulating the physics of the face. The main goal of this application is to create a realistic result.

Changes

This application uses a Newton-Raphson method and a conjugate gradient algorithm to solve the nonlinear and linear system of equations respectively. So, it contains a good amount of math functions like exponentials. We were able to trade these implementations to reduce the consumption but with a lower quality.

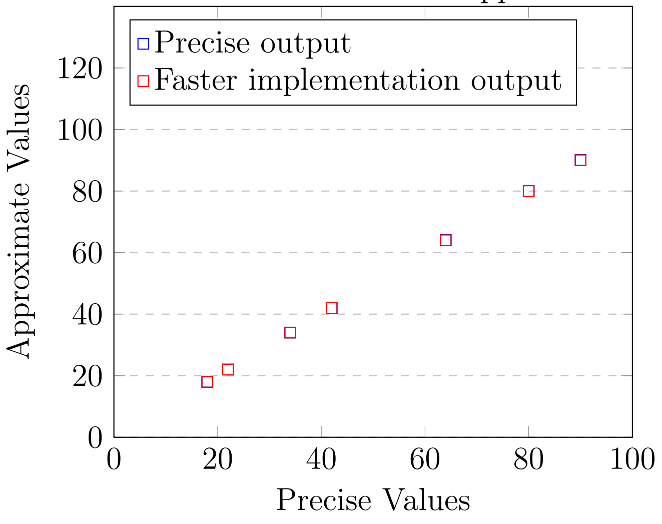
Quality

The program writes the final state of the face mesh to several files, we are using those states to measure the quality. We calculate the quality doing a root-mean-square deviation between the real states and the ones obtained by the approximate run.

Results

This application has been approximated in the current work and didn't obtained basically any energy saving and time saving for a quality cost of zero in the metric NRMSE using the fast approximation (medium quality). Using the faster approximation, we didn't obtained any energy saving and time saving as well for a cost of zero in the metric NRMSE, which shows us that Facesim does not have a heavy used of math functions in the core parts of the application. Figure 4.2 shows how the output values from Susan change when the faster approximation implementation is applied, but in this case we can see that there isn't practically any difference in the values.

Figure 4.2: Facesim: Precise results vs approximate results



4.2.3 Ferret

This application is a configured version of the Ferret toolkit adapted for image similarity search. The main stages of this application are query image segmentation, feature extraction, indexing and ranking. And the main goal is, given a data set and a query image, getting the final set of images that are most similar to the query image.

Changes

As image segmentation and feature extraction are key parts of the applications, there is a good amount of logarithm, exponential and sinusoidal functions that we are modifying as a similar way as the previous applications.

Quality

The programs has as output the measure of similarity of the most similar images for each image query, so this measure will be our measure of quality for the application.

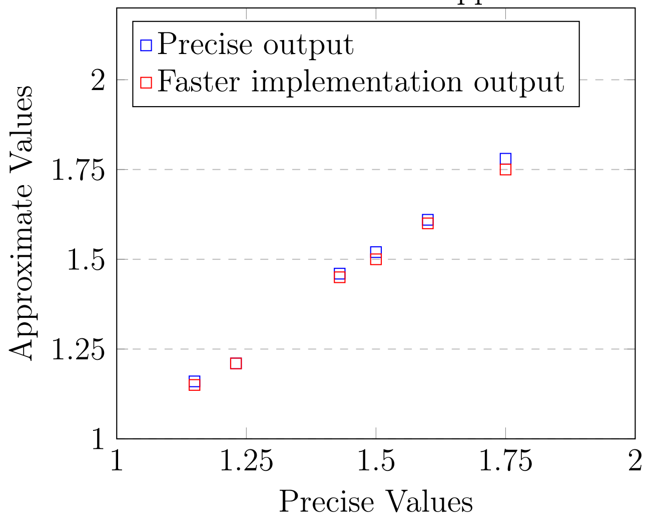
Related results

Ferret has been optimized by JouleGuard [12], obtaining a Speedup 1.24 by a loss of quality of 18.2% trough Loop Perforation in coordination with a system resource usage.

Results

This application has been approximated obtaining a 11.15% in energy saving and 17.62% in time savings for a cost of 1.46% in the metric NRMSE using the fast approximation (medium quality). Furthermore, we got 15.21% in energy saving and 21.22% in time saving for a cost of 2.73% in the metric NRMSE using the faster approximation, which give us a good amount of time and energy saving for a small additional loss of quality, also there is a clear difference in savings between these two approximations which is another advantage. Figure 4.3 shows the precise output values from Ferret compared to the approximate output values when the faster approximation implementation is applied. In the graphic, we can see clearly an slightly difference in the values shown in the graphic. We only chose some of them to represent the whole output.

Figure 4.3: Ferret: Precise results vs approximate results



4.2.4 Vips

The benchmark is based on the the VASARI Image Processing System and includes fundamental image operations such as an affine transformation and a convolution.

Changes

In this application, trigonometric functions are used very often. So, we are changing those implementations as well as other functions like logarithm and exponentials in a similar way as the other applications.

Quality

The programs writes an output image that we are using to measure the quality. We compare each of the pixel of the image obtained by the approximate run with the precise image using a root-mean-square deviation.

Results

This application has been approximated obtaining a 12.10% in energy saving and 12.04% in time savings for a cost of 0.84% in the metric NRMSE using the fast approximation (medium quality). Also, we got 18.23% in energy saving and 18.02% in time saving for a cost of 1.96% in the metric NRMSE using the faster approximation. We obtained here a good amount of savings for a small loss of quality in the first approximation, and a better saving for the second one but with a higher but not to big loss in quality. Figure 4.4 shows the precise image output of Vips compared to the approximate image output.

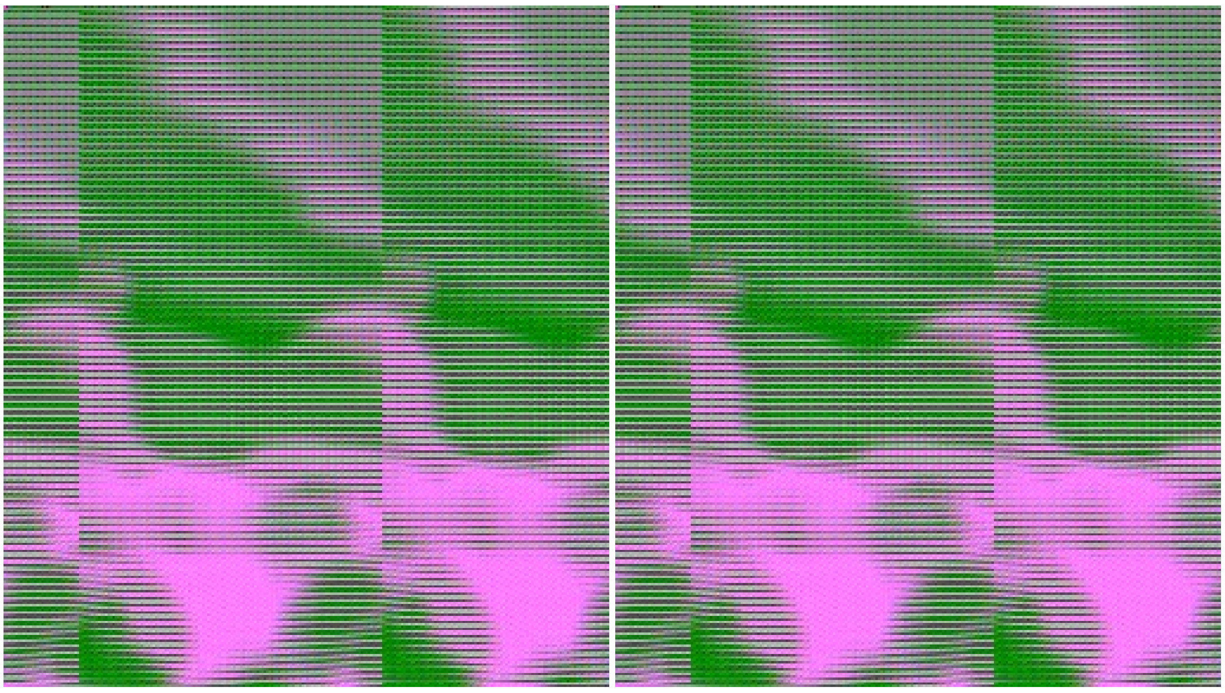


Figure 4.4: Vips: Image precise output vs approximate output

4.2.5 FFT

This benchmark's function is to perform the very known Fast Fourier Transform as well as the inverse transform on an specific data. The data is an array generated randomly by the same application.

Changes

As this is a Fast Fourier Transform implementation, it presents a heavy use of sin and cos functions inside. which is suitable for our work. We replace these functions to evaluate how good the computer consumption improves.

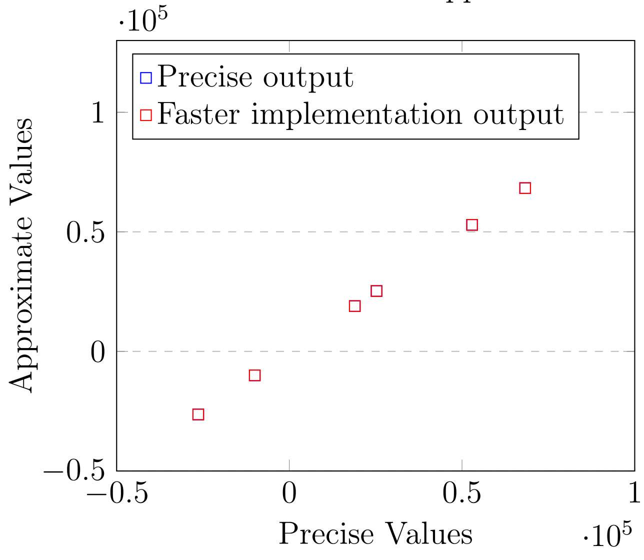
Quality

The output of this application is an array of data that we are using to measure the quality using a root-mean-square deviation calculation.

Results

This application has been approximated obtaining a 61.81% in energy saving and 61.60% in time savings for a cost of 2.54% in the metric NRMSE using the fast approximation (medium quality). Also, we got 62.90% in energy saving and 62.51% in time saving for a cost of 3.61% in the metric NRMSE using the faster approximation. We obtained here a good amount of savings for a small loss of quality in the first approximation, and a better saving for the second one but with a higher loss in quality. Figure **??** shows some of the output values of the FFT application, we can see a slightly difference between the values in graphic, which is also because of the output range of values is big in this graphic.

Figure 4.5: FFT: Precise results vs approximate results



4.2.6 Basicmath

The goal of this application is to perform mathematical calculations that usually don't have dedicated hardware support in embedded processors. Some examples of these calculations are the cubic function solving, integer square root and angle conversions from degrees.

Changes

Inside all the mathematical calculations that this application does, there is a heavy amount of math functions. Using the library we developed, we replace these functions and evaluate how much the computer consumption is saved.

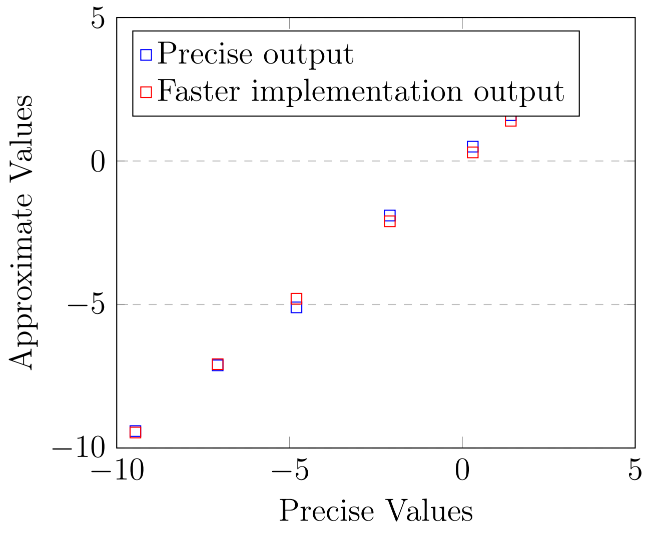
Quality

The output of this application is a set of numbers and we use these numbers to evaluate the application calculating the difference using a root-mean-square deviation.

Results

This application has been approximated obtaining a 38.83% in energy saving and 38.72% in time savings for a cost of 0.12% in the metric NRMSE using the fast approximation (medium quality). Also, we got 42.83% in energy saving and 41.72% in time saving for a cost of 0.23% in the metric NRMSE using the faster approximation. We obtained here a good amount of savings for a small loss of quality in the first approximation, and a better saving for the second one but with a higher but still really small loss in quality. Figure 4.6 shows how the output values from Basicmath varies when the faster approximation implementation of the library is applied, the difference in this graphic are clear.

Figure 4.6: Basicmath: Precise results vs approximate results



4.2.7 Susan

This image recognition application has the function of recognizing corners and edges in Magnetic Resonance Images of the brain. Susan also allows smoothing the image and making adjustments for threshold and brightness.

Changes

Inside this application, there is a heavy use of math functions, specially sinusoidal functions. Our system supports these kind of functions so we were able to replace these implementations.

Quality

The output of Susan is a data image array, we are using those values to measure the quality in the same way as the previous applications using a root-mean-square deviation.

Results

This application has been approximated obtaining a 7.47% in energy saving and 6.82% in time savings for a cost of 0.15% in the metric NRMSE using the fast approximation (medium quality). Also, we got 9.82% in energy saving and 9.47% in time saving for a cost of 0.35% in the metric NRMSE using the faster approximation. We obtained here a good amount of savings for a small loss of quality in the first approximation, and a better saving for the second one but with a higher but still not too big loss in quality. So the reductions weren't too big but the quality loss were really small. Figure **??** shows the precise output values compared with the approximate output values and we can see that the difference is imperceptible to the eye. This output image in particular is the result of an image smoothing.

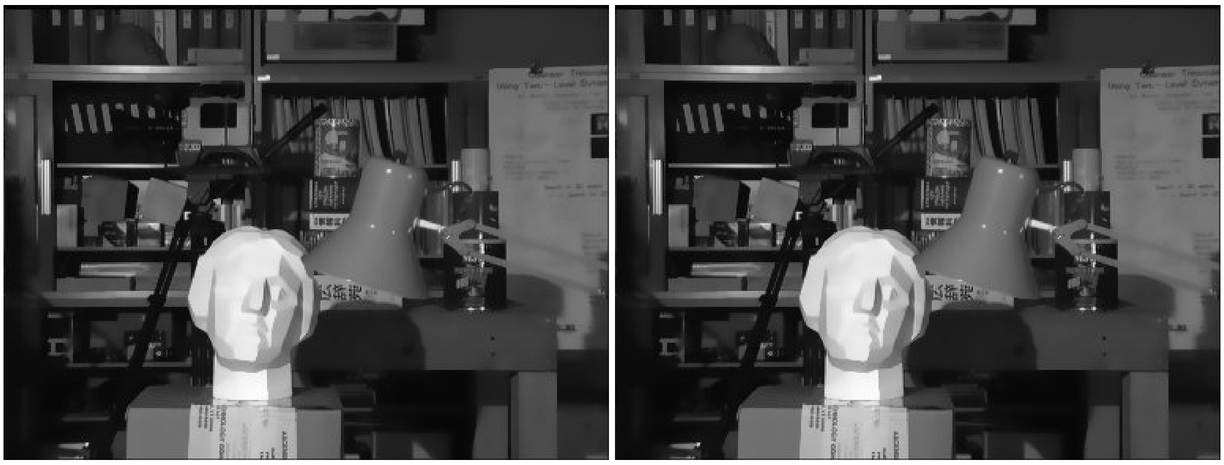


Figure 4.7: Susan: Image precise output vs approximate output

4.3 Sumary of results

In the following table 4.1, we show the results of using library function implementations for each of the target applications, making a trade-off between energy consumption and application quality. We got a reduction in energy consumption and time execution by the cost of some amount of quality:

4.3.1 Quality and Energy

Table 4.1: Approximate Computing Applications

Applications	Versions					
	Medium quality			Low quality		
	time (% savings)	energy (% savings)	quality (% loss)	time (% savings)	energy (% savings)	quality (% loss)
Bodytrack	16.41	16.22	1.86	17.54	17.71	2.14
Facesim	0.00	0.00	0.00	0.00	0.00	0.00
Ferret	17.62	11.15	1.46	21.22	15.21	2.73
Vips	12.04	12.10	0.84	18.02	18.23	1.96
FFT	61.60	61.81	2.54	62.51	62.90	3.61
Basicmatch	38.72	38.83	0.12	41.72	42.83	0.23
Susan	7.47	6.82	0.15	9.47	9.82	0.35

We use the metric NRMSE (normalized root mean square error) to analyze the output quality of each of the applications.
For most of the applications, there is a good percentage of saving in execution time as well as in energy consumption. We got a over 15% in five of the applications we evaluated, without losing more than 4% of their quality output (using NRMSE). In Susan where we got low time savings and energy savings, the quality loss was very low as well, so at least we got some amount of saving for a very low lost in quality. In particular in the case of Facesim, seems that the use of math functions that the system supports does not have too much impact in the whole application performance.

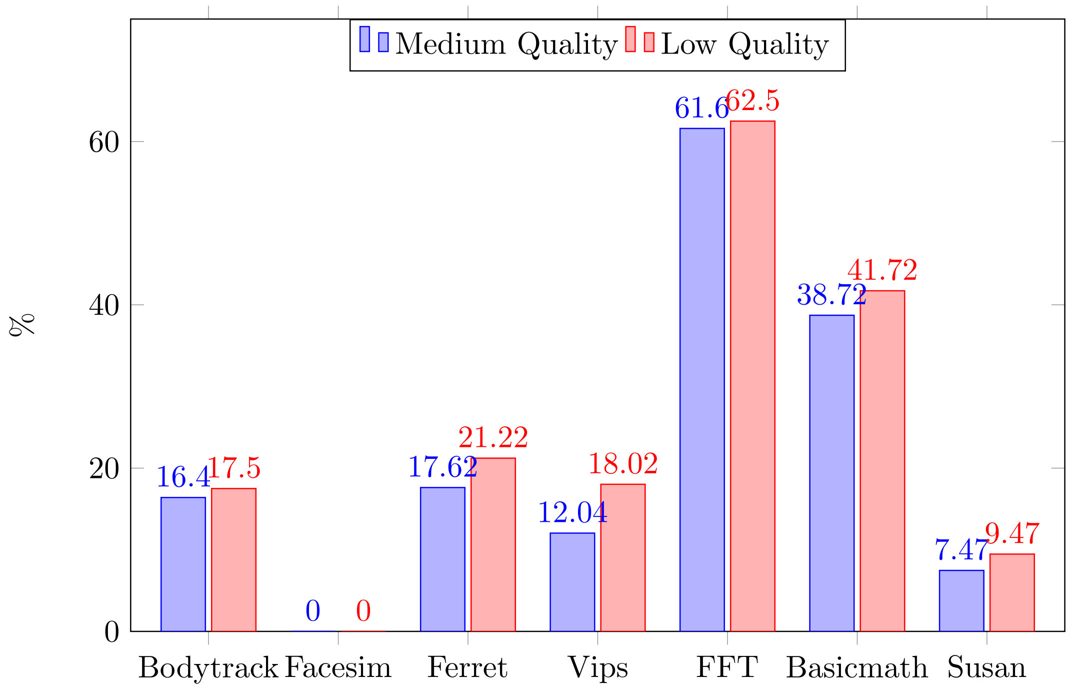


Figure 4.8: Time savings for the applications

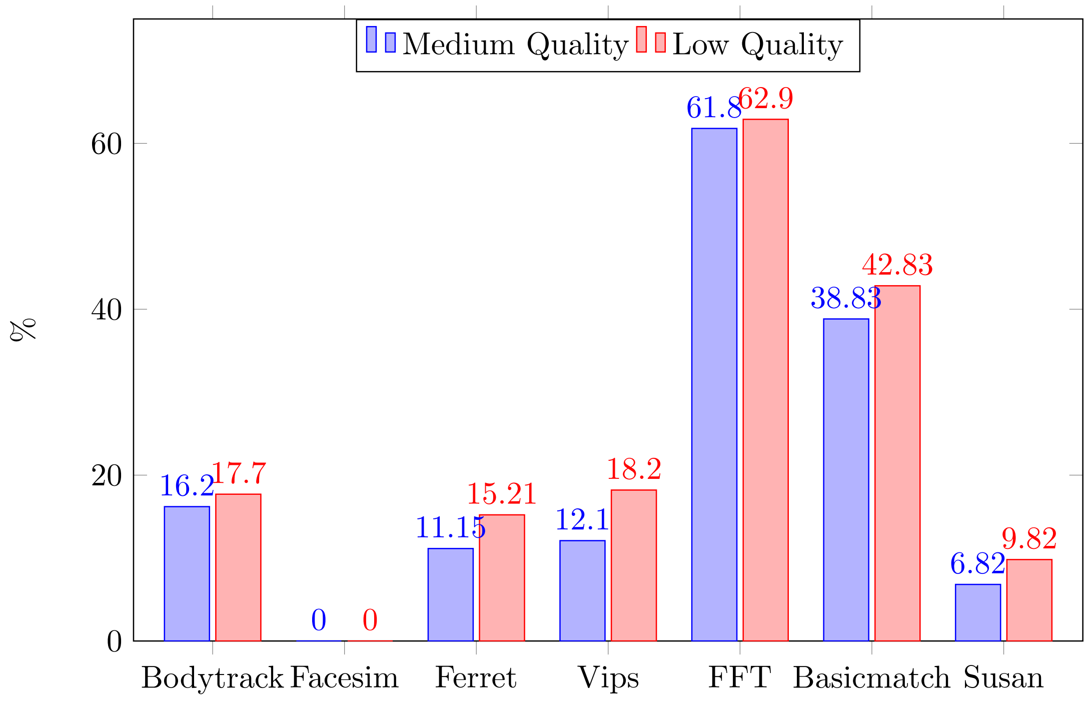


Figure 4.9: Energy savings for the applications

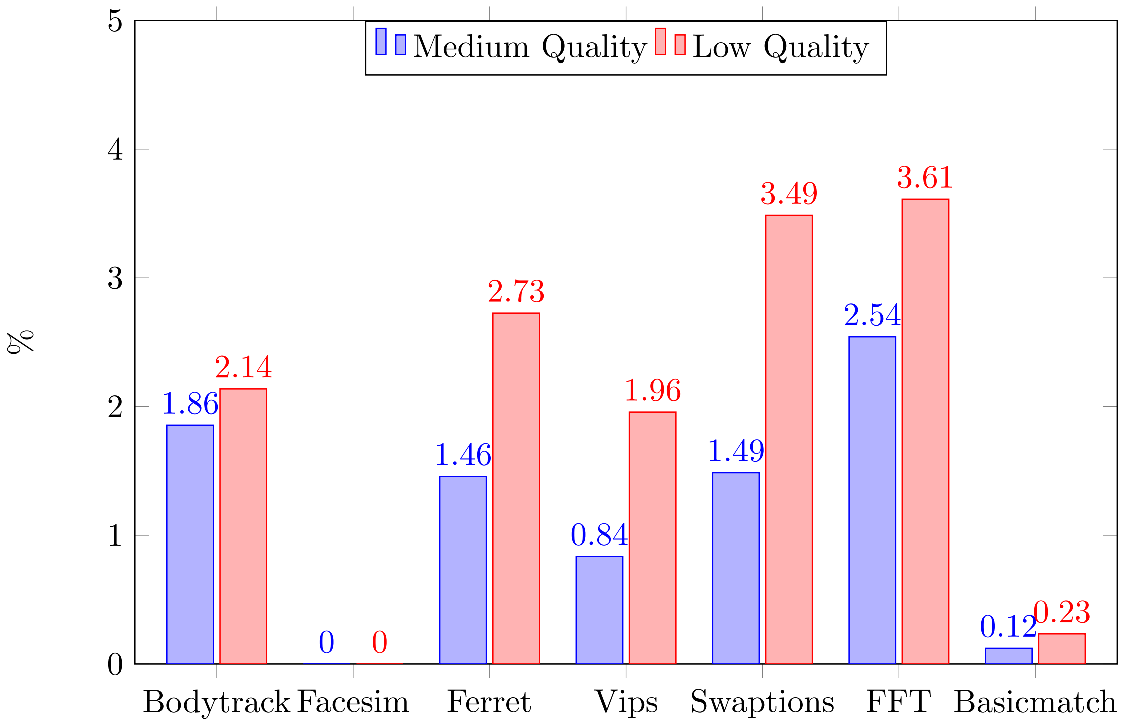


Figure 4.10: Quality Loss for the applications

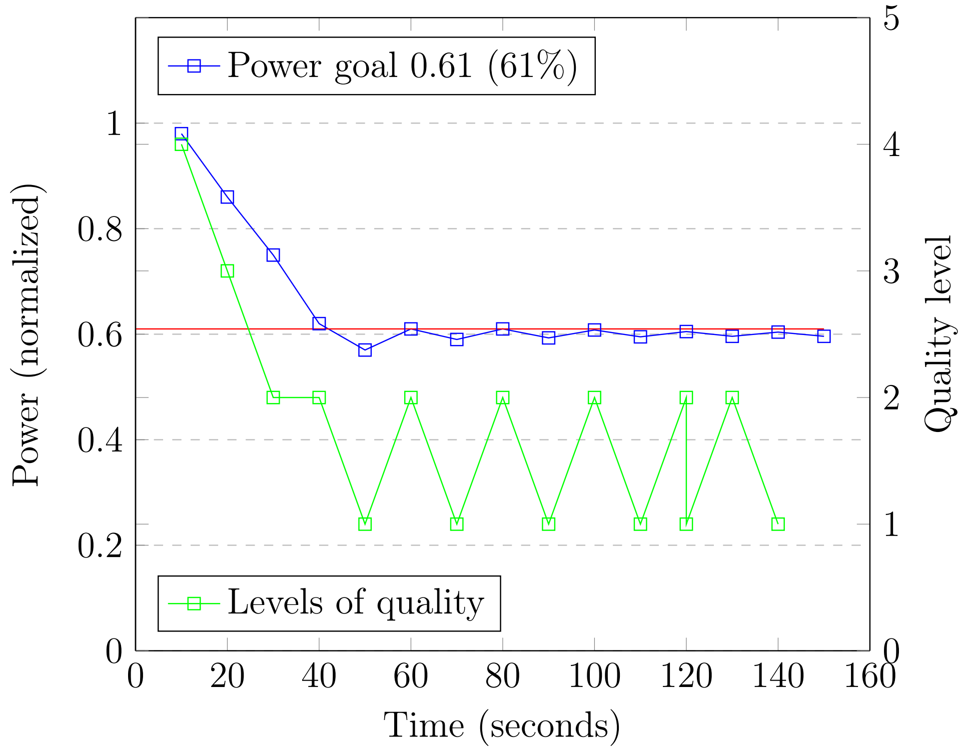
4.4 Dynamic Adaptation

Besides the evaluation for each application individually. We also evaluate how the monitoring system works with the applications. This monitoring system starts with a power goal and it sends signals to the application, so it decreases its quality output (which can be see the be graphics as levels) and this causes a lower power consumption, getting closer to the power goal or increases its quality output to cause a higher power consumption. In the experiments, the monitoring system measures the energy every 10 seconds, this is a parameter that can be changed easily by the user. The power consumption show in the following graphics is an accumulative power consumption over the time and we are showing a graphic for each of the evaluated application shown in the table 4.1. Also, for our evaluations of the applications together with the monitoring system, we didn't take into account the impact of running the monitoring system in the results because this impact is minimal as the monitoring system is a small application compared to the target applications.

4.4.1 Case study 1: Basicmath with 61% energy goal.

In the Figure 4.11 about the Basicmath Application, we can see how the power goal start decreasing at the beginning (starting at 100%, the default power consumption) and the quality start decreasing as well (starting at level 4, the highest and default quality). Once the accumulative power consumption has decrease enough that now is below the power goal (61%) the logic inside the monitoring system will start increasing the power consumption value. This will lead to a point where the accumulative power consumption is again over the power goal, which will cause that the power consumption starts decreasing again, generating an oscillate and repetitive behavior over the time the application is running. We can see in the graphic that this behavior also belongs to the quality levels, it decreases until the power consumption gets below the power goal, then it starts oscillating in the same way as the accumulative power consumption.

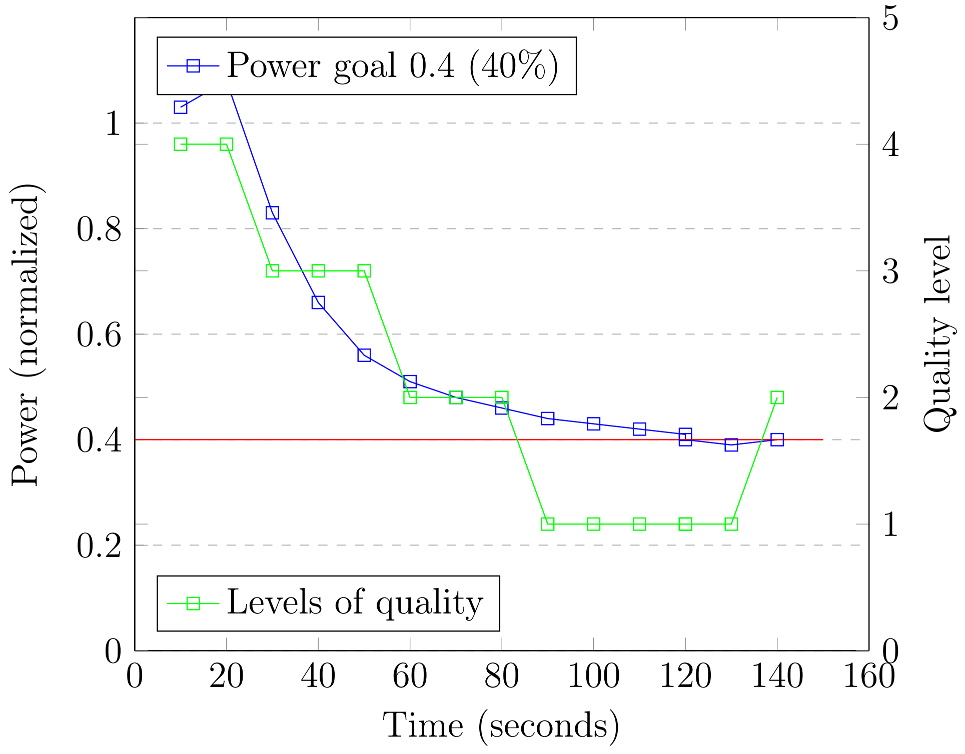
Figure 4.11: Power vs time - Basicmath



4.4.2 Case study 2: FFT with 40% energy goal and a threshold.

In this other graphic 4.12, we are evaluating the behavior of the FFT application using the monitoring system as well. We did a small change in the monitoring system logic. Now when the power consumption is near the power goal, there is a threshold around the power goal. For this case, we fixed a 5% threshold around the power goal. The behavior is similar to the previous application but in this case we can see that oscillation when power consumption is near the power goal is different. The quality oscillates but in a slower way, it stays at quality level 1 for 4 iterations and something similar happens when it gets over the power goal again.

Figure 4.12: Power vs time - FFT Application



4.4.3 Case study 3: Bodytrack with 84% and 60% energy goal.

This other application called Bodytrack has a similar behavior as the first application as shown in the figure 4.13. The power consumption starts oscillating at each iteration when it reaches the power goal. To evaluate the behavior of the application in a different situation we changed the power goal to a value that is even below the lowest power consumption of the application. This is one of the limit scenarios where the power goal is too low that is unreachable for the application. We can see in the second Bodytrack graphic 4.14 that power consumption tries to get closer and closer to the power goal but it never reaches it and just converges to another value, that in our example is close to 0.79. We can see also that the level of quality stays at level 1 once it reaches this value.

Figure 4.13: Power vs time - Bodytrack

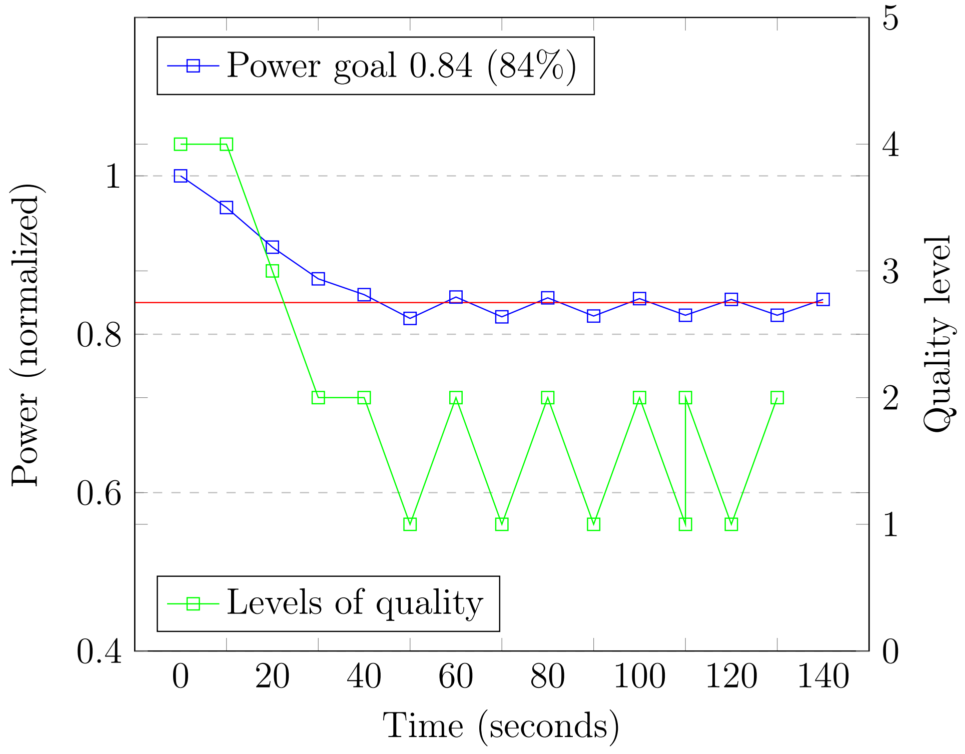
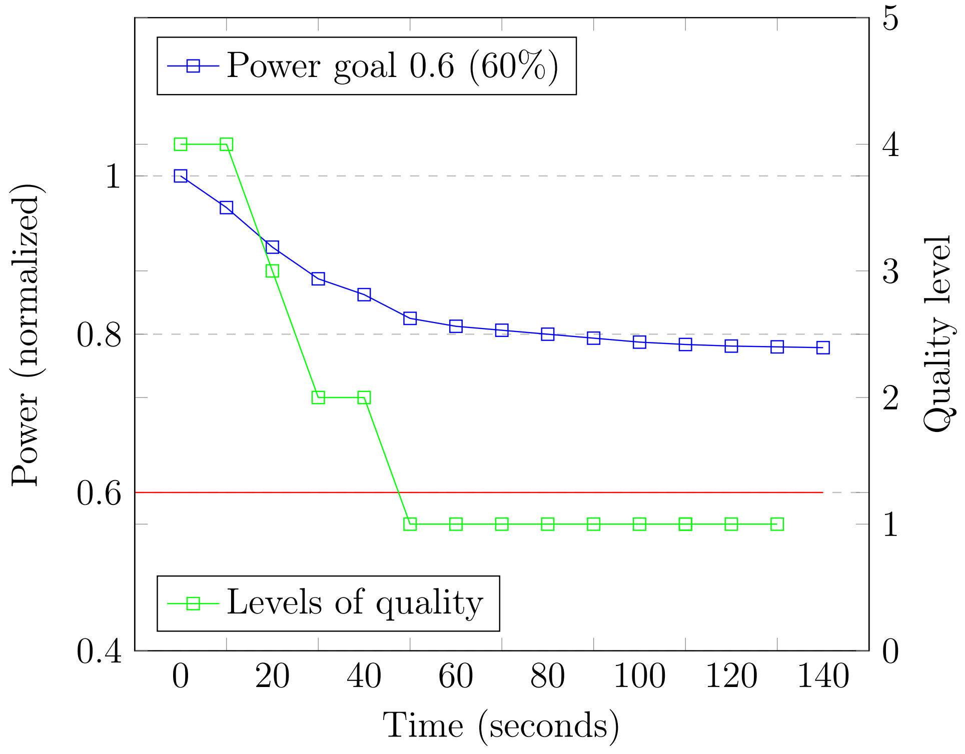


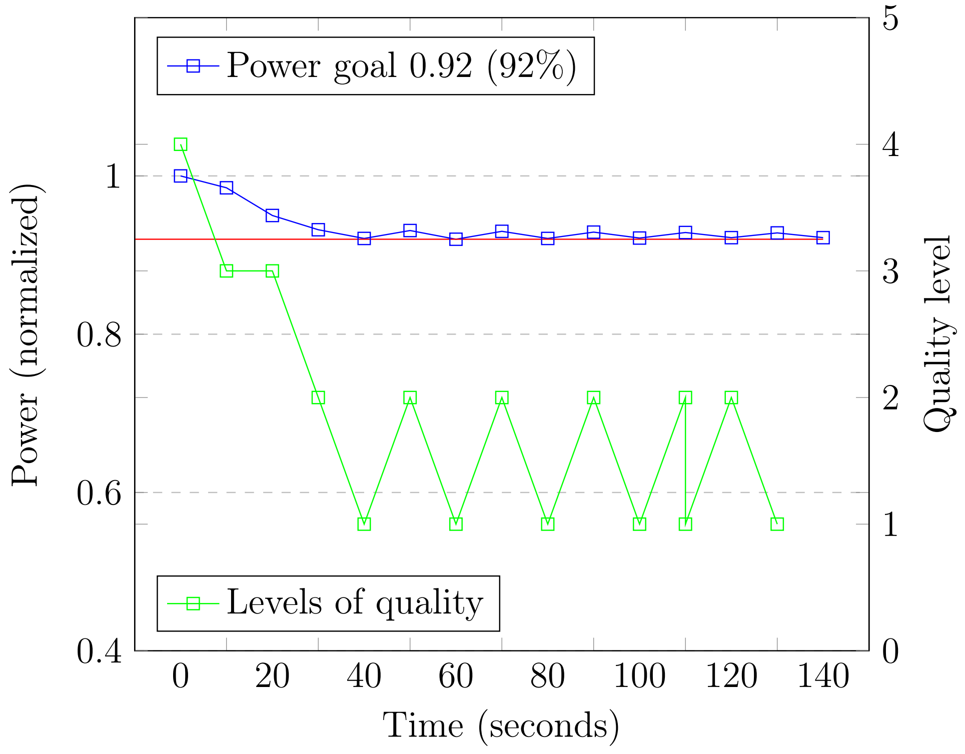
Figure 4.14: Power vs time - Bodytrack modified version



4.4.4 Case study 4: Susan with 92% energy goal.

In this other application shown in the figure 4.15 we can see the same oscillate behavior once it reaches the power goal. This application did not get too much reduction in the power consumption, so we can see all the power consumption values are pretty close to each other.

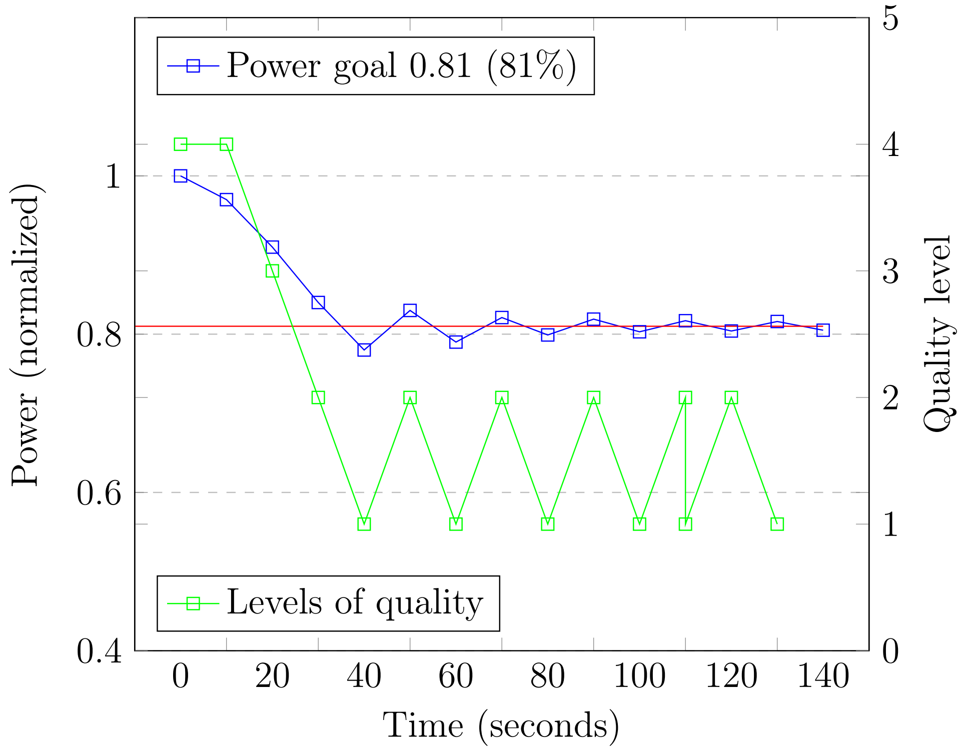
Figure 4.15: Power vs time - Susan

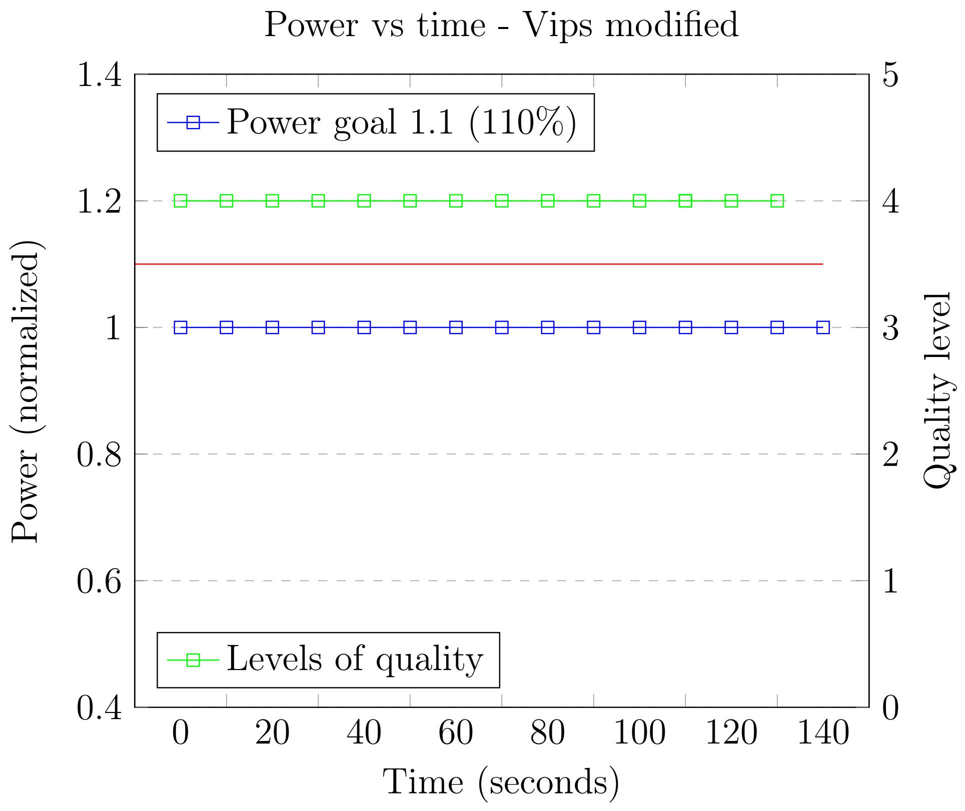


4.4.5 Case study 5: Vips with 81% and 110% energy goal.

This last application Vips has a similar behavior as the previous application as we can see in the figure . So, we will see how it behaves when the power consumption goal is too high for the application. This is the second limit scenario where the power goal is too high to be reached for the application. As we can see in the second graphic, the application starts at the highest quality and highest power consumption so when it tries to reach higher values, the power consumption and quality level just stays at the same value because it can not increase anymore.

Figure 4.16: Power vs time - Vips





Chapter 5

Conclusion

We develop a application support system that includes a library that contains a series of application optimizations having each of these optimization different quality of results and energy consumption. Besides this library, we also present a script that allows an easy incorporation of this library to any application with a minimal user intervention. Our library contains common math functions that are quite used in applications in the field of image processing, signal processing, machine learning, artificial intelligences, scientific computing along with other computer science fields of study. We do a function replacement in polymorphic versions of the standard C math library function in Linux. The application case studies using this library shows that we can trade off a marginal effect on output quality from 0.23% to 3.61% degradation (an average of 2.36%) in application quality for savings from 9.82% to 62.9% (an average of 27.78%) in energy consumption and time execution.

Our work also presents a monitoring system that measures current energy consumption of the system in previously fixed intervals of time. After each of this intervals, the monitoring system can send signals to the target application sending an order of increasing or decreasing the quality of certain function implementations that are in the library and at the same time decreasing or increasing the energy consumption of these implementations respectively, which will cause to decrease or increase the energy consumption of the whole target application. This kind of interaction will allow us to build policies around each of these function implementations and generic performance system values like energy consumption, in this work we are working with energy but the logic and approximate implementations can be worked along with other parameters like for example CPU usage. In our experiments, our goal is fixed in an energy consumption value and our policies guide the system energy consumption to this value decreasing or increasing the quality of the target application depending on how the current energy consumption of the system is. In our results, we did different case of studies, each them having different scenarios and in all of them the results were satisfactory and in all the cases we were able to reach the energy goal as much as possible by the cost of the quality of the application that we could sacrifice.

Our application support system strengths are that it can reduce considerably the energy consumption of heavy oriented math function applications as our results show. Another advantage is that user intervention for using this system is minimal, it only requires a few changes in the compilation settings of the target application. And the last advantage is that the function implementations of the library can be changed dynamically with the use of signals, which allows us to build policies around a performance goal for the application. Our limitations are on applications which do not use math functions inside them or applications where math functions are not presented in the most energy consumption part of the application.

Bibliography

[1] Fawzi Albalooshi Amjad Mahmood, Salman A. Khan and Noor Awwad. Energy-aware real-time task scheduling in multiprocessor systems using a hybrid genetic algorithm. 2017.

[2] Jason Ansel, Cy Chan, Yee Lok Wong, Marek Olszewski, Qin Zhao, Alan Edelman, and Saman Amarasinghe. Petabricks: A language and compiler for algorithmic choice. *SIGPLAN Not.*, 44(6):38–49, June 2009.

[3] Woongki Baek and Trishul M. Chilimbi. Green: A framework for supporting energy-conscious programming using controlled approximation. *SIGPLAN Not.*, 45(6):198–209, June 2010.

[4] Christian Bienia. *Benchmarking Modern Multiprocessors.* PhD thesis, Princeton, NJ, USA, 2011. AAI3445564.

[5] Caswell D. and P. Debaty. Creating web representations for places. *Proceedings of the 2nd International Symposium on Handheld and Ubiquitous Computing (HUC2K)*, 2000.

[6] Salma Elmalaki, Lucas Wanner, and Mani Srivastava. Caredroid: Adaptation framework for android context-aware applications. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, MobiCom '15, pages 386–399, New York, NY, USA, 2015. ACM.

[7] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. Neural acceleration for general-purpose approximate programs. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-45, pages 449–460, Washington, DC, USA, 2012. IEEE Computer Society.

[8] Mark Gottscho, Abbas BanaiyanMofrad, Nikil Dutt, Alex Nicolau, and Puneet Gupta. Power / capacity scaling: Energy savings with simple fault-tolerant caches. In *Proceedings of the 51st Annual Design Automation Conference*, DAC '14, pages 100:1–100:6, New York, NY, USA, 2014. ACM.

[9] Soheil Hashemi, R. Iris Bahar, and Sherief Reda. Drum: A dynamic range unbiased multiplier for approximate applications. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '15, pages 418–425, Piscataway, NJ, USA, 2015. IEEE Press.

[10] Henry Hoffmann, Stelios Sidiroglou, Michael Carbin, Sasa Misailovic, Anant Agarwal, and Martin Rinard. Dynamic knobs for responsive power-aware computing. *SIGPLAN Not.*, 46(3):199–212, March 2011.

[11] Kevin A. Li, Timothy Y. Sohn, Steven Huang, and William G. Griswold. Peopletones: A system for the detection and notification of buddy proximity on mobile phones. In *In Proc. 6th Int'l. Conf. on Mobile Systems (MobiSys)*, 2008.

[12] Tung-Hung Lu, Hsing-Chen Lin, Rong-Rong Chen, and Ya-Ling Chen. Motion-sensing based management system for smart context-awareness rehabilitation healthcare. *Advances in Internet of Things*, 3(2A), 2013.

[13] Liangzhen Lai Puneet Gupta Lucas Wanner, Salma Elmalaki and year = 2013 note = AAI3445564 Mani Srivastava, title = VarEMU: An Emulation Testbed for Variability-Aware Software. PhD thesis.

[14] Anil Madhavapeddy, David Scott, and Richard Sharp. Context-aware computing with sound. In *In Proceedings Of The 5th International Conference On Ubiquitous Computing*, pages 315–332, 2003.

[15] Joshua San Miguel, Jorge Albericio, Andreas Moshovos, and Natalie Enright Jerger. Doppelganger: A cache for approximate computing. In *Proceedings of the 48th International Symposium on Microarchitecture*, MICRO-48, pages 50–61, New York, NY, USA, 2015. ACM.

[16] Pradeep K. Murukannaiah, Ricard Fogues, and Munindar P. Singh. Platys: A framework for supporting context-aware personal agents. In *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '14, pages 1689–1690, Richland, SC, 2014. International Foundation for Autonomous Agents and Multiagent Systems.

[17] Elizabeth D. Mynatt, Jim Rowan, Sarah Craighill, and Annie Jacobs. Digital family portraits: Supporting peace of mind for extended family members. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '01, pages 333–340, New York, NY, USA, 2001. ACM.

[18] G. J. Nelson. *Context-aware and location systems.* PhD thesis, University of Cambridge, Cambridge, U.K, 1998.

[19] Jason Pascoe. The stick-e note architecture: Extending the interface beyond the user. In *Proceedings of the 2Nd International Conference on Intelligent User Interfaces*, IUI '97, pages 261–264, New York, NY, USA, 1997. ACM.

[20] J. G. Reis, A. A. Frohlich, and L. F. Wanner. X-ware: mutant computing substrates. In *2015 International Symposium on Rapid System Prototyping (RSP)*, pages 25–31, Oct 2015.

[21] Adrian Sampson, Werner Dietl, Emily Fortuna, Danushen Gnanapragasam, Luis Ceze, and Dan Grossman. Enerj: Approximate data types for safe and general low-power computation. *SIGPLAN Not.*, 46(6):164–174, June 2011.

[22] William Noah Schilit. *A System Architecture for Context-aware Mobile Computing.* PhD thesis, New York, NY, USA, 1995. UMI Order No. GAX95-33659.

[23] Hardik Sharma, William Wahby, Thomas Sarvey, Muhannad S Bakir, and Hadi Esmailzadeh. Video anomaly detection in real time on a power-aware heterogeneous platform. 2016.

[24] Jacob Sorber, Alexander Kostadinov, Matthew Garber, Matthew Brennan, Mark D. Corner, and Emery D. Berger. Eon: A language and runtime system for perpetual systems. In *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems*, SenSys '07, pages 161–174, New York, NY, USA, 2007. ACM.

[25] Kimberly Tee, Saul Greenberg, and Carl Gutwin. Providing artifact awareness to a distributed group through screen sharing. In *Proceedings of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work*, CSCW '06, pages 99–108, New York, NY, USA, 2006. ACM.

[26] Alf Inge Wang, Qadeer Khan Ahmad, and Key Words. Camf - context-aware machine learning framework for android. In *In Proceedings of SEA '10. Marina Del Rey.* ACTA Press, 2010.

[27] Lucas Wanner and Mani Srivastava. Virus: Virtual function replacement under stress. In *Proceedings of the 6th USENIX Conference on Power-Aware Computing and Systems*, HotPower'14, pages 2–2, Berkeley, CA, USA, 2014. USENIX Association.

[28] Roy Want, Andy Hopper, Veronica Falcão, and Jonathan Gibbons. The active badge location system. *ACM Trans. Inf. Syst.*, 10(1):91–102, January 1992.