FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Energy-aware resource management for heterogeneous systems

Eduardo Fernandes

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Jorge Barbosa

July 7, 2016

Energy-aware resource management for heterogeneous systems

Eduardo Fernandes

Mestrado Integrado em Engenharia Informática e Computação

July 7, 2016

Abstract

Nowadays computers, be they personal or a node contained in a multi machine environment, can contain different kinds of processing units. A common example is the personal computer that nowadays always includes a CPU and a GPU, both capable of executing code, sometimes even in the same integrated circuit package. These are the so called heterogeneous systems.

It's important to be aware that the various processing units aren't equal, for instance CPUs are very different from GPUs. This raises a problem, since not every task can be executed in all processing units.

To solve this problem a new task scheduling algorithm was developed with the aid of SimDag from the SimGrid toolkit. This algorithm uses a DAG (directed acyclic graph) to aid the scheduling of different tasks, be they from a single application or from various different applications.

The algorithm is based on the HEFT scheduling algorithm, a greedy algorithm with a short execution time, developed by Topcuoglu et al. This new algorithm is aware of the different processing units and of the different performance/power levels. This solves the problem of not all tasks being able to be executed in all processing units.

Since previous studies show that reducing the CPU clock speed on DVFS (dynamic voltage frequency scaling) CPUs can reduce the energy spent by the CPU while executing various tasks with little increase in runtime. Various tests were made to obtain the power rating of a test CPU while operating on different performance levels. With this it was possible to obtain performance and power information on the power states, this information is then later used by the algorithm in order to find the optimal performance/power ratio.

The algorithm main objective is to spend the least amount of energy possible, in contrast to the HEFT goal that is to execute tasks as fast as possible. The algorithm behavior can be modified by changing the minimum power state that the processing units should run or by changing the goal. Two goals are provided, the EFT (earliest finish time) from the original HEFT algorithm and the LEC (least energy cost). Both goals are affected by the defined minimum power state.

Using this new algorithm it was possible to reduce total energy spent some times at the cost of increased runtime.

Resumo

Nos dias de hoje os computadores quer sejam pessoais ou um nó contido num ambiente multi maquina, podem conter diversos tipos de unidades de processamento. Um exemplo comum é o computador pessoal que nos dias de hoje inclui sempre um CPU e um GPU ambos capaz de executar código, muitas vezes no mesmo circuito integrado. Estes sistemas são heterogéneos.

É importante estar consciente que as varias unidades de processamento não são iguais. Por exemplo os CPUs são bastante diferentes dos GPUs. Com isto surge um problema, pois nem todas as tarefas podem ser executadas em todas as unidades de processamento.

Para solucionar este problema um novo algoritmo de escalonamento foi desenvolvido recorrendo ao SimDag pertencente ao toolkit do SimGrid. Este algoritmo utiliza um DAG (grafo direcionado acíclico) para facilitar o escalonamento de diferentes tarefas, sejam elas provenientes de uma única aplicação ou de várias aplicações diferentes.

Este algoritmo é baseado no algoritmo de escalonamento HEFT desenvolvido por Topcuoglu et al. É um algoritmo ganancioso, mas de rápida execução. Este novo algoritmo está ciente tanto das varias unidades de processamento como dos diferentes níveis de performance/potência. Com isto o problema de nem todas as tarefas poderem executar em todas as unidades de processamento fica resolvido.

Visto que estudos anteriores mostram que reduzindo a frequência de relógio do CPU em sistemas baseados em DVFS (sistemas de escalamento dinâmico de voltagem e frequência) os CPUs podem reduzir a energia gasta a executar diferentes tarefas com um pequeno aumento no tempo de execução. Vários testes foram efetuados para obter a potência consumida por um CPU enquanto este operava em diferentes níveis de performance. Com isto foi possível obter informação a cerca da performance e respetiva potência relativos aos diversos níveis de performance, esta informação é depois utilizada pelo algoritmo de maneira a encontrar o rácio mais vantajoso de performance/potência.

O objetivo principal do algoritmo é gastar a menor quantidade de energia possível, isto em contraste com o HEFT cujo objetivo é executar as tarefas o mais rápido possível. O comportamento do algoritmo pode ser modificado alterando o estado de energia mínimo que as unidades de processamento devem executar ou alterando o objetivo. Dois objetivos são fornecidos, o EFT (tempo para completar mínimo) do algoritmo original HEFT e o LEC (menor custo de energia). Ambos os objetivos são afetados pelo estado de energia mínimo definido.

Utilizando este novo algoritmo foi possível reduzir o total de energia gasta. As vezes a custa do aumento do tempo de execução.

Acknowledgements

I would first like to thank my supervisor Jorge Barbosa for all the input and advice given during the development of this thesis. I would also like to thank to all the people from the SPeCS research group at FEUP for the input given.

I would like to thank my family and friends for helping and supporting me at all times.

I am very grateful by the constant help and support from Vanessa Ramos.

I would like to thank Ricardo Coutinho for the help given reviewing this thesis.

Eduardo Fernandes

"Don't kick the robots."

Mikko Hyppönen

Contents

List of Figures

LIST OF FIGURES

List of Tables

Abbreviations

API	*Application Programming Interface*
CPU	*Central Processing Unit*
CUDA	*Compute Unified Device Architecture*
DAG	*Directed Acyclic Graph*
DRAM	*Dynamic Random-Access Memory*
DSP	*Digital Signal Processor*
DVFS	*Dynamic Voltage and Frequency Scaling*
FLOPS	*Floating-Point Operations Per Second*
FPGA	*Field Programmable Gate Array*
GLOPS	*Giga Floating-Point Operations Per Second*
GPGPU	*General Purpose computing on Graphic Processing Units*
GPU	*Graphics Processing Unit*
HPC	*High Performance Computing*
IC	*Integrated Circuit*
MSR	*Model Specific Registers*
NVML	*NVIDIA Management Library*
RAM	*Random-Access Memory*
RAPL	*Running Average Power Limit*
SoC	*System-On-Chip*
TFLOPS	*Tera Floating-Point Operations Per Second*
TDP	*Thermal Design Power*
TSC	*Time Stamp Counter*
ULV	*Ultra Low Voltage*
WWW	*World Wide Web*

Chapter 1

Introduction

1.1 Problem statement

Nowadays different kinds of processing units exist (CPUs, GPUs, special purpose accelerators, among others) and some times more than one processing unit is present in a single computing system.

Many examples of this exist: a typical laptop computer usually includes two different processing units (CPU and GPU), the highly integrated *SoC* present on a smart phone also includes various processing units (CPU, GPU, DSP, some times others), to high performance computing grids that use large amounts of processing units. All these systems are heterogeneous, since they employ different processing units with distinct characteristics and most of the times with different architectures.

These systems pose a new problem: leveraging the right architecture for the task at hand. This leads to increasing research efforts on how to leverage computing power from these systems.

High Performance Computing is an important area that many fields depend upon. Heavy computing reliant domains like climate research, molecular modeling, weather forecasting, aerodynamic modeling and others can benefit from higher throughput and better energy efficiency.

Since many of these computations need to be finished in useful time or within some financial cost, most of the times models have their accuracy reduced to finish computation faster and stay within budget. This problem can be solved by providing more efficient computing power so that more accurate models can be used.

1.2 Motivation and Objectives

Currently there is a demand for high computing power, but it comes at a very high cost, be it financial, temporal or energetic.

Many times the software doesn't take advantage of the existing hardware. One example is the rise of multi-core CPU and the lack of proper use of the multiple cores present in the system. Another example is the lack of use of GPUs for computational purposes.

With this comes the need or better leveraging of different processing units present in computer systems.

Since there are various computing architectures available, with different characteristics, many times in the same machine there is the need to better use those architectures. Different architectures have distinct advantages and disadvantages, so leveraging the correct architecture for a specific task might provide lower costs and increase performance.

The main objective is to create a scheduling algorithm that optimizes the use of current heterogeneous computing platforms and better distribute tasks while being aware on the energy spent.

This work is being developed under the Antarex project, whose objective is to provide a way to manage and auto tune applications in order to reach exascale computing at around 20 Megawatts.

In order to reach this goal the Antarex project proposes controlling all the decision layers, in order to implement a self-adaptive application optimized for energy efficiency [SL].

1.3 Dissertation Structure

This dissertation is divided in six chapters, the first is a brief introduction (this chapter). It is followed by the second chapter where the literature review and related work are presented. The third chapter explains the used methodology. The fourth chapter contains the obtained results. The fifth chapter contains the implementation details. Finally the last chapter is where the conclusions and further work are discussed.

Chapter 2

Background

2.1 Introduction

In this chapter the literature review and related work are presented. Different computing system types are also explored. A small introduction to task graphs and scheduling algorithms is given. The main tools usable in this project will also be discussed.

2.2 Computing System Types

In this section different computing system types will be discussed: homogeneous and heterogeneous systems.

2.2.1 Homogeneous Systems

An homogeneous computing system is a system that uses only one kind of processor. One example is a headless computer that only has a kind of processing unit like a CPU. Older personal computers could also be considered an homogeneous system, since the video card included at the time could not do anything else than provide a way to output images to a display.

These kind of systems are not the most appropriate since the only available processor may or may not be efficient for the task at hand.

It should also be noted that on mainstream computing platforms this type of system is quickly disappearing.

2.2.2 Heterogeneous Systems

An heterogeneous computing system is a system that uses more than one kind of processor. One example of an heterogeneous system is the regular personal computer that includes a CPU and a GPU (some cases in the same IC package).

This type of system is widely available, nowadays most devices ranging from mobile phones to high performance computing systems typically include more than one kind of processor. The

most common processor types available today are the regular CPU (x86 based or ARM based) and the GPU (Intel HD Graphics, NVIDIA graphic cards and AMD graphic cards).

With the evolution of graphic cards, some parts of the graphics pipeline began to be programmable, leading to the birth of the GPGPU (General Purpose computing on Graphic Processing Units) [ND10]. Some tasks like large data driven parallel operations (eg: image processing, matrix computations) can be executed on the GPU, requiring less energy and time to accomplish [RRB+08].

Currently there are two widely used platforms: CUDA and OpenCL. CUDA currently only works on NVIDIA GPUs [Cor16a]. On the other hand OpenCL is a more compatible solution, since it is supported by the three major GPU manufacturers (AMD, Intel and NVIDIA). It should also be noted that OpenCL supports more kinds of processors, like DSPs and FPGAs [Gro15].

Since most GPUs nowadays support GPGPU, either by CUDA or OpenCL, it is a good idea, to leverage the computing power provided by them (if beneficial for the task in hand).

2.3 Task Graphs

A graph is a representation of a set of objects (vertices) and links between those objects (edges), an example is shown in figure 2.1 where A, B, C, D and E are vertices, edges are not labeled but one example is the link between vertex A and B.

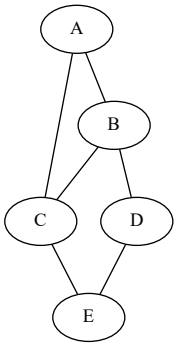

Figure 2.1: Simple Graph

Task graphs are a simple way of representing task dependencies. For this work in particular the tasks are computational tasks. Tasks will be represented by vertices and dependencies will be represented by edges. An example is shown in figure 2.2. In this example it should be noted that the edge value is not represented. Depending on the context the edge can have different meanings, like cost or time.

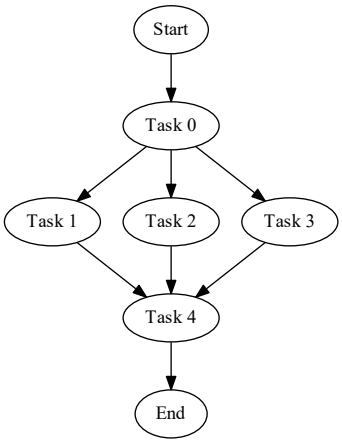

Figure 2.2: Sample Task Graph

Task Graphs are widely used in the study of scheduling parallel tasks [RHC15].

2.3.1 Directed Acyclic Graph (DAG)

A directed acyclic graph (DAG) is a directed graph that doesn't contain cycles. It is a simple way to represent intertask dependencies, since one task cannot be executed until the previous task has been executed. Weights can be assigned to vertexes and edges, in this case computation costs will be assigned to vertexes and communication costs to edges. This kind of notation has already been used by various works (some omit cost) [AB14, RHC15].

As the name implies, this method of task organization doesn't allow a program to go back to a previous task, since that would create a cycle, thus imposing a limitation on the possible programs to be represented by this kind of graph.

2.4 Scheduling Algorithms

A scheduling algorithm is what determines how a specific task will be assigned to the resources in order to complete that task. There are many uses for scheduling algorithms, the most known examples in computer systems are the operating system process schedulers which determine how much CPU time is allocated to a determined process. This is the main focus of this thesis.

There are various kinds of scheduling algorithms and they can either be Best-effort based or QoS-constraint based [YBR08].

2.4.1 Best-effort

Best-effort algorithms attempt "to minimize the execution time ignoring other factors" [YBR08]. These algorithms ignore all kinds of costs. If there are no other users in the system these algorithm's biggest problem is not being energy aware since they ignore any kind of costs. This might increase power consumption, push systems power budgets further than what is allowed.

2.4.2 QoS-constraint

QoS constraint algorithms attempt "to minimize performance under most important QoS constraints, for example time minimization under budget constraints or cost minimization under deadline constraints" [YBR08].

The scheduling algorithm to be developed fits in this category since it has to work with restrictions, like energy consumption.

2.5 Power Consumption Measurements

In this section two ways to measure a single node (or a personal computer) power consumption will be discussed, either by using internal hardware counters or through the use of specialized hardware.

2.5.1 Internal Hardware Counters

Recent processors provide hardware counters which report energy consumption by the CPU. Intel started providing these counters with the Sandy Bridge architecture, they are the RAPL (Running Average Power Limit) interfaces.

They are divided in different counters:

- PP0: power plane 0, consumption of all (physical) cores
- PP1: (only present in the client segment): uncore devices, like the integrated GPU
- Package: processor die, includes all CPU components (whole CPU package)
- DRAM (only present in the server segment): directly attached DRAM

This information is provided by Intel on it's IA-32 Manual Volume 3B, section 14.9.1 [Cor11]. These values are stored in model specific registers (MSR) and can be read using appropriate instructions. There are APIs that abstract the user from having to deal with reading these values from the CPU. There are previous studies that have used this approach [RRS+14].

It should be noted that on the Intel Xeon Phi family even more accurate values can be obtained [Cor15b].

No concrete information for AMD CPUs was found, but it appears that some AMD CPUs report the power consumption of different components as seen on CodeXL profiler examples (compute units, integrated GPU and package total are available). Since the author could not obtain any AMD based system no further research was done.

It should be noted that Intel RAPL counters are based in a sophisticated mathematical model tailored to each CPU model [Corb].

2.5.2 External Hardware

There are many ways to obtain power consumption measurements of a computer system each with its pros and cons. One common con to all of them is the need of external hardware to do the measurements (power meter). Another con is the need of a new communication channel so that the system knows how much power is currently consuming.

The most simple method is to use a power meter that measures the power drawn by the system from the electric outlet. This method only provides a global value, the power consumption from the various internal devices is added up into a single value, having a major disadvantage: there

is no way to know how much energy the CPU, GPU, RAM, storage devices, cooling systems, external peripherals and other components are individually consuming [Cor15b].

The other available method is to modify the computer system so that all power rails are separately monitored. This method is somewhat more accurate and should provide a more detailed power consumption value of each individual component. It has two big disadvantages: the need to modify the computer system, making it impractical for regular use and the need to really know how the hardware works electrically, since the various power rails might not be entirely isolated (for example: various 12 volt rails exist). Like the internal hardware counters there is a study that has used this approach [RRS+14].

2.6 Available Simulation Tools

In this section there is a small overview of some available simulation tools.

BigSim

BigSim is a "simulation system [that] consists of an emulator and a simulator." [EB15]. The BigSim is a tool to help programmers create applications that will be used in supercomputers, helping programmers "develop, debug and tune/scale/predict the performance of applications before such machines are available" [EB15].

It is released under the Charm++/Converse license (open source with restrictions) and written mainly in C language. Currently it supports only Charm++ or AMPI programs which limits its usefulness. It is available with the Charm++ language source package.

GroudSim

GroudSim is "a Grid and Cloud simulation toolkit" [OPPF11]. GroudSim is written in Java, it is not freely available (there is a version available for students but its use is restricted) and is part of the ASKALON toolkit [OPPF11].

There is almost no documentation or references to GroudSim.

Narses

Narses is a network simulator targeting large distributed applications [GB02]. Narses is released under an unknown license, written in Java and was last updated in 2003 [Giu13].

It is only a network level simulation tool so it lacks many required features needed for the development of this work. The project seems dead and there is no documentation available. It should also be noted that according to Da SimGrid Team, Narses has known issues [Tea15].

OptorSim

OptorSim by EU DataGrid is a Data Grid simulator, its goal is to "allow experimentation with and evaluation of various replica optimization strategies." [CCSF+06] It is released under the EU DataGrid Software License, written in Java and was last updated in 2006 [APM13].

The project resulted from the DataGrid Project which ended in March of 2004 [Tea04]. Since then most web pages are missing and the only page left online is hosted on SourceForge (which is a hosting provider) project page that has the source code and the user guide available for download. It should also be noted that according to Da SimGrid Team, OptorSim has known issues [Tea15].

SimGrid

SimGrid by Da SimGrid Team is "a toolkit that provides core functionalities for the simulation of distributed applications in heterogeneous distributed environments" [Tea15]. It is released under the GNU Lesser General Public license (LGPL) making it open source.

The latest release was made in April of 2016 and its actively maintained. It is also one of the most complete tools available [CGL+14]. Da SimGrid Team also provides documentation and tutorials on the SimGrid website ([Tea16]).

2.6.1 Comparison between tools

Where a small comparison between the tools is made. The table 2.1 shows a comparison of the tools.

Tool	Development Status	License	Documentation	Needed features
BigSim	Active	Charm++/Converse license	Yes	Yes
GroudSim	Unkown	Unkown	Unsure	Unsure
Narses	Inactive	Unkown	No	No
OptorSim	Inactive	EU DataGrid Software License	Yes	No
SimGrid	Active	LGPL	Yes	Yes

Table 2.1: Comparison between simulation tools

Tools with no recent developments were excluded (some of them have more than 10 years without updates). This excludes both Narses and OptorSim.

Tools with unknown or restricted licensing were also excluded from the tool list, excluding GroudSim and Narses.

Excluding GrouSim, Narses and OptorSim from the tools list only leaves BigSim and SimGrid as viable options.

Ultimately SimGrid was chosen to be used for the simulations because it has been proven to be accurate, scalable, has good documentation and is popular on the academic world.

2.7 Available Energy Consumption Reporting Tools and APIs

Power API

Power API by Sandia Corporation is "a portable API for power measurement and control" [Cor14]. A reference implementation is provided ([Cor16b]) and it is written in C++ and C languages with sources available. It is released under a special license but it has the source code freely available.

It has a broad scope and a portable API [Cor15c]. It provides the possibility to be extended, so that it supports other hardware.

Intel Power Gadget

Intel Power Gadget is a software tool designed to monitor the power usage of Intel Core processors (starting from the 2nd generation). It reads the MSR registers and allows the logging of power consumption over time for different components of the CPU package [Cor15a].

Intel micsmc utility

Intel micsim utility is a tool designed to "manage the Intel Xeon Phi power configuration and facilitate power measurements" [Cor15b]. This tool while accurate is limited in usefulness since it only works with the Xeon Phi family.

Pstate-frequency

Pstate-frequency is a tool designed to manipulate the Linux kernel *intel_pstate* CPU scaling driver, enabling the user to set the maximum and minimum frequencies of the CPU and enable or disable the Intel Turbo Boost feature [Yam].

NVIDIA Management Library

The NVIDIA Management Library (NVML) is a C based API for "monitoring and managing various states of the NVIDIA GPU devices" [Corc]. It's a tool that only works on NVIDIA GPUs making it limited in usefulness.

NVIDIA System Management Interface

The NVIDIA System Management Interface is a utility that is based on the NVML explained earlier. This utility allows to get the CPU power consumption from some NVIDIA cards [Cord].

AMD CodeXL Profiler

The AMD CodeXL profiler is a utility that helps developers profile their code and provides power consumption values from some CPUs and GPUs [AMDc]. It should be noted that CodeXL is now open source (as of April 19, 2016) and is now part of GPUOpen [AMDb].

2.7.1 Comparison between tools

All tools are somewhat limited in one way or another. Most of them have limited hardware support. This creates an additional overhead since the tools must be tested on every test platform. All of them rely on hardware built in functionally and unless a bug is present the same hardware results should be the same across various tools.

Power API was not chosen since it had to be integrated on the used application in test. Intel micsmc was not used since no Xeon Phi platform was available for testing. Both NVIDIA tools were not used since the available NVIDIA GPU doesn't support power consumption reporting. AMD CodeXL doesn't support the available hardware so it was also not used. Both Intel Power Gadget and Pstate-frequency utilities were used since they support the available hardware.

Chapter 3

Methodology

3.1 Introduction

This chapter contains the used methodology. First the energy analysis is discussed followed by the directly related performance analysis. Then the chosen benchmarks are detailed but it should be noted that this was not the main focus of the thesis so its scope is rather narrow. Finally the simulated platform model is detailed along with the task graph model.

3.2 Power and Energy Analysis

In the previous chapter two methods of obtaining power and energy consumptions were discussed, the hardware and software methods.

Since the external hardware methods obtain either a global value (that might be influenced by different factors) or involve invasive hardware modifications, they will not be used. Also there is another issue with the external hardware method that is the need to communicate and log the data provided by the measurement equipment.

The method that utilizes internal hardware counters was chosen since the values provided offer "a good match" [RRS+14] between the obtained values measured with the hardware counters and by using a power meter on the power rails that feed the CPU. Using internal hardware counters also provides another benefit: the ease to obtain those values through software, for example by using the Power API, which is very important in a multi node system. Another benefit is that more detailed power consumption data can be obtained from Intel CPUs, since they are partitioned in different internal power planes that provide their power consumption independently from each other.

Another thing that must be considered is the so called Turbo Boost (Intel) or Turbo Core (AMD) and other similar boost technologies that attempt to increase the performance of the CPU or GPU if possible. One example is if only a CPU core is being used that core can use a higher clock and effectively speeds up what is running on the core in use.

System Time	The system time when this particular measurement was taken.
RDTSC	The *TSC* value, which is the number of cycles since CPU reset.
Elapsed Time (sec)	Time elapsed since the Intel Power Gadget tool was started.
IA Frequency_0 (MHz)	CPU Core frequency. If more than one core exists in the package, this contains the average frequency of those cores.
Processor Power_0 (Watt)	Total CPU package power consumption.
Cumulative Processor Energy_0 (Joules)	Sum of the energy spent by the CPU package since the logging start.
Cumulative Processor Energy_0 (mWh)	
IA Power_0 (Watt)	CPU cores power consumption. (Also known as PP0)
Cumulative IA Energy_0 (Joules)	Sum of the energy spent by the CPU cores since the logging start.
Cumulative IA Energy_0 (mWh)	
GT Power_0 (Watt)	Internal GPU power consumption. (Also known as PP1)
Cumulative GT Energy_0 (Joules)	Cumulative energy spent by the internal GPU since the logging start.
Cumulative GT Energy_0 (mWh)	

Table 3.1: Values provided on by the Intel Power Gadget on an i7-4500U

Another example is that not all tasks will make the CPU spend the same amount of power, if there is a power headroom the CPU might overclock itself to take advantage of this headroom. This technology has its benefits, but make the CPU or GPU performance more variable since this depends from many variables like temperature and current limits. When possible benchmarks with this technology enabled and disabled are performed and compared.

3.2.1 CPU

For CPU power consumption analysis the Intel Power Gadget for Linux was used. This tool enables easy logging various values obtained from the MSR registers from Intel CPUs (on Linux this requires that the *msr* kernel module to be loaded). Intel Power Gadget was chosen since only Intel CPUs were available for this work.

The Intel Power Gadget outputs a CSV file with the various values separated by time interval chosen by the user. A table explaining the obtained values from one of the test systems is provided, see 3.1. The provided values vary from CPU to CPU (some Intel CPUs don't have an integrated GPU for example). It should also be noted that if the computer system has more than one CPU this tool also provides the values for other present CPUs (eg multi socket workstations).

It should be noted that this output doesn't match what was present in the documentation analyzed in the previous chapter. There are no PP0 and PP1 values, in this case PP0 equals to IA, and PP1 equals to GT.

System Time	Elapsed Time (sec)	IA Frequency_0 (MHz)	Processor Power_0 (Watt)	IA Power_0 (Watt)	GT Power_0 (Watt)
13:24:21:595	1.0009	1547	3.9592	0.2958	0.1761
13:24:22:596	2.0022	1800	3.9665	0.3806	0.1591
13:24:23:598	3.0033	1800	4.1273	0.4153	0.1553

Table 3.2: Modified Intel Power Gadget sample output on an i7-4500U CPU

Since many of the outputs are not required, the Intel Power Gadget source code was modified to remove them. The *RDTSC* value and all cumulative values were removed. The cumulative values are then calculated using a spreadsheet according to the relevant time period in analysis. This was also required since some of the cumulative calculations were incorrect. The modified Intel Power Gadget results were validated against the non modified version as to make sure that no bugs were introduced.

To log the power consumption, the Intel Power Gadget is left running in the background and its output is stored to the computer storage in the form of a CSV file. The program reads the values each second.

A sample output from the modified Intel Power Gadget on one of the test systems is provided in the table 3.2.

From this logs it is possible to get an insight on how much power the CPU consumes on a particular instant while executing different tasks. To calculate the energy consumption a simple calculation must be done, multiplying the power consumption average by the running time gives the spent energy in Joules.

3.2.1.1 CPU speed control

Due to time and hardware availability constrains only Intel CPUs will be used. Since the tested CPUs all use the Intel pstate Linux kernel driver, the *pstate-frequency* utility will be used to manipulate the maximum and minimum frequencies. Since Intel uses SpeedStep (dynamic frequency scaling technology) the CPU clock frequency varies according to the load present on the CPU, this behavior will be studied when using the default CPU configurations.

Before running any benchmarks the possible frequency ranges are tested manually, since both the CPU or Intel pstate driver might ignore the defined frequency values (some frequencies steps might not be supported).

Then after obtaining the maximum and minimum percentage values required for the *pstate-frequency* utility these are noted with the respective frequency. Doing this permits to know that the settings are really doing what they are supposed to do. This also has the added benefit of permitting scripting the frequency changes.

3.2.2 GPU

For GPU power consumption three tools were found: the NVIDIA System Management Interface one and Intel Power Gadget. AMD CodeXL profiler also reports AMD GPU power consumption.

However on the available hardware none of them supported measuring power consumption for the dedicated GPU, only on the Intel integrated GPU was possible to measure power consumption.

3.2.3 GPU speed control

GPU speed control is not very different from the CPU speed control, but there are less power states to chose from. It is possible to change the running frequency of the GPU through some overclocking utilities but since the goal is not tweak each individual GPU (some GPUs even have this option locked) this will not be pursued.

NVIDIA has PowerMizer, a technology that automatically sets the power state according to the load imposed on the GPU. Some GPUs like the NVIDIA 980 GTX have 4 different performance levels and one boost state (similar in function to Intel Turbo Boost or AMD Turbo Core).

AMD GPUs also have multiple power states, for example the AMD R9 270X has 4 power states, a boost state is also provided (AMD PowerTune).

Intel integrated GPUs also have multiple power states, it should also be noted that they depend on the CPU load, since the TDP is shared between the CPU and GPU.

No research was made on AMD CPUs with integrated GPUs.

All analyzed manufacturers provide utilities to change the power settings, most of the times through performance profiles in the driver utility.

3.3 Performance Analysis

In this section the benchmarks for performance analysis and their respective configurations will be discussed.

3.3.1 Chosen Benchmarks

3.3.1.1 Intel Optimized Linpack

Linpack was chosen since it measures how fast a system of linear equations of the type Ax=b can be solved by the computer system.

Since only Intel CPUs are being tested, using the Intel Linpack benchmark should take better advantage of the CPU instead of using a generic version of the benchmark. However it should be noted that this makes the comparison with other manufacturers CPUs invalid, since Intel does not "grantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel" [Cora]. This should be taken in consideration for all benchmark utilities that are built using Intel Compilers.

Three configurations were made:

- Big: runs one test with problem size 20000. Chosen to simulate long jobs, avoiding job start and finish overheads (ex: memory allocations). The best results are expected in this test.

```
1  1 # number of tests
2  20000 # problem sizes
3  20000 # leading dimensions
4  4 # times to run a test
5  4 # alignment values (in KBytes)
```

Listing 3.1: Intel Linpack Big Configuration

- Small: runs one test with problem size 1000. Chosen to simulate small burst jobs, more overhead prone than the big test. Worse results are expected in this test.

```
1  1 # number of tests
2  1000 # problem sizes
3  1000 # leading dimensions
4  4 # times to run a test
5  4 # alignment values (in KBytes)
```

Listing 3.2: Intel Linpack Small Configuration

- Mixed: runs seven tests starting from 1000 to 20000 sizes. Includes both small and big tests, chosen to see the problem size effect of the obtained results.

```
7 # number of tests
1000 2000 5000 10000 15000 18000 20000 # problem sizes
1000 2000 5000 10000 15000 18000 20000 # leading dimensions
4 4 4 4 4 4 4 # times to run a test
4 4 4 4 4 4 4 # alignment values (in KBytes)
```

Listing 3.3: Intel Linpack Mixed Configuration

All configurations run each test four times so that in the case an erroneous output is present (for example while Intel Turbo Boost is active) it can be discarded. For the output to be considered valid at least 3 outputs must be consistent (time and GFlops values must be within 5%).

3.3.1.2 High Performance Linpack for GPUs

This was chosen so that GPU performance would be obtained in a similar way of the CPU performance. It should be noted the previous benchmark and this one are not comparable since the code base and compiler are not the same.

No configurations were made since due to the lack of time and the non availability of built binaries this benchmark was not used.

3.3.2 Outputs

3.3.2.1 Intel Optimized Linpack

The Intel Linpack benchmark has two output values of interest: the time taken and the GFlops value. Listing 3.4 shows a snippet of the Intel Linpack output. This output shows that four 20000 problems were solved, each problem has its own time and GFlops value.

```
Size   LDA    Align. Time(s)    GFlops   Residual     Residual(norm) Check
20000  20000  4      99.032     53.8625  3.466513e-10 3.068624e-02   pass
20000  20000  4      105.277    50.6675  3.466513e-10 3.068624e-02   pass
20000  20000  4      105.131    50.7381  3.466513e-10 3.068624e-02   pass
20000  20000  4      104.774    50.9106  3.466513e-10 3.068624e-02   pass
```

Listing 3.4: Intel Linpack output snippet

The outputs are then imported to a spreadsheet so that they can be further processed.

3.4 Simulated Platform Model

In this section the simulated platform model will be discussed. The utilized model is the SimGrid platform model but complemented since various shortcomings exist in it.

3.4.1 SimGrid Platform Model

SimGrid provides a platform model that is very network oriented, since it is not the goal of this dissertation to explore the networking aspect of SimGrid only a brief overview will be given.

While the SimGrid network modeling is very detailed, the actual computation modeling is not. This is an issue that will be discussed in 3.4.2.

A sample SimGrid platform file is provided at the listing 3.5. The provided sample describes a simple computer with a dedicated graphics card.

```
1  <?xml version='1.0'?>
2  <!DOCTYPE platform SYSTEM "http://simgrid.gforge.inria.fr/simgrid/simgrid.dtd">
3  <platform version="4">
4    <AS  id="AS0"  routing="Full">
5      <host id="CPU0" speed="50Gf"/>
6      <host id="GPU0" speed="38Gf"/>
7      <link id="PEG0" bandwidth="2GBps" latency="10us" sharing_policy="FATPIPE" />
8      <route src="CPU0" dst="GPU0">
9        <link_ctn id="PEG0"/>
10     </route>
11   </AS>
12 </platform>
```

Listing 3.5: Sample SimGrid Platform file

3.4.2 SimGrid Platform Model limitations

SimGrid defines the entities that actually do the computation themselves as *hosts*. The *hosts* as defined in SimGrid 3.13 are very simple, they rely on a *power* value and a core count as far as performance is concerned.

No actual power values exist. This might create a slight confusion at first, since what SimGrid calls power is not at all related to energy and instead represents the number of flops that a host core can compute. A problem arises with using flops as a performance measure unit, running on the same exact hardware different programs that do exactly the same (eg multiplying two matrices) may yield very different flops performance values.

The core count has recently added to SimGrid (added on version 3.13) and it is not yet validated. SimGrid assumes that if 4 cores are present, 4 tasks can be executed without interfering with each other what is not true, there are a lot of factors that influence this.

Factors such Intel Power Boost, TDP limits, cache access patterns, memory access times among many others influence the obtained performance.

With this in mind the SimGrid core count cannot be utilized due to the numerous issues that arise from using it (the speed value just gets multiplied by the core count).

SimGrid also ignores the architecture kind of the processing unit, so that, for example, a GPU cannot be distinguished from a CPU. This is very problematic since not all tasks can run on all architectures and task performance varies widely in different architectures. Since this dissertation needs to consider heterogeneous systems this is a problem that needs to be solved.

Another issue is the lack of power states. Nowadays many processing units have different power states with different performances. Since one of the goals of this dissertation is to be energy aware this is another problem that needs to be solved.

With this in mind SimGrid needs to be complemented as to better understand the underlying platform that it is simulating.

3.5 Task Graph Model

In this chapter the chosen task graph model will be explained. As the basis of the task graph is a directed acyclic graph, more commonly known as a DAG. No further details will be given since this already discussed in chapter 2.

3.5.1 SimGrid Task Model

SimGrid tasks are defined as *jobs*, they contain an unique identifier (*id*), a name-space and finally a name. Before going in to more detail it should be noted that valid DAX files are not accepted by SimGrid and that the provided examples fail to validate against the defined xml schema.

As far as computation time estimation they contain a runtime number that corresponds to the time it takes do complete the task on a host running at 4.2GFlops. This might pose a problem since it is assumed that the timings were made on a 4.2GFlops host for that particular task. Since different tasks behave differently on different platforms this is a dangerous assumption to make, with this in mind extra care will be taken dealing with this question.

Each task can have outputs and inputs from/to other tasks, each of them is represented on a separate *uses* tag. The link parameter defines if it is an input or an output. Also of hight importance is the size of the data to be transfered since that value is utilized to calculate the data transfer time.

3.5.2 SimGrid Task Graph Model

SimGrid uses a non standard DAX file as mentioned before. It focuses in the task description, the edges are obtained from the tasks themselves instead of being explicitly declared. A sample DAX is provided in the listing 3.6, this is a very simple example with three sequential tasks.

Since SimGrid doesn't allow modifications to the DAX file, all the required additions will be stored in a separate file, this is explored in detail in chapter 4.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <adag xmlns="http://pegasus.isi.edu/schema/DAX" xmlns:xsi="http://www.w3.org/2001/
    XMLSchema-instance" xsi:schemaLocation="http://pegasus.isi.edu/schema/DAX http
    ://pegasus.isi.edu/schema/dax-2.1.xsd" version="2.1" count="1" index="0" name="
    test" jobCount="3" fileCount="0" childCount="20">
3
4 <job id="ID00000" namespace="Test" name="T1" version="1.0" runtime="15">
5 <uses file="input_0" link="input" register="true" transfer="true" optional="false"
    type="data" size="10"/>
6 <uses file="output_0" link="output" register="true" transfer="true" optional="false
    " type="data" size="10"/>
7 </job>
8 <job id="ID00001" namespace="Test" name="T2" version="1.0" runtime="4">
9 <uses file="output_0" link="input" register="true" transfer="true" optional="false"
     type="data" size="10"/>
```

```
10 <uses file="output_1" link="output" register="true" transfer="true" optional="false
       " type="data" size="10"/>
11 </job>
12 <job id="ID00002" namespace="Test" name="T3" version="1.0" runtime="1">
13 <uses file="output_1" link="input" register="true" transfer="true" optional="false"
        type="data" size="10"/>
14 <uses file="output_2" link="output" register="true" transfer="true" optional="false
       " type="data" size="10"/>
15 </job>
16
17 <child ref="ID00001">
18 <parent ref="ID00000"/>
19 </child>
20 <child ref="ID00002">
21 <parent ref="ID00001"/>
22 </child>
23 </adag>
```

Listing 3.6: Sample DAX

In the DAX file tasks are represented by *job* tags and relationships between tasks are described by *child* tags (see the edges section).

The job attributes are listed in the table 3.3 with a short description of what they are. All attributes are mandatory.

Attribute	Description
id	Job (task) id.
namespace	Current task name space.
name	Task name.
version	Needs always to be "1.0", has no importance for the simulation.
runtime	Task execution time in seconds to execute on a 4.2 GFlops host.

Table 3.3: DAX job tag attributes

Edges are not directly described by the DAX file format, but are represented in the nodes themselves through the *file* attribute on each node. Both inputs and outputs are possible in each node, so to create an edge between two nodes the output of the source node must be the input of the destination node and this relationship must be represented by a *child* tag. The *uses* tag attributes are listed in table 3.4.

3.5.3 SimGrid Task Graph Model limitations

Not as bad as the platform model, the task graph model still has some limitations. The most important being the lack of platform awareness. With this in mind the task graph will be augmented so that platform awareness can be used, so that the supported platforms are known and the hopefully best platform can be recommended.

Attribute	Description
file	File name.
link	Defines if the file is an input or output.
register	Boolean defining whenver the file is registered on a replica catalog.
transfer	Boolean defining whenever the result should be transmitted or not.
optional	Boolean defining if the *file* is optional.
type	Defines the type of the output.
size	Data size for transmission calculations

Table 3.4: DAX file tag attributes

Two test DAGs will be created and other already existing DAGs will be used to test the scheduling algorithm.

3.5.4 Contech

Since its very useful to use real task graphs without having to make them manually, the Contech framework by Railing, Brian P. and Hein, Eric R. and Conte, Thomas M. will be analyzed.

Chapter 4

Scheduler

4.1 Proposed Algorithm

4.1.1 Introduction

In this section the implemented scheduler algorithm will be discussed. Since it is based on the HEFT algorithm with minor changes, a brief overview of the HEFT algorithm is given first. Then the implemented scheduler is detailed.

4.1.2 HEFT Algorithm

The HEFT (or Heterogeneous Earliest Finish Time) is a scheduling algorithm proposed by Topcuoglu et al. It is described in conjunction with the CPOP algorithm [THW02]. In the algorithm 1 the original pseudo code as proposed by Topcuoglu et al is transcribed.

The EFT value is calculated using the equation 4.1.

$$EFT(n_i, p_k) = StartTime(n_i, p_k) + ExecutionTime(n_i, p_k) \tag{4.1}$$

Algorithm 1 HEFT scheduling algorithm.

1: Set the computation costs of tasks and communication costs of edges with mean values.
2: Compute $rank_u$ for all tasks by traversing graph upward, starting from the exit task.
3: Sort the tasks in a scheduling list by non increasing order of $rank_u$ values.
4: **while** there are unscheduled tasks in the list **do**
5: Select the first task, n_i, from the list for scheduling.
6: **for** each processor p_k in the processor-set ($p_k \in Q$) **do**
7: Compute $EFT(n_i, p_k)$ value using the $insertion-based$ scheduling policy.
8: **end for**
9: Assign task n_i to the processor p_j that minimizes EFT of task n_i.
10: **end while**

4.1.3 HLEC Algorithm

The implemented scheduling algorithm is based on the HEFT algorithm. Two modifications were made to the algorithm.

The first modification was to make HEFT platform aware, ie: distinguish between a GPU and a CPU. This is important since each task must only be scheduled in the platforms that are supported by it.

The second modification was to change the EFT (Earliest Finish Time) to LEC (Lowest Energy Cost). This change was made so that instead of preferring the fastest execution time, the lowest energy consumption is preferred. With this modification the algorithm name became HLEC.

The modified pseudo code can be seen in the algorithm 2.

The equation 4.2 shows how the LEC value can be calculated, by minimizing the LEC value the best host in terms of power consumption will be selected.

$$LEC(n_i, p_k) = EnergySpentSoFar(p_k) + EnergyRequired(n_i, p_k) \tag{4.2}$$

In the implementation the utilized equation 4.3 is not the same as the equation 4.2, since due to the way that SimGrid works it was not possible to properly calculate the power consumption of a host so far, with this in mind the equation was changed to assume that the host was always in use before. See the chapter 6 for more details.

$$LEC(n_i, p_k) = Power(p_k) * (StartTime(n_i, p_k) + ExecutionTime(n_i, p_k)) \tag{4.3}$$

Algorithm 2 Modified HEFT scheduling algorithm (HLEC).

1: Set the computation costs of tasks and communication costs of edges with mean values.
2: Compute $rank_u$ for all tasks by traversing graph upward, starting from the exit task.
3: Sort the tasks in a scheduling list by non increasing order of $rank_u$ values.
4: **while** there are unscheduled tasks in the list **do**
5: Select the first task, n_i, from the list for scheduling.
6: **for** each processor p_k in the task compatible processor-set ($p_k \in QT$) **do**
7: Compute $LEC(n_i, p_k)$ value using the $insertion - based$ scheduling policy.
8: **end for**
9: Assign task n_i to the processor p_j that minimizes LEC of task n_i.
10: **end while**

4.2 Implementation

In this section the scheduler implementation will be discussed. The code will not be explained in detail since the developed program is somewhat large.

4.2.1 Program structure

The program was initially built using the C language due to compatibility issues while linking the SimGrid library with the C++ based application. During the development process SimGrid 3.13 was released and fixed the C++ compatibility problem, so the program was ported to C++ in order facilitate the development process.

After testing the new library version it was obvious that some methods were missing. Those methods were added to the program itself so that no unnecessary modifications to the SimGrid code were done.

A object oriented approach was utilized, with this the program was separated in different classes.

There are 6 different parameters defined in this file.

1. *PowerState* Contains the power state information, including frequency and power rating.
2. *AttributeHost* Contains all the additional host attributes not supported by SimGrid.
3. *AttributeTask* Contains all the additional task attributes not supported by SimGrid.
4. *Config* Class with static variables containing the program configuration.
5. *Output* Class with static methods that provide scheduling result output, supporting CSV, Jedule and console output.
6. *Scheduler* The most important class: contains the scheduler, contains the implemented algorithms (HEFT and HLEC) and all auxiliary methods required.
7. *Simulation* Wrapper for the simulation, responsible for initializing the simulation environment and loading the task graph and platform files.

4.3 Input Files

In this section the input files that the implemented scheduler uses are described.

4.3.1 Configuration Files

The developed application needs two configuration files, the global configuration file and the input configuration file. Both are detailed bellow.

4.3.1.1 Global configuration file

This file stores global configurations common to all possible inputs (nothing directly affects the simulation). It's name must be config.json and must be on the DAG_Scheduler binary folder. The default configuration file is shown in the listing 4.1.

```
{
  "DAG_Scheduler_params": [{
    "platformsDir": "platforms/",
    "platformAddonsDir": "platforms_addon/",
    "daxDagsDir": "dax_dags/",
    "daxDagAddonsDir": "dax_dags_addon/",
    "outputDir": "scheduler_output/"
  }]
}
```

Listing 4.1: Default config.json

The file is self explanatory and since it is not relevant for anything other than the initial configuration it will be not detailed any further.

4.3.1.2 Input configuration file

This file defines the inputs that the program will use. It defines what platform and graph files to load and the settings for the scheduler.

A sample input file is show in the listing 4.2

```
{
  "DAG_Scheduler_inputs": [{
    "platform": "S440.xml",
    "daxDag": "CyberShake_100.xml",
    "algorithm" : "LEEC",
    "powerBudget": 60,
    "useFastestSpeed" : false,
    "speedThresholdPercent" : 60
}]
}
```

Listing 4.2: Sample input.json

There are 6 different parameters defined in this file.

1. *platform* The platform file to load.
2. *daxDag* The graph file to load.
3. *algorithm* The algorithm to use, both LEC and HEFT are valid options here.
4. *powerBudget* The power budget in watts. This is currently ignored, see further work.
5. *useFastestSpeed* Boolean that determines if the highest performing power state is selected or the optimal power state is selected.
6. *speedThresholdPercent* Integer that must be within 1 and 100 that defines the minimum power state possible.

If no parameter is passed the program will attempt to use a input.json file in the program folder, if passed it will use that file instead. This is used for scripting.

4.3.2 Graph

Here the two files required for the graph to be loaded will be discussed. Two files had to be used since SimGrid doesn't allow any extra fields on the DAX file (as of SimGrid 3.13 if any extra field is found, the graph loading is aborted and exit is invoked killing the application).

The first file is the DAX graph that is widely used in many examples, it was chosen since its more complete and easy to work with than the other SimGrid supported (dotty) format. It should be noted that the DAX file format as used by SimGrid fails the XML Schema (XSD) Validation, so it is not standard and it's compatibility with other software is unknown to the author.

The second file is a complementary file that stores extra parameters needed for the developed scheduler. This file uses JSON to store extra fields. With this approach no SimGrid code had to be modified and the original DAX files can be used with the oficial SimGrid version and other tools that support it since it still respects its original specifications.

4.3.2.1 DAX File

This file defines the task graph, it contains all the nodes (tasks) and edges (dependencies).

4.3.2.2 JSON File

In this file each node (task) must have an entry with the required extra fields. In the JSON file all extra attributes are defined, all nodes in the DAX file must also be present in this file (if using an extra JSON file at all). This works as a complement to the DAX file.

The extra job attributes are listed in table 4.1 with a short description of what they are, again all atributes are mandatory.

Attribute	Description
jobID	Job (task) id, must match a job ID in the DAX file.
preferedPlatform	Preferred platform for the task execution.
supportedPlatforms	An JSON array of the supported platforms for the task to execute.

Table 4.1: JSON job attributes

4.3.3 Platform

As on the graph two files are required. Once again the reason is the same from above (prevent SimGrid source code modifications and loss of compatibility).

The first file is the platform definition that SimGrid supports.

The second file is a complementary file that stores extra parameters that are required for the developed scheduler. As before JSON was chosen as the file format.

4.3.3.1 XML File

The XML file describes the platform that SimGrid will simulate. A more detailed description of this platform file is available in the SimGrid documentation. One thing that must be noted is that the host speed value defined on this file is ingnored since it is calculated from the complementary JSON file.

4.3.3.2 JSON File

In this file the extra fields for the platform are defined. All existing platforms on the XML file must also be present on this file. The most important fields present in this file are the idlePower and the powerStates. Without these fields it would be impossible to calculate power consumption values. The powerStates are also used to find the best operating point for that element in the platform. Table 4.2 contains all possible attributes.

Attribute	Description
hostID	Host id, must match a host ID in the xml platform file.
platform	Platform kind (CPU, GPU, etc..)
vendor	Vendor name (AMD, Intel, NVIDIA, etc..)
model	Model name
coreCount	Core count. Does not include logical cores (Hyper-threading, etc..)
idlePower	The power spent in idle by the host in watts.
maximumPower	The maximum power that the host can drawn in watts.
frequencyToGFlop	The approximated frequency to flops ratio.
powerStates	A json array of power states, consisting in id, power and frequency.

Table 4.2: JSON host attributes

A simplified example can be seen at listing 4.3, only one power states is shown.

```
{
  "platform_attributes": [{
    "hostID": "CPU0",
    "platform": "CPU",
    "vendor": "Intel",
    "model": "i7-4500U",
    "coreCount": 2,
    "idlePower": 3.13,
    "maximumPower": 16.0,
    "frequencyToGFlop" : 0.028,
    "powerStates": [{
      "id": "P00",
      "power": 7.14,
      "frequency": 800
    }, {
    ...
  }
  }]
}
```

Listing 4.3: Sample platfrom addon json file

4.4 SimGrid library modification

As to avoid modifications in the SimGrid library code when possible new methods were written and called by the developed program. But since SimDag has some methods that are very difficult to override like the scheduling function, one modification had to be made.

4.4.1 Host speed change in runtime

Due to the way that SimGrid works, the host speed needed to be modified in runtime. The problem was that SimGrid gets the speed from the platform xml file and doesn't provide methods to change the speed. As to keep compatibility the fastest speed should be set on the original xml file, but this is not required since the developed program ignores this value and calculates it from the json file values.

At least as of SimGrid 3.13 that is not possible, so the SimGrid library needed to be modified in order to support this. Since SimGrid is an open source project the source code was modified by the author. The modification can be found in the appendix.

4.5 Conclusions

Many low level details are omitted, since in order to comprehend the scheduler most of the developed code is not required to be understood.

However some difficulties arose during the implementation phase, either by the lack of documentation or due to the way that the SimGrid was programed.

Chapter 5

Results

In this chapter the obtained results are discussed.

5.1 Task Graph

Before starting with the scheduler development it was analyzed if it was feasible to generate task graphs using already existing tools.

5.1.1 Contech

The only analyzed tool was Contech. The code is available online [Rai], and was downloaded and compiled by the author. However since it was unusable at the time when it was tested by the author (either by some issue with the author setup or by some issue with the tested version).

Issues with the build system were fixed and the tool was built, however when attempting to profile a simple program it failed to produce a usable output. With this in mind it was discarded.

Due to time that it took to rebuild every time a fix was made the author ended spending more time than expected and no other tool was analyzed.

5.1.2 Existing examples

Many examples of task graphs in the DAX format were found. Some were used to test the scheduler, but since there was no information on what platform the code ran they are only used to test the scheduler while ignoring all platforms differences.

5.1.3 Manually created

In order to test the additions to the DAX task graph two task graphs were created manually. One with three sequential tasks and another with a parallel task section.

5.1.4 Simulation Results

The obtained results from the simulation are shown here, all simulations used the same platform configuration.

Due to the size of the utilized task graphs only the scheduling output from the created task graphs will be shown.

On figure 5.1 the sequential test can be seen and on figure 5.2 the parallel test can be seen (since both have the same result either using HEFT or HLEC, so only the HLEC result is shown). The processor 0 is the CPU and 1 is the GPU.

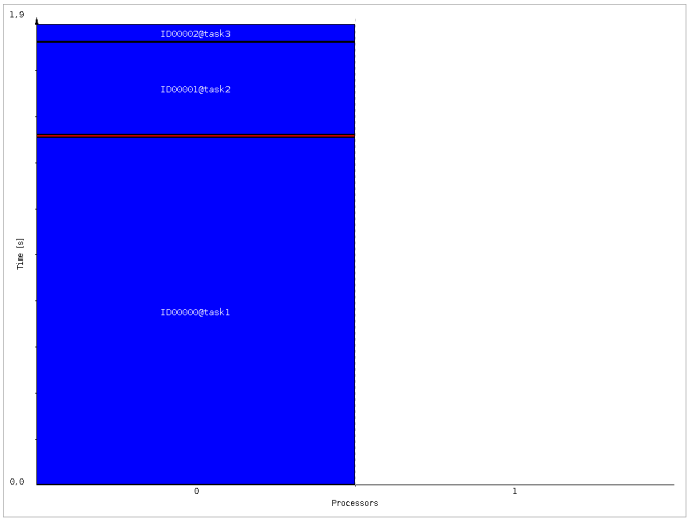

Figure 5.1: Sequential Test HLEC scheduler result

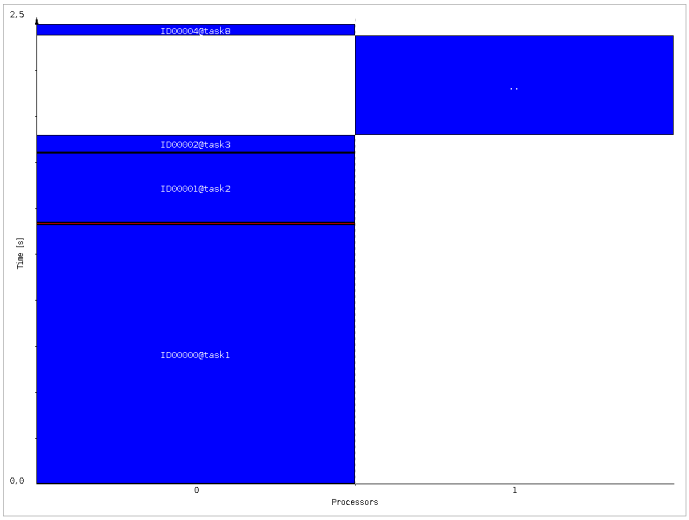

Figure 5.2: Parallel Test HLEC scheduler result

Since the previous graphs had the same result using both algorithms, another graph was selected to highlight the main difference between both algorithms. The HEFT scheduler result is presented in the figure 5.3, and the HLEC in 5.4.

Since HLEC main goal is to save energy it will attempt to avoid using the dedicated GPU in the test platform and since the dedicated GPU idles at 0 watts this ends up saving energy, once again at the expense of worse runtimes.

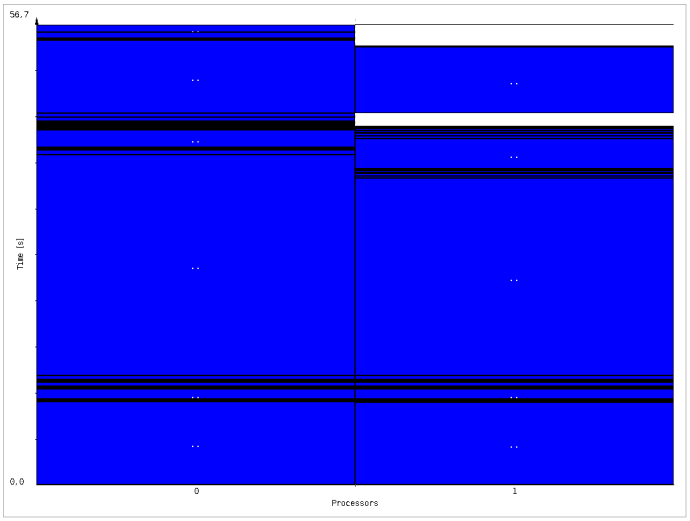

Figure 5.3: Montage 100 HEFT scheduler result

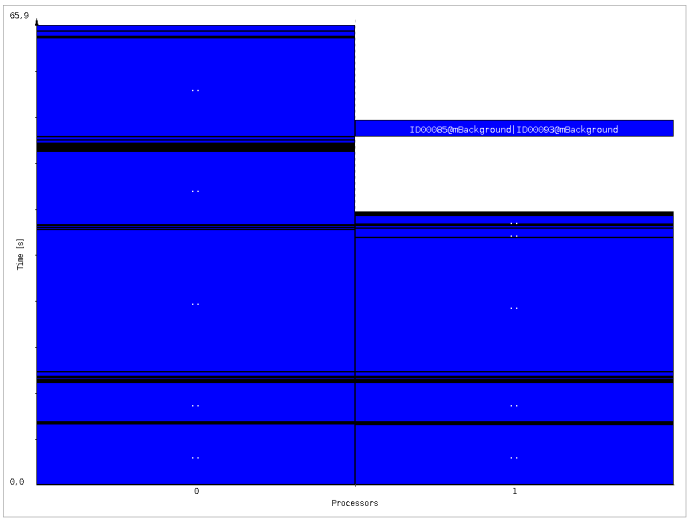

Figure 5.4: Montage 100 HLEC scheduler result

Energy results are provided in table 5.1.

5.1.5 Comparison between HEFT and HLEC algorithms

Here the obtained results will be discussed. Before comparing results it should be noted that the HEFT algorithm has the platform awareness modification, since without it the results can not be compared.

From the table 5.1 it is possible to see that enabling the efficient speed configuration makes both algorithms spend less energy at the expense of runtime.

Graph	Algorithm	Speed Configuration	Total execution time (Seconds)	Total energy spent (Joules)
Sequential Test	HEFT	Efficient	1,89	22,78
	HEFT	Fast	1,59	24,21
	HLEC	Efficient	1,89	22,78
	HLEC	Fast	1,59	24,21
Parallel Test	HEFT	Efficient	2,52	33,67
	HEFT	Fast	2,21	35,17
	HLEC	Efficient	2,52	33,67
	HLEC	Fast	2,21	35,17
CyberShake 100	HEFT	Efficient	187,17	4572,78
	HEFT	Fast	173,56	4633,52
	HLEC	Efficient	224,41	4378,41
	HLEC	Fast	209,90	4446,25
Epigenomics 100	HEFT	Efficient	21050,38	546303,03
	HEFT	Fast	19129,84	561775,34
	HLEC	Efficient	22247,78	540058,20
	HLEC	Fast	20023,01	550227,34
Montage 100	HEFT	Efficient	56,66	1466,12
	HEFT	Fast	51,07	1492,58
	HLEC	Efficient	65,91	1412,64
	HLEC	Fast	58,87	1445,11

Table 5.1: Scheduler time and energy outputs

Using the HEFT algorithm on the fast configuration as a baseline, it is possible to compare the different algorithm configurations on figure 5.5.

In this graph the configurations are grouped for each of the 5 different graphs, since the HEFT algorithm on the fast configuration was used as baseline it doesn't appear in the graph as its values would always be zero. For each configuration two bars exist, one with the percentage of energy saved and another with the percentage of increased runtime. For example on the Montage 100 graph it is possible to see that the LEC-Efficient configuration there was a 22,7% increase in runtime and a 5.8% decrease in energy consumption.

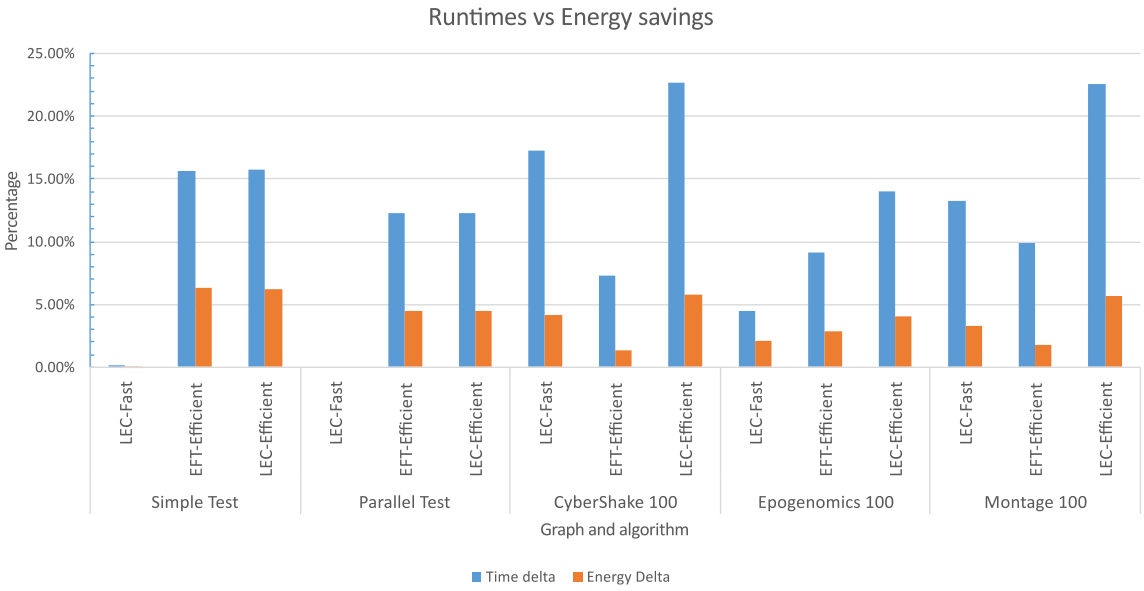

Figure 5.5: SimGrid Runtime vs Energy results

It is possible to conclude that the HLEC algorithm most of the times reduces the energy spent while doing the same tasks as the HEFT algorithm, also it is possible so see that by reducing the platform CPU speed the runtime increases but the spent energy decreases.

5.2 Hardware Performance and Energy Results

In this section the results obtained from running benchmarks on hardware are presented. The main goal was to assess the performance and energy consumption of different hardware configurations.

5.2.1 CPU only, Intel Core i7-4500U

For this benchmarks only the CPU was used. The test computer is a laptop, it uses Ubuntu 16.04 as its operating system (vanilla install, without any modifications). The integrated GPU is always enabled and in use. It should be noted that this CPU being an ULV CPU has a low TDP value of 15 Watts, shared between the CPU and internal GPU.

5.2.1.1 CPU Frequency Control

The table 5.2 contains the used steps on the test CPU while under load running the Intel Linpac benchmark. The obtained frequency values are the average of both cores, the PState setting is the *pstate* driver tunable parameters and the PState reported frequencies is what the *pstate* driver reports.

It should be noted that both the Default and 1800Mhz settings make use of the Intel Turbo Boost. All configurations with the exception of the *Default* and *no turbo* had its lowest speed locked. This is not required for the maximum speed control, but was done to stabilize the CPU frequency.

From the obtained values it becomes obvious that disabling the Turbo Boost function makes the CPU work at a lower frequency than what Intel guarantees that it work on regular working conditions (sufficient power, no overheating). With Turbo Boost disabled the obtained average frequency corresponds to the 1700Mhz configuration average frequency.

Setting	Obtained frequency values		PState setting		PState reported frequencies	
	Maximum	**Average**	**Minimum**	**Maximum**	**Minimum**	**Maximum**
Default	2700	1900	26%	100%	800	3000
1800	2700	1895	61%	62%	1830	1860
No turbo	1702	1699	26%	100%	800	3000
1700	1701	1699	58%	59%	1740	1770
1600	1601	1599	54%	55%	1620	1650
1500	1501	1499	50%	51%	1500	1530
1400	1401	1399	47%	48%	1410	1440
1300	1301	1299	44%	45%	1320	1350
1200	1201	1199	41%	42%	1230	1260
1100	1101	1099	37%	38%	1110	1140
1000	1001	999	34%	35%	1020	1050
900	901	899	31%	32%	930	960
800	801	799	26%	27%	800	810

Table 5.2: PState settings on an i7-4500U CPU (Linpack, 2 cores active)

5.2.1.2 Intel Linpack benchmark

Here the results obtained by running the Intel optimized linpack benchmark are shown and discussed.

5.2.1.3 Big configuration

The big configuration provides a long, stable workload for the CPU. The figure 5.6 shows the power drawn versus the total energy drawn by the CPU.

In this graph it is possible to see that the power consumption varies according to the CPU frequency (as expected). The most prominent difference was noted in the CPU cores (IA Power) power consumption. Also worth noting that the uncore power consumption doesn't vary much and that the internal GPU was most likely powered off.

The main conclusion to take from this graph is that the most efficient power state is the one that has the frequency set to 1600Mhz, as seen by the spent energy line.

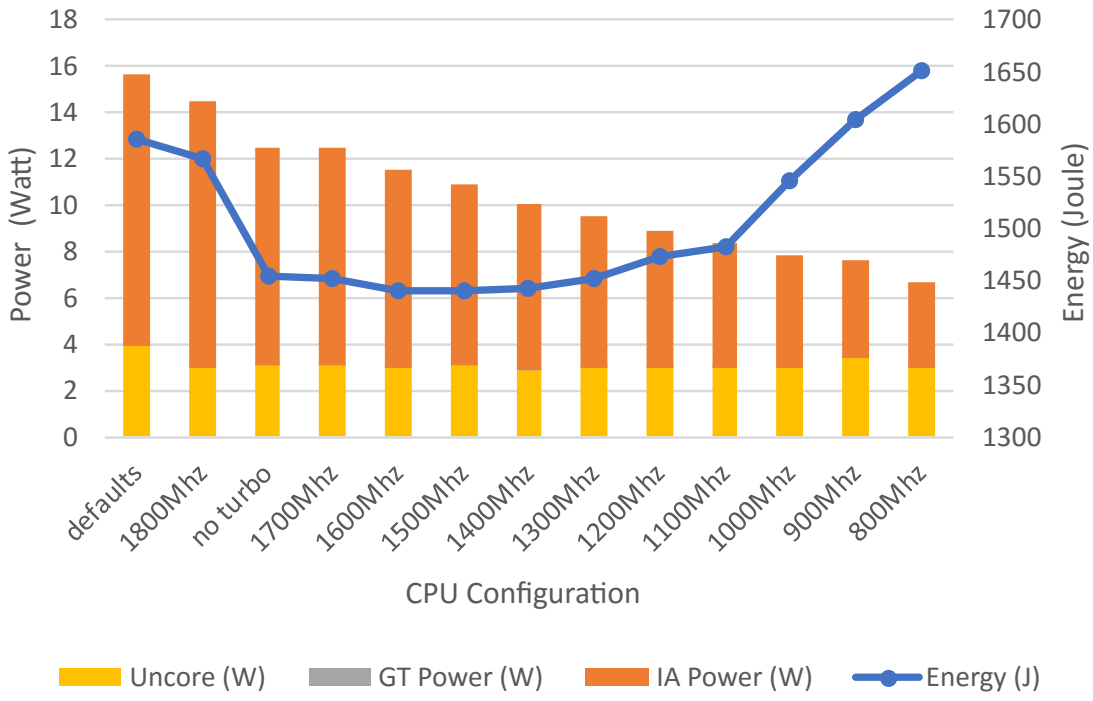

Figure 5.6: Linpack Power vs Energy benchmark results (big configuration)

In the figure 5.7 the execution time versus the energy consumption is shown. With this graph it is possible to see that runtime does not increase significantly with the frequency reduction, but the energy does decrease a bit. It is also worth noting that decreasing the frequency too much reduces efficiency.

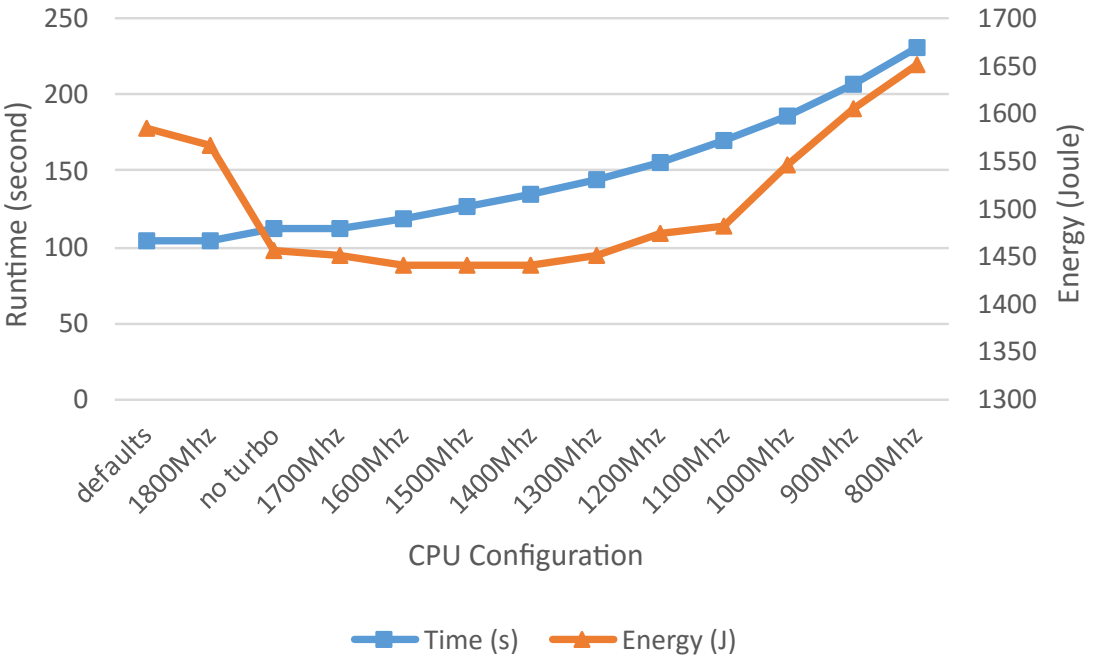

Figure 5.7: Linpack Execution time vs Energy benchmark results (big configuration)

5.2.1.4 Small configuration

The small configuration provides a very short workload for the CPU. It exposes a problem with this power consumption logging methodology, since the worst case runtime is well under one second (0.046 seconds) and the logging is done on a second by second basis the power and energy consumption most likely will not match what in reality was drawn. Due to this issue no further analysis was made.

5.2.1.5 Mixed configuration

The mixed configuration starts with very short workloads and progressively increases the workload until it reaches the big configuration problem size.

The figure 5.8 shows the power drawn versus the total energy drawn by the CPU. This somewhat matches the obtained results on the big benchmark configuration, the main difference is that the most efficient power state is the one with the 1500Mhz frequency setting.

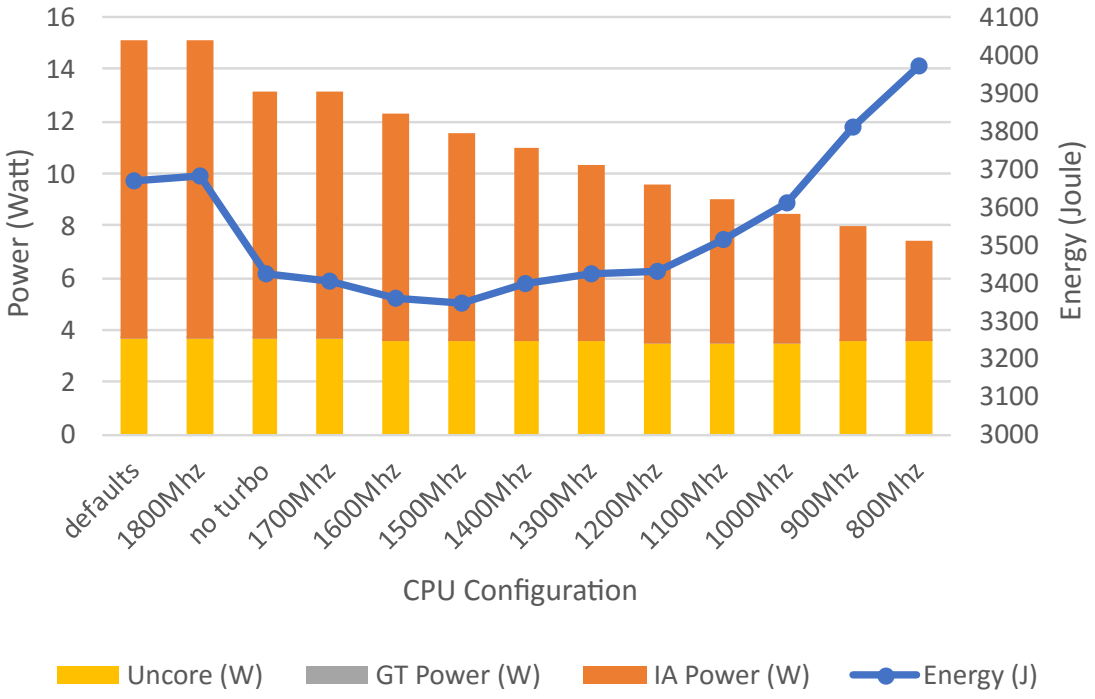

Figure 5.8: Linpack Execution time vs Energy benchmark results (mixed configuration)

The figure 5.9 shows the execution time versus the energy consumption. The same conclusions taken on the big frequency apply.

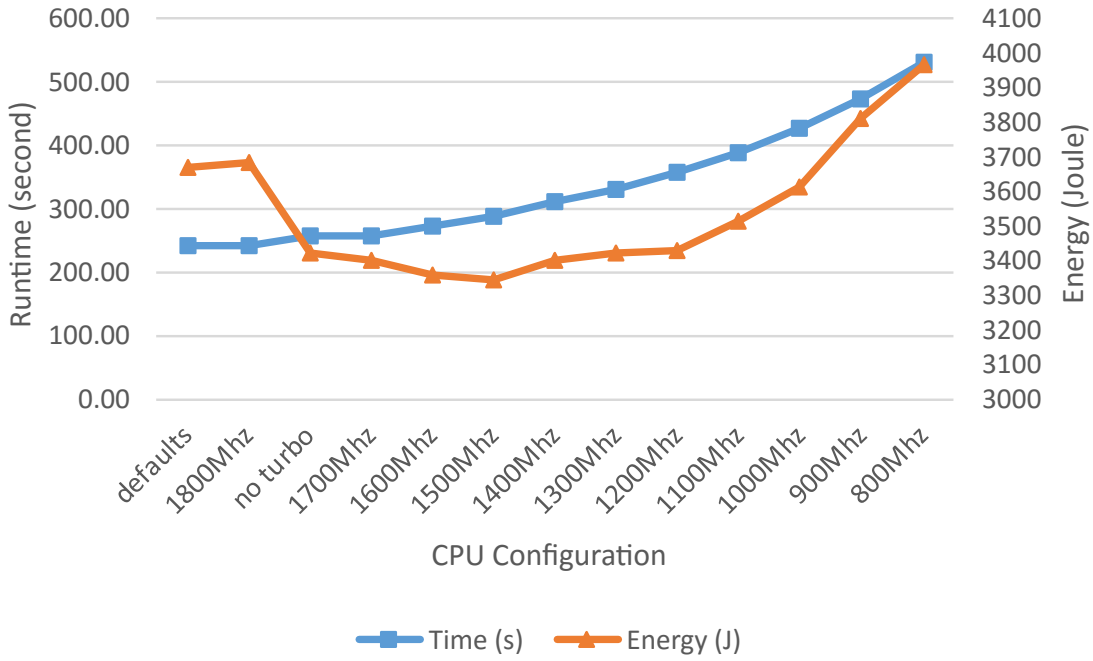

Figure 5.9: Linpack Power vs Energy benchmark results (mixed configuration)

5.2.1.6 Conclusions

Using only the big and mixed benchmark results we can easily see that on this particular CPU-benchmark combination that disabling the Intel Turbo Boost increases run time by 8.61% but decreases the energy consumption by 8.22% in the best case (see table 5.3 for more details). Lowering the CPU clock too much makes the energy consumption increase more than the fastest speed, this is mostly due to the uncore CPU power consumption that is always present even when the CPU is in idle.

This benchmarks also shows that using Ubuntu in text mode only, makes the integrated GPU report 0 watts of power.

	Big		Small		Mixed	
Config	**Time**	**Energy**	**Time**	**Energy**	**Time**	**Energy**
defaults	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1800Mhz	0.34%	−1.11%	25.97%	15.89%	0.33%	0.43%
no turbo	8.61%	−8.22%	1.30%	42.75%	6.79%	−6.60%
1700Mhz	8.43%	−8.45%	25.97%	17.24%	6.48%	−7.20%
1600Mhz	14.75%	−9.11%	31.17%	17.68%	12.34%	−8.33%
1500Mhz	21.83%	−9.14%	37.66%	18.88%	19.12%	−8.79%
1400Mhz	29.99%	−9.06%	42.86%	20.00%	27.52%	−7.30%
1300Mhz	39.56%	−8.42%	51.95%	24.62%	36.85%	−6.56%
1200Mhz	50.66%	−7.07%	64.94%	33.08%	47.39%	−6.40%
1100Mhz	63.37%	−6.50%	74.03%	41.65%	60.59%	−4.13%
1000Mhz	79.73%	−2.55%	88.31%	43.90%	76.04%	−1.44%
900Mhz	99.16%	1.23%	111.69%	55.23%	95.53%	3.87%
800Mhz	123.33%	4.19%	131.17%	62.75%	119.31%	8.26%

Table 5.3: Time and energy differences between frequencies, i7-4500U CPU

5.2.2 Test Platform Model

The test platform was modeled with the results obtained from the previous benchmarks. The platform is a laptop computer and has an Intel 4500U CPU, a Intel HD Graphics 4400 GPU and a Radeon 8670M GPU. The integrated GPU was ignored in the platform model since the TDP is shared with the CPU.

Since no performance values were obtained from the AMD GPU, the manufacturer specified values were used, 39 GFLOPS (Double-precision) [AMDa].

It is important to note that no thermal throttling was observed during the tests.

The produced and later utilized files are provided in the appendix A.

Chapter 6

Conclusions and Further Work

6.1 Attained Goals

It was possible to implement a scheduler using the SimGrid as its foundation, create a platform model that is power state and energy aware, also an scheduling algorithm was adapted as to be energy aware.

Regarding previous studies it was possible to verify the analysis on recent Intel CPU power consumptions (from Sandy Bridge and up).

The developed scheduler uses the power consumption values obtained in the benchmarks and uses them to calculate the energy consumption values of each simulation. These values are useful while testing different algorithms and task scheduling strategies.

6.2 Study limitations and further work

In this section both the study limitations and further work will be discussed. For convenience both are in separate subsections.

6.2.1 Limitations

Like all studies, some limitations exist, these are detailed bellow.

As for the algorithm, tests are limited mostly due to the fact that SimGrid was used. SimGrid quickly became an hindrance since it imposes many limitations, this was specially a problem when extending SimGrid. It should be noted that SimGrid is currently going through major architectural changes, including a migration to C++.

The implemented scheduler only supports two algorithms and one is heavily based on the other, this limits the test options.

Regarding the platform simulation most limitation lie on the lack of test platforms and benchmarks. No AMD platforms or ARM based CPUs were tested. No OpenCL benchmarks were run, CUDA benchmarks were run but since the GPU where they were executed didn't support power measurements they were not further explored.

Another issue with the platform simulation is that only the IC package of a CPU or GPU is considered, this means that energy spend on cooling, storage drives, network adapters, power losses, memory and plenty of other components present in computer systems are ignored.

Since the task graph can be comprised of program functions, one limitation is that the cost is not calculated and it is only used the supplied value in the task graph.

6.2.2 Further Work

Here possible future work will be presented.

6.2.2.1 Algorithm

Algorithm wise, more testing is needed and improvements are also needed, specially regarding the fact that the algorithm is greedy and incapable of making short term sacrifices for long term benefits.

Since many platforms nowadays have the CPU and GPU in the same package and share a total TDP value, the algorithm should be aware of that value and manage how much CPU or GPU should use.

Another interesting thing to improve is the addition of other computer system auxiliary devices to the energy awareness like cooling.

6.2.2.2 Scheduler Implementation

As far implementation goes the host power consumption issue needs to be solved. This issue makes the algorithm behave badly on some cases (one typical example are dual GPU systems that can power off unused GPUs completely).

Other algorithms can also be implemented and tested with the developed program.

The TDP sharing between CPU and GPU can also be implemented in conjunction with the proposed algorithm improvement.

Currently the power budget does nothing in the code, since it is an important aspect of power management it is of great interest to be implemented in the future.

As mentioned in the limitations it should be interesting to implement a task cost estimation program.

6.2.2.3 Benchmarks

Further benchmarks should be run with more power data rich platforms. It could also prove to be interesting to physically measure the power consumption of a computer system and compare the values with both the simulation output and the software counters output.

It also should be useful to test different versions of the same benchmark, for example running the same benchmark with and without using vectorization, and see what is the most efficient version.

References

[AB14] Hamid Arabnejad and Jorge G. Barbosa. A budget constrained scheduling algorithm for workflow applications. *Journal of Grid Computing*, 12(4):665–679, 2014.

[AMDa] Inc. Advanced Micro Devices. Amd radeon^{TM} hd 8000m series gpu specifications.

[AMDb] Inc. Advanced Micro Devices. Codexl - gpuopen.

[AMDc] Inc. Advanced Micro Devices. Codexl - powerful debugging, profiling & analysis - amd.

[APM13] Caitriana Niholson A. Paul Millar. Optorsim download | sourceforge.net, May 2013.

[CCSF+06] David G. Cameron, Ruben Carvajal-Shiaffino, Jamie Ferguson, A. Paul Millar, Caitriana Niholson, Kurt Stokinger, and Floriano Zini. *OptorSim v2.1 Installation and User Guide*, October 2006.

[CGL+14] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson, and Frédéric Suter. Versatile, scalable, and accurate simulation of distributed applications and platforms. *Journal of Parallel and Distributed Computing*, 74(10):2899–2917, June 2014.

[Cora] Intel Corporation. Intel® math kernel library benchmarks (intel® mkl benchmarks) | intel® software.

[Corb] Intel Corporation. Running average power limit – rapl | 01.org.

[Corc] NVIDIA Corporation. Nvidia management library (nvml).

[Cord] NVIDIA Corporation. Nvidia system management interface.

[Cor11] Intel Corporation. Intel(R) 64 and IA-32 Architectures Software Developer's Manual, Combined Volumes. *Architecture*, December 2011.

[Cor14] Sandia Corporation. Papi, 2014.

[Cor15a] Intel Corporation. Intel power gadget, December 2015.

[Cor15b] Intel Corporation. Measuring power on intel® xeon phi^{TM} product family devices, March 2015.

[Cor15c] Sandia Corporation. Power api for hpc: Standardizing power measurement and cont, 2015.

[Cor16a] NVIDIA Corporation. Cuda gpus, February 2016.

[Cor16b] Sandia Corporation. Papi, 2016.

[EB15] Nikhil Jain Eric Bohm, Phil Miller. Parallel programming laboratory, 2015.

[GB02] T. Giuli and M. Baker. Narses: A Scalable Flow-Based Network Simulator. *eprint arXiv:cs/0211024*, November 2002.

[Giu13] TJ Giuli. Narses network simulator download | sourceforge.net, March 2013.

[Gro15] Khronos Group. Opencl 2.1 and spir-v 1.0 launch, November 2015.

[ND10] John Nickolls and William J. Dally. The gpu computing era. *IEEE Micro*, 30(2):56–69, 2010.

[OPPF11] Simon Ostermann, Kassian Plankensteiner, Radu Prodan, and Thomas Fahringer. *GroudSim: An Event-Based Simulation Framework for Computational Grids and Clouds*, pages 305–313. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[Rai] Brian P. Railing. Github - bprail/contech: The contech analysis framework provides the means for generating and analyzing task graphs that enable computer architects and programmers to gain a deeper understanding of parallel programs.:.

[RHC15] Brian P. Railing, Eric R. Hein, and Thomas M. Conte. Contech: Efficiently generating dynamic task graphs for arbitrary parallel programs. *ACM Trans. Archit. Code Optim.*, 12(2):25:1–25:24, July 2015.

[RRB+08] Shane Ryoo, Christopher I. Rodrigues, Sara S. Baghsorkhi, Sam S. Stone, David B. Kirk, and Wen-mei W. Hwu. Optimization principles and application performance evaluation of a multithreaded gpu using cuda. In *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '08, pages 73–82, New York, NY, USA, 2008. ACM.

[RRS+14] Thomas Rauber, Gudula Rünger, Michael Schwind, Haibin Xu, and Simon Melzner. Energy measurement, modeling, and prediction for processors with frequency scaling. *The Journal of Supercomputing*, 70(3):1451–1476, 2014.

[SL] Porto SPeCS Lab, FEUP. Antarex: Project description.

[Tea04] DataGrid Project Team. The datagrid project, March 2004.

[Tea15] Da SimGrid Team. Introduction to simgrid, June 2015.

[Tea16] Da SimGrid Team. Simgrid home page, May 2016.

[THW02] Haluk Topcuoglu, Salim Hariri, and M Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *Parallel and Distributed Systems, ...*, 13(3):260–274, 2002.

[Yam] Peter Kenji Yamanaka. Pstate-frequency.

[YBR08] Jia Yu, Rajkumar Buyya, and Kotagiri Ramamohanarao. *Workflow Scheduling Algorithms for Grid Computing*, pages 173–214. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

Appendix A

SimGrid Platform Files

A.1 XML File

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "http://simgrid.gforge.inria.fr/simgrid/simgrid.dtd">
<platform version="4">
 <AS  id="AS0"  routing="Full">
   <host id="CPU0" speed="50Gf"/>
   <host id="GPU0" speed="38Gf"/>
   <link id="PEG0" bandwidth="2GBps" latency="10us" sharing_policy="FATPIPE" />
   <route src="CPU0" dst="GPU0">
      <link_ctn id="PEG0"/>
   </route>
 </AS>
 </platform>
```

Listing A.1: Test Platform SimGrid XML File

A.2 JSON File

```
{
    "platform_attributes": [{
        "hostID": "CPU0",
        "platform": "CPU",
        "vendor": "Intel",
        "model": "i7-4500U",
        "coreCount": 2,
        "idlePower": 3.13,
        "maximumPower": 16.0,
        "frequencyToGFlop" : 0.028,
        "powerStates": [{
```

```
            "id": "P00",
            "power": 7.14,
            "frequency": 800
        }, {
            "id": "P01",
            "power": 7.78,
            "frequency": 900
        }, {
            "id": "P02",
            "power": 8.30,
            "frequency": 1000
        }, {
            "id": "P03",
            "power": 8.76,
            "frequency": 1100
        }, {
            "id": "P04",
            "power": 9.44,
            "frequency": 1200
        }, {
            "id": "P05",
            "power": 10.05,
            "frequency": 1300
        }, {
            "id": "P06",
            "power": 10.71,
            "frequency": 1400
        }, {
            "id": "P07",
            "power": 11.42,
            "frequency": 1500
        }, {
            "id": "P08",
            "power": 12.13,
            "frequency": 1600
        }, {
            "id": "P09",
            "power": 12.93,
            "frequency": 1700
        }, {
            "id": "P10",
            "power": 15.31,
            "frequency": 1900
        }]
    }, {
        "hostID": "GPU0",
        "platform": "GPU",
        "vendor": "AMD",
        "model": "Radeon",
```

```
61          "coreCount": 384,
62          "idlePower": 0,
63          "maximumPower": 15,
64          "frequencyToGFlop" : 0.05,
65          "powerStates": [{
66              "id": "P00",
67              "power": 15,
68              "frequency": 775
69          }]
70      }]
71  }
```

Listing A.2: Test Platform JSON add-on

Appendix B

Benchmark results

B.1 Performance - Linpack

Size / LDA	defaults		1800		no turbo		1700	
	Time	GFlops	Time	GFlops	Time	GFlops	Time	GFlops
20000	99,032	53,862	101,705	52,447	112,456	47,433	112,302	47,498
	105,277	50,667	104,582	51,004	112,455	47,433	112,290	47,503
	105,131	50,738	104,664	50,964	112,522	47,405	112,290	47,503
	104,774	50,911	104,684	50,955	112,462	47,431	112,270	47,512
Average	103,553	51,545	103,909	51,343	112,474	47,426	112,288	47,504
STD DEVI	2,6169	1,3411	1,2729	0,6379	0,028	0,0118	0,0115	0,0048

Size / LDA	1600		1500		1400		1300	
	Time	GFlops	Time	GFlops	Time	GFlops	Time	GFlops
20000	118,893	44,865	126,241	42,254	134,685	39,605	144,548	36,902
	118,812	44,896	126,133	42,290	134,585	39,634	144,442	36,929
	118,812	44,896	126,134	42,289	134,588	39,633	144,515	36,911
	118,807	44,898	126,144	42,286	134,578	39,636	144,560	36,899
Average	118,831	44,888	126,163	42,280	134,609	39,627	144,516	36,910
STD DEVI	0,0359	0,0136	0,0452	0,0152	0,044	0,013	0,0459	0,0118

Size / LDA	1200		1100		1000		900	
	Time	GFlops	Time	GFlops	Time	GFlops	Time	GFlops
20000	156,046	34,183	168,887	31,584	185,988	28,680	206,387	25,845
	155,916	34,212	168,838	31,593	186,119	28,660	206,247	25,863
	155,986	34,196	169,409	31,487	186,182	28,650	205,975	25,897
	156,089	34,174	169,576	31,456	186,188	28,649	206,347	25,850
Average	156,009	34,191	169,178	31,530	186,119	28,660	206,239	25,864
STD DEVI	0,0651	0,0143	0,321	0,0598	0,0805	0,0124	0,1607	0,0202

Size / LDA	800	
	Time	GFlops
20000	231,579	23,034
	231,236	23,068
	230,769	23,115
	231,491	23,043
Average	231,269	23,065
STD DEVI	0,3148	0,0314

Table B.1: Linpack Benchmark - Big Configuration Performance Results

Size / LDA	defaults		1800		no turbo		1700	
	Time	GFlops	Time	GFlops	Time	GFlops	Time	GFlops
1000	0,021	31,634	0,021	32,097	0,025	26,806	0,025	26,608
	0,019	34,969	0,019	34,726	0,024	28,434	0,024	28,366
	0,018	36,257	0,019	35,961	0,024	28,035	0,024	27,880
	0,019	35,624	0,019	35,310	0,024	27,964	0,024	27,931
AVG 1000	0,019	34,621	0,019	34,523	0,024	27,810	0,024	27,696
STD DEVI	0,00109	1,78379	0,00087	1,46766	0,00043	0,60642	0,00043	0,65596

Size / LDA	1600		1500		1400		1300	
	Time	GFlops	Time	GFlops	Time	GFlops	Time	GFlops
1000	0,026	25,359	0,028	24,172	0,029	22,970	0,030	22,081
	0,025	26,959	0,026	25,689	0,027	24,679	0,029	22,919
	0,025	26,980	0,026	25,552	0,027	24,727	0,029	23,112
	0,025	26,799	0,026	25,969	0,027	24,615	0,029	22,951
AVG 1000	0,025	26,524	0,026	25,346	0,028	24,248	0,029	22,766
STD DEVI	0,00043	0,6764	0,00087	0,6938	0,00087	0,73877	0,00043	0,40239

Size / LDA	1200		1100		1000		900	
	Time	GFlops	Time	GFlops	Time	GFlops	Time	GFlops
1000	0,034	19,902	0,035	19,162	0,037	18,000	0,042	16,045
	0,031	21,338	0,033	20,232	0,036	18,456	0,040	16,831
	0,031	21,452	0,033	20,301	0,036	18,617	0,040	16,837
	0,031	21,396	0,033	20,218	0,036	18,395	0,041	16,224
AVG 1000	0,032	21,022	0,034	19,978	0,036	18,367	0,041	16,484
STD DEVI	0,0013	0,64778	0,00087	0,4724	0,00043	0,22683	0,00083	0,35561

Size / LDA	800	
	Time	GFlops
1000	0,046	14,670
	0,044	15,227
	0,044	15,134
	0,044	15,198
AVG 1000	0,044	15,057
STD DEVI	0,00087	0,22582

Table B.2: Linpack Benchmark - Small Configuration Performance Results

Size / LDA	defaults		1800		no turbo		1700	
	Time(s)	GFlops	Time(s)	GFlops	Time(s)	GFlops	Time(s)	GFlops
1000	0,020	32,719	Discarded	Discarded	0,026	25,817	0,025	26,398
	0,018	36,613	0,018	36,534	0,024	27,801	0,023	28,528
	0,019	35,821	0,019	36,034	0,024	28,290	0,024	28,363
	0,019	35,036	0,020	34,216	0,024	28,211	0,024	28,192
2000	0,107	49,995	0,106	50,486	0,146	36,541	0,147	36,412
	0,106	50,486	0,105	50,856	0,145	36,794	0,147	36,249
	0,106	50,499	0,106	50,367	0,147	36,248	0,149	35,956
	0,105	50,834	0,105	51,094	0,145	36,864	0,148	36,003
5000	1,391	59,952	1,395	59,785	1,971	42,310	1,980	42,113
	1,385	60,220	1,388	60,094	1,971	42,301	1,983	42,050
	1,381	60,358	1,384	60,259	1,980	42,117	1,983	42,044
	1,379	60,461	1,382	60,330	1,979	42,138	1,983	42,052
10000	10,002	66,670	10,003	66,669	14,447	46,161	14,320	46,567
	12,686	52,568	11,931	55,893	14,331	46,532	14,316	46,582
	13,444	49,605	13,474	49,493	14,322	46,563	14,323	46,558
	13,472	49,501	13,535	49,271	14,331	46,534	14,301	46,631
15000	45,350	49,624	45,573	49,382	48,149	46,739	48,043	46,842
	45,378	49,593	45,560	49,395	48,147	46,742	48,042	46,843
	45,436	49,530	45,529	49,428	48,149	46,739	48,057	46,828
	45,326	49,651	45,641	49,307	48,157	46,732	48,050	46,835
18000	77,709	50,041	78,085	49,800	82,318	47,239	82,048	47,395
	77,749	50,015	77,902	49,917	82,313	47,242	81,382	47,783
	77,596	50,114	77,901	49,918	82,323	47,236	82,101	47,364
	77,522	50,162	77,981	49,866	82,310	47,244	82,156	47,332
20000	105,985	50,329	106,288	50,186	112,444	47,438	112,231	47,528
	105,911	50,364	106,549	50,063	112,441	47,440	112,250	47,520
	105,994	50,325	106,512	50,080	112,434	47,443	112,235	47,526
	106,014	50,316	106,257	50,200	112,428	47,445	112,108	47,581
AVG 1000	0,019	35,047	0,019	35,595	0,025	27,530	0,024	27,870
SD 1000	0,001	1,455	0,001	0,996	0,001	1,006	0,001	0,858
AVG 2000	0,106	50,453	0,105	50,701	0,146	36,612	0,148	36,155
SD 2000	0,001	0,299	0,001	0,290	0,001	0,242	0,001	0,185
AVG 5000	1,384	60,248	1,387	60,117	1,975	42,216	1,982	42,065
SD 5000	0,005	0,191	0,005	0,210	0,004	0,090	0,001	0,028
AVG 10000	12,401	54,586	12,236	55,331	14,358	46,448	14,315	46,585
SD 10000	1,421	7,085	1,440	7,065	0,052	0,166	0,008	0,028
AVG 15000	45,373	49,600	45,576	49,378	48,151	46,738	48,048	46,837
SD 15000	0,041	0,045	0,041	0,044	0,004	0,004	0,006	0,006
AVG 18000	77,644	50,083	77,967	49,875	82,316	47,240	81,922	47,469
SD 18000	0,090	0,058	0,075	0,048	0,005	0,003	0,314	0,183
AVG 20000	105,976	50,334	106,401	50,132	112,437	47,441	112,206	47,539
SD 20000	0,039	0,018	0,130	0,061	0,006	0,003	0,057	0,024

Table B.3: Linpack Benchmark - Mixed Configuration Performance Results (1/4)

Size / LDA	1600		1500		1400		1300	
	Time(s)	GFlops	Time(s)	GFlops	Time(s)	GFlops	Time(s)	GFlops
1000	0,026	26,007	0,028	24,205	0,028	23,685	0,030	22,000
	0,025	26,968	0,026	25,441	0,027	25,115	0,029	23,366
	0,025	26,989	0,027	24,643	0,027	25,107	0,029	23,096
2000	0,025	27,069	0,026	25,486	0,027	25,151	0,029	23,218
	0,152	35,141	0,161	33,210	0,172	31,117	0,180	29,618
	0,151	35,374	0,157	33,974	0,173	30,846	0,178	29,948
	0,151	35,437	0,157	34,038	0,174	30,774	0,179	29,824
	0,151	35,476	0,158	33,719	0,174	30,752	0,181	29,577
5000	2,066	40,366	2,172	38,394	2,309	36,116	2,482	33,591
	2,063	40,423	2,174	38,362	2,309	36,113	2,477	33,663
	2,066	40,366	2,174	38,363	2,305	36,168	2,483	33,585
	2,068	40,327	2,170	38,424	2,309	36,105	2,479	33,630
10000	15,146	44,029	16,004	41,669	17,059	39,092	18,403	36,237
	15,152	44,011	16,000	41,680	17,057	39,097	18,408	36,227
	15,150	44,017	16,001	41,676	17,059	39,092	18,396	36,251
	15,154	44,007	15,999	41,683	17,053	39,105	18,396	36,250
15000	50,876	44,234	53,664	41,936	57,434	39,183	61,749	36,445
	50,862	44,246	53,649	41,947	57,560	39,097	61,745	36,447
	50,867	44,242	53,641	41,954	57,540	39,111	61,736	36,453
	50,862	44,246	53,638	41,956	57,544	39,109	61,753	36,443
18000	86,994	44,700	91,843	42,340	98,525	39,468	105,798	36,755
	86,999	44,698	91,839	42,342	98,502	39,478	105,808	36,752
	86,454	44,979	91,841	42,341	98,509	39,475	105,789	36,758
	86,316	45,051	91,847	42,338	98,097	39,641	105,776	36,763
20000	117,920	45,235	125,505	42,501	133,905	39,835	143,860	37,079
	117,910	45,239	125,499	42,503	133,957	39,820	143,772	37,101
	117,918	45,236	125,502	42,502	134,598	39,630	143,776	37,100
	117,915	45,237	125,499	42,504	134,601	39,629	143,771	37,102
AVG 1000	0,025	26,758	0,027	24,944	0,027	24,765	0,029	22,920
SD 1000	0,000	0,435	0,001	0,543	0,000	0,623	0,000	0,539
AVG 2000	0,151	35,357	0,158	33,735	0,173	30,872	0,179	29,742
SD 2000	0,000	0,130	0,002	0,326	0,001	0,145	0,001	0,151
AVG 5000	2,066	40,370	2,172	38,386	2,308	36,125	2,480	33,617
SD 5000	0,002	0,034	0,002	0,025	0,002	0,025	0,002	0,032
AVG 10000	15,151	44,016	16,001	41,677	17,057	39,097	18,401	36,241
SD 10000	0,003	0,008	0,002	0,005	0,002	0,006	0,005	0,010
AVG 15000	50,867	44,242	53,648	41,948	57,520	39,125	61,746	36,447
SD 15000	0,006	0,005	0,010	0,008	0,050	0,034	0,006	0,004
AVG 18000	86,691	44,857	91,843	42,340	98,408	39,516	105,793	36,757
SD 18000	0,310	0,160	0,003	0,001	0,180	0,072	0,012	0,004
AVG 20000	117,916	45,237	125,501	42,503	134,265	39,729	143,795	37,096
SD 20000	0,004	0,001	0,002	0,001	0,335	0,099	0,038	0,010

Table B.4: Linpack Benchmark - Mixed Configuration Performance Results (2/4)

Size / LDA	1200		1100		1000		900	
	Time(s)	GFlops	Time(s)	GFlops	Time(s)	GFlops	Time(s)	GFlops
1000	0,031	21,278	0,034	19,526	0,037	17,959	0,042	16,110
	0,030	22,227	0,032	20,653	0,035	18,965	0,040	16,578
	0,030	22,217	0,032	20,625	0,035	18,953	0,040	16,573
	0,030	22,171	0,033	20,544	0,035	18,947	0,040	16,673
2000	0,195	27,419	0,206	25,893	0,227	23,565	0,250	21,339
	0,191	27,901	0,207	25,830	0,224	23,793	0,250	21,395
	0,191	27,913	0,206	25,988	0,225	23,730	0,249	21,426
	0,192	27,749	0,206	25,889	0,224	23,812	0,249	21,416
5000	2,659	31,356	2,885	28,902	3,154	26,436	3,496	23,849
	2,656	31,391	2,884	28,913	3,152	26,454	3,490	23,892
	2,659	31,355	2,882	28,930	3,148	26,491	3,501	23,816
	2,660	31,351	2,887	28,887	3,146	26,507	3,500	23,823
10000	19,773	33,727	21,501	31,015	23,544	28,324	26,177	25,475
	19,775	33,722	21,495	31,025	23,556	28,310	26,173	25,480
	19,769	33,733	21,561	30,929	23,543	28,326	26,185	25,468
	19,762	33,744	21,581	30,901	23,564	28,300	26,177	25,475
15000	66,311	33,938	72,452	31,061	79,097	28,452	87,936	25,592
	66,302	33,942	72,468	31,055	79,090	28,454	87,773	25,639
	66,301	33,943	72,458	31,059	79,093	28,453	87,753	25,645
	66,300	33,943	72,442	31,066	79,088	28,455	87,936	25,592
18000	113,696	34,202	124,209	31,307	135,712	28,654	150,896	25,770
	113,668	34,211	123,976	31,366	135,708	28,654	150,906	25,769
	113,688	34,205	123,676	31,442	135,717	28,653	150,897	25,770
	113,676	34,208	123,681	31,441	135,712	28,654	150,889	25,772
20000	155,367	34,333	169,072	31,550	185,558	28,747	206,366	25,848
	155,363	34,333	169,068	31,550	185,550	28,748	205,938	25,902
	155,373	34,331	169,065	31,551	185,888	28,695	206,317	25,854
	155,368	34,332	169,083	31,547	186,351	28,624	206,323	25,853
AVG 1000	0,030	21,973	0,033	20,337	0,036	18,706	0,041	16,483
SD 1000	0,000	0,402	0,001	0,470	0,001	0,431	0,001	0,219
AVG 2000	0,192	27,745	0,206	25,900	0,225	23,725	0,249	21,394
SD 2000	0,002	0,199	0,000	0,057	0,001	0,097	0,001	0,034
AVG 5000	2,659	31,363	2,885	28,908	3,150	26,472	3,497	23,845
SD 5000	0,001	0,016	0,002	0,016	0,003	0,028	0,004	0,030
AVG 10000	19,770	33,731	21,535	30,967	23,552	28,315	26,178	25,474
SD 10000	0,005	0,008	0,037	0,053	0,009	0,010	0,004	0,004
AVG 15000	66,303	33,942	72,455	31,060	79,092	28,453	87,850	25,617
SD 15000	0,004	0,002	0,009	0,004	0,003	0,001	0,087	0,025
AVG 18000	113,682	34,206	123,886	31,389	135,712	28,654	150,897	25,770
SD 18000	0,011	0,003	0,223	0,056	0,003	0,001	0,006	0,001
AVG 20000	155,368	34,332	169,072	31,550	185,837	28,703	206,236	25,864
SD 20000	0,004	0,001	0,007	0,001	0,327	0,050	0,173	0,022

Table B.5: Linpack Benchmark - Mixed Configuration Performance Results (3/4)

Size / LDA	800	
	Time(s)	GFlops
1000	0,046	14,546
	0,045	14,987
	0,045	14,912
	0,045	15,024
2000	0,282	18,935
	0,279	19,160
	0,281	18,990
	0,279	19,146
5000	3,922	21,262
	3,913	21,310
	3,917	21,289
	3,914	21,306
10000	29,332	22,735
	29,346	22,724
	29,337	22,732
	29,311	22,751
15000	98,296	22,895
	98,284	22,897
	98,269	22,901
	98,263	22,902
18000	169,185	22,985
	169,358	22,961
	169,346	22,963
	169,082	22,999
20000	231,407	23,051
	231,652	23,026
	231,667	23,025
	231,692	23,023
AVG 1000	0,045	14,867
SD 1000	0,000	0,190
AVG 2000	0,280	19,058
SD 2000	0,001	0,097
AVG 5000	3,917	21,292
SD 5000	0,004	0,019
AVG 10000	29,331	22,736
SD 10000	0,013	0,010
AVG 15000	98,278	22,899
SD 15000	0,013	0,003
AVG 18000	169,243	22,977
SD 18000	0,115	0,016
AVG 20000	231,605	23,031
SD 20000	0,115	0,011

Table B.6: Linpack Benchmark - Mixed Configuration Performance Results (4/4)

B.2 Energy - Linpack

Setting	IA Frequency MHz	Package Power Watt	IA Power Watt	GT Power Watt	Uncore Watt	Energy Joule
defaults	1928,205	15,309	11,712	0,000	3,943	1585,341
1800	1922,478	15,087	11,490	0,000	3,002	1567,702
no turbo	1698,641	12,937	9,368	0,000	3,139	1455,032
1700	1699,393	12,925	9,354	0,000	3,077	1451,327
1600	1599,315	12,126	8,578	0,000	2,959	1440,916
1500	1499,369	11,417	7,887	0,000	3,047	1440,380
1400	1399,283	10,710	7,195	0,000	2,883	1441,728
1300	1299,250	10,047	6,548	0,000	2,972	1451,927
1200	1199,251	9,443	5,946	0,000	2,938	1473,211
1100	1099,278	8,761	5,305	0,000	3,030	1482,244
1000	999,164	8,301	4,813	0,000	2,987	1544,979
900	899,210	7,781	4,291	0,000	3,366	1604,764
800	799,191	7,142	3,703	0,000	2,992	1651,765

Table B.7: Linpack Benchmark - Big Configuration Energy Results

Setting	IA Frequency MHz	Package Power Watt	IA Power Watt	GT Power Watt	Uncore Watt	Energy Joule
defaults	1390,500	5,612	2,353	0,000	3,259	0,108
1800	1800,000	7,908	4,322	0,000	3,586	0,154
no turbo	1440,500	5,163	1,689	0,000	3,474	0,125
1700	1700,000	5,223	1,730	0,000	3,493	0,127
1600	1600,000	5,035	1,559	0,000	3,476	0,127
1500	1500,000	4,846	1,436	0,000	3,410	0,128
1400	1400,000	4,714	1,319	0,000	3,395	0,130
1300	1300,000	4,603	1,232	0,000	3,370	0,135
1200	1199,500	4,528	1,154	0,000	3,374	0,144
1100	1100,000	4,568	1,046	0,000	3,522	0,153
1000	999,500	4,288	0,954	0,000	3,334	0,155
900	900,000	4,115	0,824	0,000	3,291	0,168
800	799,500	3,951	0,694	0,000	3,257	0,176

Table B.8: Linpack Benchmark - Small Configuration Energy Results

Setting	IA Frequency MHz	Package Power Watt	IA Power Watt	GT Power Watt	Uncore Watt	Energy Joule
defaults	1900,083	15,096	11,414	0,000	3,682	3666,785
1800	1894,865	15,111	11,424	0,000	3,687	3682,411
no turbo	1699,157	13,202	9,541	0,000	3,661	3424,650
1700	1699,317	13,155	9,502	0,000	3,653	3402,601
1600	1599,273	12,319	8,707	0,000	3,612	3361,360
1500	1499,192	11,559	7,977	0,000	3,582	3344,537
1400	1399,264	10,973	7,375	0,000	3,599	3399,033
1300	1299,197	10,307	6,724	0,000	3,582	3426,136
1200	1199,216	9,587	6,045	0,000	3,541	3432,129
1100	1099,242	9,013	5,467	0,000	3,546	3515,516
1000	999,187	8,452	4,908	0,000	3,543	3614,004
900	899,155	8,019	4,428	0,000	3,591	3808,635
800	799,164	7,452	3,876	0,000	3,575	3969,626

Table B.9: Linpack Benchmark - Mixed Configuration Energy Results

Appendix C

SimGrid modifications

C.1 Runtime host speed change

Due to the heavy layering of the SimGrid source code, some changes needed to be made in order to change the host speed value. The changes made to SimGrid version 3.13 are provided in the form of a patch in the listing C.1.

```
Description: Add sg_host_set_speed method
  Changes made to almost all simgrid layers so that the host speed can
  be changed in runtime.

Author: Eduardo Fernandes <ei12130@fe.up.pt>

--- simgrid-3.13.orig/include/simgrid/host.h
+++ simgrid-3.13/include/simgrid/host.h
@@ -51,6 +51,7 @@ XBT_PUBLIC(xbt_dict_t) sg_host_get_mount
 XBT_PUBLIC(xbt_dynar_t) sg_host_get_attached_storage_list(sg_host_t host);
 // ========== user-level functions ===============
 XBT_PUBLIC(double) sg_host_speed(sg_host_t host);
+XBT_PUBLIC(void) sg_host_set_speed(sg_host_t host, double newSpeed);
 XBT_PUBLIC(double) sg_host_get_available_speed(sg_host_t host);

 XBT_PUBLIC(int) sg_host_get_nb_pstates(sg_host_t host);
--- simgrid-3.13.orig/include/simgrid/msg.h
+++ simgrid-3.13/include/simgrid/msg.h
@@ -265,6 +265,7 @@ XBT_PUBLIC(void) MSG_host_on(msg_host_t
 XBT_PUBLIC(void) MSG_host_off(msg_host_t host);
 XBT_PUBLIC(msg_host_t) MSG_host_self(void);
 XBT_PUBLIC(double) MSG_host_get_speed(msg_host_t h);
+XBT_PUBLIC(void) MSG_host_set_speed(msg_host_t host, double newSpeed);
 XBT_PUBLIC(int) MSG_host_get_core_number(msg_host_t h);
 XBT_PUBLIC(xbt_swag_t) MSG_host_get_process_list(msg_host_t h);
 XBT_PUBLIC(int) MSG_host_is_on(msg_host_t h);
--- simgrid-3.13.orig/include/simgrid/s4u/host.hpp
```

```
+++ simgrid-3.13/include/simgrid/s4u/host.hpp
@@ -68,6 +68,7 @@ public:
   bool isOff() { return !isOn(); }

   double speed();
+  void setSpeed(double);
   int core_count();
   xbt_dict_t properties();
   const char*property(const char*key);
--- simgrid-3.13.orig/src/msg/msg_host.cpp
+++ simgrid-3.13/src/msg/msg_host.cpp
@@ -150,6 +150,10 @@ double MSG_host_get_speed(msg_host_t hos
   return host->speed();
 }

+void MSG_host_set_speed(msg_host_t host, double newSpeed) {
+  host->setSpeed(newSpeed);
+}
+
 /** \ingroup m_host_management
   * \brief Return the speed of the processor (in flop/s), regardless of the current
       load on the machine.
   * Deprecated: use MSG_host_get_speed
--- simgrid-3.13.orig/src/s4u/s4u_host.cpp
+++ simgrid-3.13/src/s4u/s4u_host.cpp
@@ -157,6 +157,10 @@ double Host::powerPeakAt(int pstate_inde
 double Host::speed() {
   return pimpl_cpu->getSpeed(1.0);
 }
+/** @brief Set the speed of the cpu associated to a host */
+void Host::setSpeed(double sp) {
+  pimpl_cpu->setSpeed(sp);
+}
 /** @brief Returns the number of core of the processor. */
 int Host::core_count() {
   return pimpl_cpu->getCore();
--- simgrid-3.13.orig/src/simgrid/host.cpp
+++ simgrid-3.13/src/simgrid/host.cpp
@@ -126,6 +126,14 @@ double sg_host_speed(sg_host_t host)
   return host->speed();
 }

+// ============ user-level functions ===============
+// =================================================
+/** @brief Changes total speed of a host */
+void sg_host_set_speed(sg_host_t host, double newSpeed)
+{
+  host->setSpeed(newSpeed);
+}
```

```
76 +
77  double sg_host_get_available_speed(sg_host_t host)
78  {
79    return host->pimpl_cpu->getAvailableSpeed();
80 --- simgrid-3.13.orig/src/surf/cpu_interface.cpp
81 +++ simgrid-3.13/src/surf/cpu_interface.cpp
82 @@ -210,6 +210,12 @@ double Cpu::getSpeed(double load)
83    return load * speed_.peak;
84  }
85
86 +void Cpu::setSpeed(double sp){
87 +  speed_.peak = sp;
88 +  lmm_update_constraint_bound(getModel()->getMaxminSystem(), getConstraint(),
       speed_.peak * speed_.scale);
89 +  onSpeedChange();
90 +}
91 +
92  double Cpu::getAvailableSpeed()
93  {
94  /* number between 0 and 1 */
95 --- simgrid-3.13.orig/src/surf/cpu_interface.hpp
96 +++ simgrid-3.13/src/surf/cpu_interface.hpp
97 @@ -109,6 +109,8 @@ public:
98
99    /** @brief Get the speed, accounting for the trace load and provided process
          load instead of the real current one */
100   virtual double getSpeed(double load);
101 +  /** @brief Set the speed */
102 +  virtual void setSpeed(double);
103
104  protected:
105   /** @brief Take speed changes (either load or max) into account */
```

Listing C.1: SimGrid runtime host speed change patch