CS5099 DISSERTATION OF MSc IN COMPUTER SCIENCE

CPU Energy Consumption Analysis of Parallel Haskell Applications

Student Name:
Theodoros Dimopoulos

Student Matriculation Number:
160022265

Supervisor:
Professor Kevin Hammond

University of
St Andrews

School of Computer Science
University of St. Andrews
St. Andrews, Fife, Scotland

August 2017

Declaration

"I hereby certify that this dissertation, which is approximately 11900 words in length, has been composed by me, that it is the record of work carried out by me and that it has not been submitted in any previous application for a higher degree. This project was conducted by me at The University of St Andrews from May 2017 to August 2017 towards fulfillment of the requirements of the University of St Andrews for the degree of MSc under the supervision of professor Kevin Hammond."

"In submitting this project report to the University of St Andrews, I give permission for it to be made available for use in accordance with the regulations of the University Library. I also give permission for the title and abstract to be published and for copies of the report to be made and supplied at cost to any bona fide library or research worker, and to be made available on the World Wide Web. I retain the copyright in this work."

Date	**Signature**
14 August 2017	Dimopoulos Theodoros

Contents

Abstract

Energy efficiency, functional programming languages and multi-core architectures constitute popular research fields with already numerous applications in the computer science field. In this dissertation, we investigate the behaviour of scalable parallel applications from the energy efficiency point of view. In addition, we analyse the relation of the time and energy variables, whilst in the last chapter, we created energy model for the benchmarks. The applications are written in Haskell a purely functional lazy programming language which lends itself well to concurrent and parallel programming due to its explicit handling of effects[1].

This project supports the opinion that energy the consumed by the CPU is tightly coupled with the application's execution time that simplifies to, "more speedup-less energy usage", due to increased CPU utilization. This result forces us to focus on application's performance by not treating energy as a different factor. This is achieved by the identification of these distinct variables. In the last part of the thesis, we make a suggestion on how this research could be adjusted in order to work on the memory level of the computer.

[1] For more information about Haskell programming language visit https://www.haskell.org/

Chapter 1

Introduction

In the past decade, multicore architectures made the software engineering society to focus on increasing the application's performance when it is assigned to more cores. Subsequently, the emerge of mobile devices turned peoples' attention on the energy efficiency of the running software [19] [8]. As a result, a lot of research has been done to investigate the application impact on power usage[17][15].

This project focuses on the impact of parallel applications speedup on energy. The experiment is performed on Intel's Xeon architecture with parallel applications written in a lazy, purely functional language.

1.1 Performance And Energy Usage Relationship

The project aims to investigate the relationship of the terms, performance and energy consumption. The question that we are focusing is, to what extent we can save energy by increasing the application performance. The main hypothesis is that the less execution time leads to higher energy efficiency. In order to reject or accept that hypothesis, we conducted a series of experiments and extensive data analysis. The experiments are executions of a collection of scalable Haskell benchmark's written by Glasgow Haskell Compiler development group. In addition, we tried to create optimal execution conditions by running the application's on a server machine with the minimum possible, work load. Later in chapter 3, all the critical information about the experimental design are provided to the reader.

The speedup impact on performance has been analysed in depth and it is easy to comprehend. Almost every, person uses multicore devices in order to benefit from multitasking which increases the performance of the overall system. However, it is not always known the energy trade-off of this improvement. Contrary, nowadays the most important property of the mobile systems is not the performance but the battery autonomy. Hence in this dissertation, we wanted to investigate these two top concerns and measure how they co-exist. As a result, another main concern in this part is *how strong is the correlation between performance and energy consumption and how consistent is this correlation in different applications.*

1.2 Energy Consumption Prediction

Currently, application performance and energy efficiency are considered distinct, forcing separate research effort. Our goal is to bring closer these two terms in order to eradicate this necessity and create a unified research model that can estimate parallel application energy requirements. In the last chapter of this thesis, we put our effort to generate energy models, for the specific utilized hardware of the parallel benchmarks and in the same time, we try to answer the question, in what extend the energy consumption can be predicted from the application performance. To achieve this result, we performed extensive statistical analyses in order to define an energy model for all the applications that we used as benchmarks. This model will provide estimations of the application energy consumption given as input variables the problem size and the number of available cores.

The selected dissertation topic is considered as important for a variety of reasons. Firstly, the multicore architectures will continue to be developed and evolved making the performance major concern in computer systems. Secondly, energy efficiency will continue to be highly important due to environmental reasons and mobile devices which tend to become more and more powerful. Finally, the functional style programming has increasing popularity with its properties being adopted by all the heavily used programming languages. Thus, it is important to investigate the aspects of Haskell parallel applications' energy efficiency.

Chapter 2

Context Survey on Energy Consumption

2.1 Literature Review On Energy Consumption

Measuring the energy consumption of an application and understanding its behaviour provides new opportunities for energy savings. In addition, energy-efficiency became more popular topic the recent years due to mobile computing particularly because of the smartphones and tablet devices [20] [5].

Gustavo Pinto [18] evaluates the performance and energy consumption of four Java applications that use three different programming concurrent constructs, plus a sequential implementation that run on an Intel(R) Xeon(R), 2.13 GHz, 4 cores/8 threads and with 16GB of memory, running Linux 64-bit, kernel 3.0.0.-31-server. The goal of this study is to demonstrate the difficulty to generate trade-offs estimations between energy-efficiency and performance. Luís Gabriel Lima et al [1] investigated the energy behaviour of programs written in Haskell. They have conducted two empirical studies to analyse the energy efficiency of Haskell programs from two different perspectives: strictness and concurrency. [12] presents a general methodology for investigating the impact of software change on power consumption. First, it relates the power consumption with software changes and then, it investigates the impact of static Object Oriented software metrics on power consumption. All of the cited work demonstrates that software structure can affect power consumption.

2.2 Profiling Tools

Regarding the resource monitoring, we use in our research Intel's"RAPL" interface which stands for Running Average Power Limit[1]. RAPL interface provides a set of counters providing energy and power consumption information. RAPL is not an analog power meter, but rather a software power model. This software power model estimates energy usage by accessing data from hardware performance counters and I/O models[9].

2.2.1 Energy Measurement

Measuring Energy in different computer components is something more or less ordinary nowadays because of mobile devices such as laptops and smartphones. This thesis will measure the energy consumption of the Intel Xeon E5-2690 v4 processor. Being released in January 2016, this relatively new CPU has a lot of new features compared to its predecessors making him an ideal execution environment. More important, this fourth generation of Intel Xeon processor provides accurate measurements of energy consumption [7] using the RAPL interface. The Intel's RAPL interface enables a program to gain energy information directly from the CPU registers. Few application examples are shown below.

Turbostat
Turbostat is provided by the kernel-tools package. It reports on processor topology, frequency, idle power-state statistics, temperature, and power usage on Intel® 64 processors. Turbostat is useful for identifying servers that are inefficient in terms of power usage or idle time. It also helps to identify the rate of system management interrupts (SMIs) occurring on the system. It can also be used to verify the effects of power management tuning [2].

PowerTOP
Intel's open source application PowerTOP is a software utility designed to measure, explain and minimize a computer's electrical power consumption. It was released by Intel in 2007 under the GPLv2 license. It works for Intel, AMD, ARM and UltraSPARC processors. Moreover, PowerTOP is a Linux tool to diagnose issues with power consumption and power management and also has an interactive mode where the user can experiment various power management settings for cases where the Linux distribution has not enabled these settings [10].

For our experiment purposes it was decided not to use the above software, for the reasons that PowerTOP is designed for mobile devices [2]. Although it was possible to use turbostat, the process of developing a software that integrates with turbostat was considered unnecessary overhead because the goal was to obtain energy data and not the process of obtaining them. As a result, two applications that take advantage of RAPL interface were used to measure accurately as possible the energy consumption

[1] Information about the RAPL interface are available on https://01.org/blogs/2014/running-average-power-limit-%E2%80%93-rapl

[2] Even though PowerTOP can be used for desktop and server computers it provides the majority of its functionality to devices that operate with batteries

on CPU and DRAM package level. The first one[3] was used to measure the power that is consumed by the application on CPU package level. Th *AppPowerMeter* application was simple and did exactly what was necessary to perform our experiment. To collect the information we had just to executed and pass the benchmark as a parameter while the execution results were stored into text files for later analysis. Unexpectedly, in order to be able to run it on the *Corryvreckan* server it was necessary to amend it because the Intel Xeon E5-2690 v4 is relatively new CPU and was not supported by the application. This action performed with the supervisor. On the other hand, the *rapl-plot* application[4] was utilized to measure energy on the machine's memory package.

In conclusion, to support further our research we gathered more information from the benchmark execution by enabling the GHC profiling tool with "+RTS -s" flags which provide details of memory usage, garbage collection, execution time and many others that were used for the benchmarks run time analysis.

[3] The application is available on https://github.com/kentcz/rapl-tools

[4] The source code is available on https://github.com/deater/uarch-configure

Chapter 3

Experimental Design

3.1 Overview

The experiment consists of running parallel Haskell applications as benchmarks and generating data from different profiling tools. The next step is validation and finally to data analysis and information extraction.

3.2 Objectives

The objectives of the data analysis are as follows:

- Performance of the application in terms of speedup the sequential execution.
- Energy Consumption of the application in terms of Joules and Watts.
- Core affinity effect on energy
- Correlation of performance and energy consumption.
- Examination of the CPU Hyper Threading and its impact on Energy Consumption
- Create an Energy Consumption Model.

3.3 Experiment Setup

3.3.1 Resources

- **Hardware.** The benchmarks were executed on the "Corryvreckan" University of St. Andrews machine, a 2.6GHz Intel Xeon E5-2690 v4 machine with 28 physical cores and 256GB of RAM running scientific Linux 3.10.0. This machine allows TurboBoost up to 3.6GHz, and supports automatic dynamic frequency scaling and hyper-threading up to 56 virtual cores. It was released in the first quarter of 2016 and has 135W Thermal Design Power[1], which was confirmed during our experiments.

- **Haskell and GHC.** Glasgow Haskell Compiler version 7.6.3, stage 2 is used to compile the benchmarks[2]. Regarding the garbage collection mechanism, we benefit from a parallel generational garbage collector with overall 2 generations. In every benchmark, we use the default flags for the Garbage Collector which means that the Garbage Collector will allocate 512 Kilobytes of memory. The allocation area of the garbage collector kept small because it is not guaranteed that a larger area will perform better. A large allocation area means worse cache behaviour but fewer garbage collections and less promotion to the highest generation of referenced objects.

  After few tests with a simple parallel Fibonacci application, it was decided not to change the Garbage Collector allocation area because it would have overall worse performance, compared to the default size something that would bias the overall experiment.

- **Benchmarks.** Parallel Haskell applications were selected to act as benchmarks for our research project [3]. These applications have been configured and compiled in ways to provide the greatest speedups. Moreover, some of them have been amended to exclude functions of writing to standard output. In the beginning of the project, five parallel applications were used in the experiments and selected due to their scalability. These were *NBoby*, *Sum Euler*, *Matrix Multiplication*, *Queens* and *Sudoku*. These applications were investigated also in [4] which acts as guide for this thesis. In the second part of the experiment and after initial data analysis, we decided to extend the number of benchmarks by adding another four parallel applications from the same benchmark suite. This decision improved the quality of our research by adding more useful data to our analysis. These new four benchmarks were *Coins*,*Mandelbrot*, *parallel RSA* and *Ray*.

[1] Thermal Design Power (TDP) represents the average power, in watts, the processor dissipates when operating at a base frequency with all cores active under an Intel-defined, high-complexity workload.

[2] More information about Glasgow Haskell Compiler on https://www.haskell.org/.

[3] The benchmarks are available on https://github.com/ghc/nofib.

3.3.2 Experiment Process

The experiment consists of running the configured benchmarks on the "Corryvreckan" machine. This server is dedicated to research and does not run any web services or other processes, except for the ones necessary for operating. The benchmarks were executed through a *bash* script. I every application was executed with a range of input from **relatively small** to **relatively large**, which is application specific, exactly ten times on the following set of cores : [1, 4, 8, 12, 16, 20, 24, 27, 32, 36, 40, 44, 48, 52, 55, 56]. The executions results were stored in text files for later analysis. The next table summarizes the benchmarks with their input which specifies the workload[4].

Benchmark	Min Value	Mid Value	Max Value
NBody	10000	75000	250000
Euler Summation	1000	25000	50000
Matrix Mult.	1500	3000	5000
Queens	11	14	16
Sudoku	16000	320000	800000
Coins	1400	1800	2200
Mandelbrot	2000	2750	3500
Parallel RSA	1000000	5000000	10000000
Ray	1600	2800	4000

Table 3.1: Information about benchmarks input sizes.

The total execution time for all the benchmarks was approximately six full days.

[4] The table illustrates the edge input values and the middle one. However, the benchmarks were tested in many different input problem sizes with size inside the min and the max value range.

Chapter 4

Data Analysis. Core Affinity

4.1 Overview

Data analysis constitutes the main part of the thesis. The data themselves were able to guide the direction of the thesis. During the analysis, we were able to gain significant amount of information that we wanted to analyse even more. However, we had to put a threshold on the information extraction and provide sustainable and reasonable explanations. The main analysis is split into chunks as follows:

- Investigate performance of parallel applications in conjunction with core affinity.
- Identify the relationship of performance, CPU energy consumption and power.
- Explain the CPU's hyper- threading behaviour.
- Extend of the research to memory level.

In order not to overwhelm the reader with tables and graphs of results, we illustrate just a subset of the graphs and results. However, our explanations are a result of a careful examination of all nine benchmarks' data.

4.2 Performance Core Affinity and Task Migration

In the beginning of the research, it was decided to proceed with five scalable benchmarks and perform an experiment to test whether core affinity improves the performance of the application.

Core Affinity With Glasgow Haskell Compiler
Core Affinity is the action of locking a process on a specific processor core which prevents its migration to others. This can be effective because there will be less IO operations due to less state migration. To elaborate more on this action, when a process is running on a specific core, it keeps its state in the L1 cache memory. In Xeon E2690 v4 processor architecture, every core has its own L1 and L2 thus the process context has to be copied to another core cache which consists a significant overhead. However, the core pinning can have a negative effect on power usage and

workload balancing of the system. In addition, having all CPU cores occupied on a similar workload minimizes the power consumption.

GHC provides the following options to control how tasks are mapped to available set of cores.

-qa	Set core affinity by using kernel routines.
-qm	Prevent migration of worker tasks between cores
-qw	Migrate worker task to the current core when it is woken up from sleep

Table 4.1: GHC Core Affinity options.

The **-qa** option assigns the worker task to the same core using the system scheduler. This is expected to have better performance results because the context of a task that lies in the L1 cache (at least level 1) won't be copied to another level 1 and 2 CPU cache memory. The **-qm** and **-qm** have similar results according to the GHC documentation. In our experiments. we use the flag **-qa**.

4.2.1 Core Affinity Assessment

The core affinity impact on performance was investigated by an experiment. Five benchmark applications selected, NBody, Euler Summation, Matrix Multiplication, Queens and Sudoku with different sets of problem input sizes. These sizes begin from relatively small values and they increase up to a certain point, in order to provide enough amount of work for the CPU. Typically the application on maximum input sizes and core number need to need than a minute in their sequential execution [1]. Each benchmark is executed on a different number of CPU cores. In the experiment overall, each unique combination of *benchmark*, *input size* and *number of cores* is executed ten times with the core affinity enabled and ten times with core affinity disabled. Tables 4.2, 4.3, 4.4, 4.5 and 4.6 display the benchmarks' result.

The two hypothesis for each unique execution combination are:

> **H0:** There is no significant difference between the execution times of the application's unique combination with and without core affinity
> **H1:** There is a significant difference between the execution times of the application's unique combination with and without core affinity

The execution times mean were tested by T-test using R-Studio[2] which showed if there is a statistically significant. The t-tests will compare the mean of the execution time of the overall 20 executions (ten with CPU pinning and ten without).

[1] The maximum input sizes need around 100 seconds on average to execute.

[2] R-Studio is available at https://www.rstudio.com/

Threads Input	1	8	16	24	32	40	48	56
10000	⬛	🟩	⬛	⬛	🟥	⬛	🟩	⬛
25000	⬛	🟩	⬛	🟥	⬛	⬛	🟩	⬛
50000	⬛	⬛	🟩	⬛	🟥	⬛	🟩	⬛
75000	⬛	⬛	🟩	⬛	🟥	🟩	🟩	🟩
100000	⬛	🟩	🟩	⬛	🟥	🟩	🟩	⬛
250000	⬛	⬛	⬛	⬛	⬛	🟩	🟩	⬛

Table 4.2: Nbody application Results on Core Affinity Experiment.

Threads Input	1	8	16	24	32	40	48	56
1500	⬛	🟥	🟥	🟥	⬛	⬛	⬛	⬛
2000	⬛	🟥	🟥	⬛	🟥	🟥	⬛	⬛
2500	⬛	🟥	🟥	🟥	⬛	🟥	⬛	🟥
3000	⬛	🟥	🟥	🟥	⬛	⬛	⬛	⬛
3500	⬛	🟥	🟥	⬛	⬛	🟥	⬛	⬛
4000	⬛	🟥	🟥	⬛	⬛	🟥	⬛	⬛
4500	⬛	🟥	🟥	⬛	⬛	🟥	🟥	⬛
5000	⬛	🟥	⬛	⬛	⬛	🟥	🟥	⬛

Table 4.3: Matrix Multiplication application results on Core Affinity experiment.

Threads Input	1	8	16	24	32	40	48	56
11	⬛	⬛	⬛	⬛	⬛	⬛	🟩	🟩
12	⬛	⬛	⬛	⬛	⬛	⬛	🟩	🟩
13	⬛	⬛	⬛	⬛	⬛	🟩	🟩	🟩
14	⬛	⬛	⬛	⬛	⬛	⬛	⬛	🟩
15	⬛	⬛	⬛	⬛	⬛	🟩	⬛	⬛
16	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛

Table 4.4: Queens application results on Core Affinity experiment.

Threads Input	1	8	16	24	32	40	48	56
16000	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛
160000	⬛	⬛	🟥	⬛	⬛	⬛	⬛	⬛
320000	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛
480000	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛
640000	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛
800000	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛

Table 4.5: Sudoku application results on Core Affinity experiment.

Threads Input	1	8	16	24	32	40	48	56
1000	⬛	⬛	⬛	⬛	⬛	⬛	🟩	🟩
5000	⬛	🟥	🟥	🟥	⬛	⬛	🟩	🟩
10000	⬛	🟥	🟥	🟥	🟥	🟩	⬛	🟩
15000	⬛	🟥	🟥	🟥	🟥	🟥	⬛	🟩
20000	⬛	🟥	🟥	🟥	🟥	🟥	⬛	🟩
25000	⬛	🟥	🟥	🟥	🟥	🟥	⬛	🟩
30000	⬛	🟥	🟥	🟥	⬛	🟥	⬛	🟩
35000	⬛	🟥	🟥	🟥	🟩	🟥	⬛	🟩
40000	⬛	🟥	🟥	🟥	⬛	🟥	⬛	🟩
45000	⬛	🟥	🟥	🟥	🟩	🟥	⬛	🟩
50000	⬛	🟥	🟥	🟥	🟩	🟥	⬛	🟩

Table 4.6: Euler Summation application Results on Core Affinity Experiment.

Tables 4.2, 4.6, 4.3 4.4 and 4.5 illustrate the results of the T-tests with significance level of $a = 0.05$. The T-tests are able to highlight a statistically significant difference in the means of execution time values (coming from a normally distributed population) which is exactly what we are investigating. Every coloured cell in the tables is a unique execution combination that has been applied to a T-test. In the cases that the null hypothesis has been rejected (the execution time means are not equal) we coloured the cell green or red according to core affinity better or worse performance. As a result, in the green cells the process executes faster with pinning enabled while in the red cells the application runs slower.

4.2.2 Core Affinity Experiment Results

Based on theory, core affinity is expected to give better performance because there is no need to move the threads along the CPU which includes at least cache level 1 and 2 memory cost due to state transfer. Furthermore, if the thread is constantly rescheduled to execute on the same core, it can benefit even more from the accumulated state which will cause less cache misses[3]. On the other hand, if a processor is constantly working while others are idle the overall work load is not balanced with impact on energy consumption and performance.

Even though the experiment did not provide consistent results for every benchmark, we definitely can view some patterns in them. In the majority of the cases, there is not any significant difference between the regular scheduling and the process pinning (the gray table cells). However, it seems that there is a correlation between the size of the available cores and the effectiveness of the core affinity. In almost every benchmark, the larger core values achieve better or equal performance when the application is pinned to a core. In contrast, all the applications except for NBody, execute slower in the low number of cores. Finally, we see that the sequential execution is not affected by the core affinity setting.

Based on our results, it seems that process migration is more expensive when most of the CPU Cores are active. In contrast, it is better to have threads “jumping” to idle cores than being locked when only a small subset of cores is available.

Despite our expectations that core affinity would perform better in the majority of the cases, the statistical analysis did not confirmed us. Even for the case of sequential execution that is easier to conceive the thread execution, the process migration did not affect the performance and proved us wrong. Because of the lack of evidence we focused on how the applications are processed by the CPU in order to explain this phenomenon.

In our tables, it is obvious that when most of the cores are enabled, having threads migrating from one core to another, delays the execution of the process because the threads are moving to overloaded cores and they suffer from extensive sleep and memory IO. Moreover, it is possible that threads are locked on the same core so the scalability is not the optimal. Finally, another reasonable consideration is that high

[3] A cache miss happens when the CPU request data that are not in any cache level and they have to be obtain from the main memory

core temperatures cause worse performance and that also thread migrations have the same effect. This behaviour can explain why the locking has worse results on small sets of available cores. The enabled cores have large workloads and increase their temperature thus, their performance drops. This observation, boost our research on the CPU performance and temperature which was confirmed by the constructor of our testing machine CPU[11]. Older CPUs would simply fail if they started to overheat, but modern CPUs adjust their frequency based on temperature to prevent a dramatic failure. This explains all of our findings, except for why there is no difference in the sequential execution. The answer is that both cases are delayed from high core temperatures, in core affinity case and from state migration overhead, in regular scheduling case. Moreover, we claim that in the small input sizes, the temperature and core affinity do not impact the performance because the short execution time cannot increase the core temperature and the thread migration cannot happen many times.

This dissertation topic was not the impact of pinning. This experiment took place because we wanted to achieve the highest possible performance on the benchmarks, thus we had to take the core affinity under consideration in this project. We did not extend the investigation on core affinity, however, if this was the case, we would research how GHC achieves core affinity. Based on the investigation results, it was decided that the benchmarks would be executed without any GHC core affinity flag enabled.

Chapter 5

Data Analysis. Energy Consumption

5.1 Introduction in Energy Consumption Research

The primary objective of the dissertation is to investigate the energy consumption of the application. The critical decisions on how to achieve this were:

1. On which computer device we will capture the energy consumption.
2. Which benchmarks we will run on the computer while we monitor the process.
3. How the energy is going to be measured.

The obtained data have been analysed using R-Studio package. Along with statistic results we will provide useful graphs created using the open source *ggplot2*[1] library. The statistical analysis made in R-Studio with the data coming from text files. These files were parsed and the useful information accumulated into Comma Separated Value files that R-Studio makes easy to read.

This chapter section will be filled by picking four benchmarks and illustrate basic graphs that will demonstrate the performance and the energy consumption of the applications.

[1] The ggplot2 library is available on https://cran.r-project.org/web/packages/ggplot2/index.html

5.1.1 CPU Energy Consumption

Firstly, simple physics definitions are provided to help the reader stay on track.

(Energy) E = The capacity of doing work [2] *which is measured in Joules (J)*
(Power) P = The Energy that is consumed per second from the machine, which is measured in Watts(W).

In the rest of this chapter and in the next ones, the terms energy and power consumption will be used interchangeably to describe how many Joules the machine has consumed during the execution.

Regarding this experimental study, we decided to measure the energy of the CPU package and to capture the energy consumption of the CPU using energy software models that utilize the RAPL interface.

5.1.2 Haskell Benchmarks For Energy Consumption

The dissertation is focusing on parallel application's behaviour which was inspired from the module CS4204-Concurrency and Multi-Core Architectures of the University of St Andrews. The fastest solution, in order to perform the experiment swiftly and save time for the analysis, was to obtain quality parallel applications from an official GHC Github repository[3]. This repository contains plenty of different Haskell benchmarks developed by people involved in the GHC development. It was considered the best source for benchmarks for our research project. The parallel benchmark suite repository contains 21 applications. Overall, we selected the NBody, Euler Summation, Matrix Multiplication, Queens, Sudoku, Coins, Mandelbrot, Ray and Parallel RSA as experiment participants.

5.2 Benchmarks Performance Analysis

For this study, we selected to illustrate results of four out of nine applications that we tested, namely Nbody, Euler Summation, Sudoku and Ray. The execution times of the applications are illustrated in figure 5.1. The graphs display the total execution time of each application across the number of cores and problem sizes. On the graph's $x - axis$ there is the number of cores that were available for the application and on the $y - axis$, the total execution time in seconds which comprises the mutation and the garbage collection time, also called, total user time. On the right side of the graph is the group of input sizes of each benchmark. The input size is relatively arbitrary from small to large values, large enough to keep the maximum number of available cores working for more than 10 seconds in 56 cores.

The vertical dashed line at 28 cores highlights the number of physical cores of the Intel Xeon E5-2690 v4 CPU. This CPU architecture achieves fifty-six virtual cores

[2] Definition is available on https://www.britannica.com/science/energy
[3] The GHC repository is available on https://github.com/ghc/nofib

due to Hyper-Threading mechanism. We place the dashed line in the graphs because that point is critical for the execution of the parallel applications.

5.2.1 Graphs' Interpretation

On the left side of the figure 5.1, there is the NBody and Euler Summation applications. These applications have similar line charts, displaying the reduction of the execution time of the various problem sizes. In contrast, on the right side, the Sudoku and Ray application improve their performance up to the 20 and 24 number of cores respectively. Beyond this point, the programs total execution time is increasing.

Even though the graphs give a taste of parallelization and speedup, there is a lot of information that they cannot display. The largest problem sizes, especially on the left side are concealing the information of the smaller ones. On a higher abstraction, the benchmarks unveil two patterns. Benchmarks on the left are similar to each other and the same is happening with the right side. However, if we exclude the core values which are greater than 12, we can see that all the graphs look pretty similar more or less. Unfortunately, to gain more detail about execution times it is necessary to transform these graphs.

5.2.2 Benchmarks' Speedup

In parallel processing, it is expected that parallel applications run faster when they assigned in more cores. However, that usually is not the case which highlights why the parallel processing is hard to work on. At this point, we will elaborate on the application performance and we will try to explain the results.

In order to attain better visualization results we transformed the execution time variable and we generated the "speedup". The term speedup value defines how many times faster the application was executed across the different sets of available cores. As a result, the figure 5.2 is able to illustrate more accurate information of our data. On this graph's $x - axis$ again there is the number of cores whilst on y it is the speedup value. The speedup values in the chart are generated by dividing the sequential execution time with the current case, as it is defined in equation 5.1.

$$Speedup_i = Time_1/Time_i, i \in [1, 4, 8, 16, 24, 27, 32, 36, 40, 48, 52, 55, 56] \qquad (5.1)$$

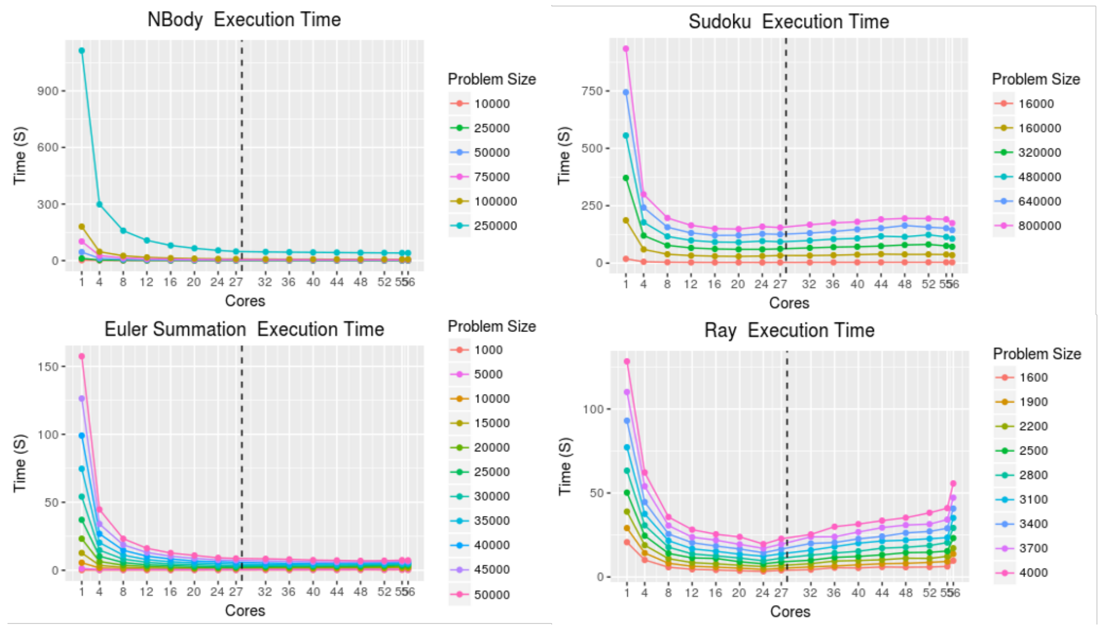


Figure 5.1: Execution time of the NBody, Euler Summation, Sudoku and Ray applications on multiple problem sizes.

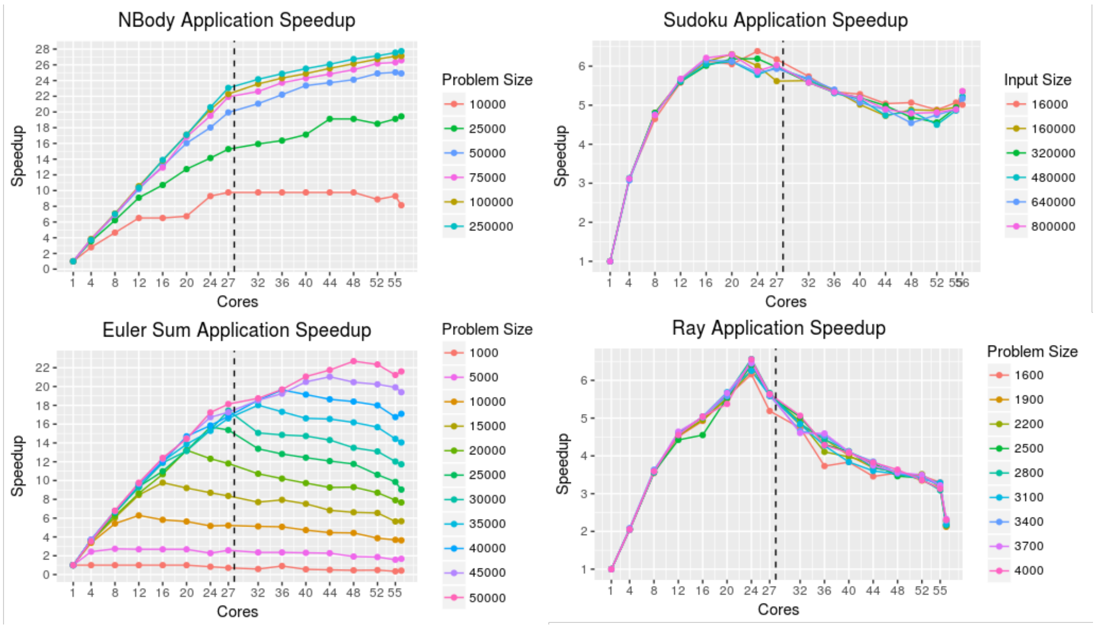


Figure 5.2: Speedup in the NBody, Euler Summation, Sudoku and Ray applications on multiple problem sizes.

Figure 5.2 displays the benchmarks' speedups. We benefit from this transformation by getting information for all the problem input sizes. Similarly to execution time figure, on the left side there are the NBody and Euler Summation applications and on the right the Sudoku and Ray. The left side benchmarks achieve gradually higher speedup values while moving towards to the highest inputs. This is happening because the lower input sizes do not provide enough workload to the available processors thus the overhead from the multi-threading (cost to instantiate the new threads) is relatively big compared to the useful work eradicating the possibility of further speedup. The left side applications' speedup graphs, look so similar because both of them have low memory requirements which minimize the overhead from the garbage collection mechanism. As a consequence, they scale nicely and close to optimal values as they are defined by Amdahl's Law[3].

On the right side, we see opposite results. The different inputs sizes do not lead to different speedups. This means that the input sizes assign enough work to the processor, at least up to the point of the highest speedup. Beyond that maximum value, on 20 and 24 cores respectively for the Sudoku and Ray, the performance drops because of the extensive overhead introduced by the garbage collection. In our data, the GHC profiling tool shows that alongside with the number of capabilities, the overall amount of work was increased which means that the cores were doing work but not useful one. The side effect of this phenomenon of doing not useful work impact memory which leads to extensive garbage collections. The critical point is between 20 to 27 cores, in that points the overhead of thread creation, inter-process communication and amount of useless work force the application to perform worse. In this occasion, the option to speedup the application lies in the application code and the parallelization strategies. Unfortunately, the applications are not able to scale up infinitely. In both cases, the speedup and the cores have logarithmic relationship core numbers lower than 28. Finally, the rate of speed-down is similar for both benchmarks with an exception at the 56 cores.

The speedup graphs showed a lot of information about the benchmark performances. The nine benchmarks we split into two categories that match the either the "left" or "right" side of the figure. The classification is based on the level of garbage collection and the different input size's impact. The applications NBody, Euler summation, Matrix Multiplication and Queens benchmarks suffer from low number of garbage collection and achieve gradually better speedup to higher numbers of the input sizes vectors while the Sudoku, Coins, Mandelbrot, Ray and Parallel RSA introduce low productivity and relatively low variation across the different input sizes. The table 5.2 summarizes the performance results of the benchmarks. It illustrates the maximum speedup of each application along with the number of available cores and input size that it was achieved on.

Benchmark	Max Speedup Value	Cores	Input Size
NBody	27.73	56	250000
Euler Summation	22.69	48	50000
Sudoku	6.38	24	16000
Ray	6.56	24	3400
Matrix Mult.	20.41	48	5000
Queens	7.83	40	14
Coins	9.00	27	1800
Mandelbrot	8.45	27	3500
PRSA	6.65	16	9000000

Table 5.1: Maximum speedup values along with the information about the problem size and number of cores of all benchmarks.

5.3 Benchmarks' Energy consumption

Analyzing the benchmarks performance wasn't necessary in order to proceed with the energy consumption results. However, we were able to identify the scalability of the benchmarks. To keep the chapter consistent we will follow the same section structure, focused on the same benchmarks for the power consumption analysis.

The benchmarks have been executed and the total energy that CPU package consumed was captured. Figure 5.3 shows the total energy consumption of Nbody, Euler Summation, Sudoku and Ray applications. On the $y-axis$ of the graphs there is the total consumed energy by the CPU. The energy values correspond to the total energy that CPU package consumed during the benchmark execution and they are measured in Joules. On the $x-axis$ there is the number of cores.

5.3.1 Graphs' Interpretation

Although the energy consumption graph is not able to share with us a lot of information, we can spot two different behaviours, again, on the left and right side. In the left side, the NBody and Euler Summation drop their energy requirements as the number of cores increases while on the right side the Sudoku and Ray applications reach the lowest point at 8 to 12 cores. Contrary to the time execution graphs, we see different energy behaviours according to the given problem sizes only on the right side.

Even though the information is very limited in the energy consumption graphs, we can view a similarity between the energy consumption and execution time. The figures 5.1 and 5.3 look relatively similar. The curves in the graphs resemble, whilst their meaning is different. In the first case, it illustrates the execution time and in the second case the energy consumption. These graphs were welcomed in our research because they indicate a strong positive correlation of execution time and energy consumption.

Similarly to the Execution time section, we proceed to the "tuning" of the energy line chart in order to extract more information from lower input sizes. After our observation that Energy Consumption is higher in the lower number of cores, which is similar to the execution time, we just applied our, previously used, “performance” tuning technique on the energy consumption. Figure 5.4 depicts the level of energy reduction across the different sizes and cores. Equation 5.2 describes the simple tuning equation of the energy consumption variable.

$$EnergyReduction_i = Energy_1/Energy_i, i \in [1, 4, 8, 16, 24, 27, 32, 36, 40, 48, 52, 55, 56] \tag{5.2}$$

Graph Explanation

The energy reduction graphs illustrate how much the energy drops in the different core numbers. The values in the $y - axis$ represent the times of energy reduction which is explained through an example. The energy requirements for sequential execution constitutes the base of comparison point.If the sequential execution costs 100 Joules, and at 4 cores costs 50 Joules, it means that the reduction level is described by $1/0.5 = 2$. Hence, the application is twice energy friendly.

Our first notification point is that different problem sizes cause different energy reduction levels in more than 8 cores[4]. Even though in the beginning (cores values below 8) the chart lines are intertwined and they start to distinguish one more than 8 cores.

[4] This statement excludes the first two problem sizes of Euler Summation application, 1500 and 5000 because they are considered low

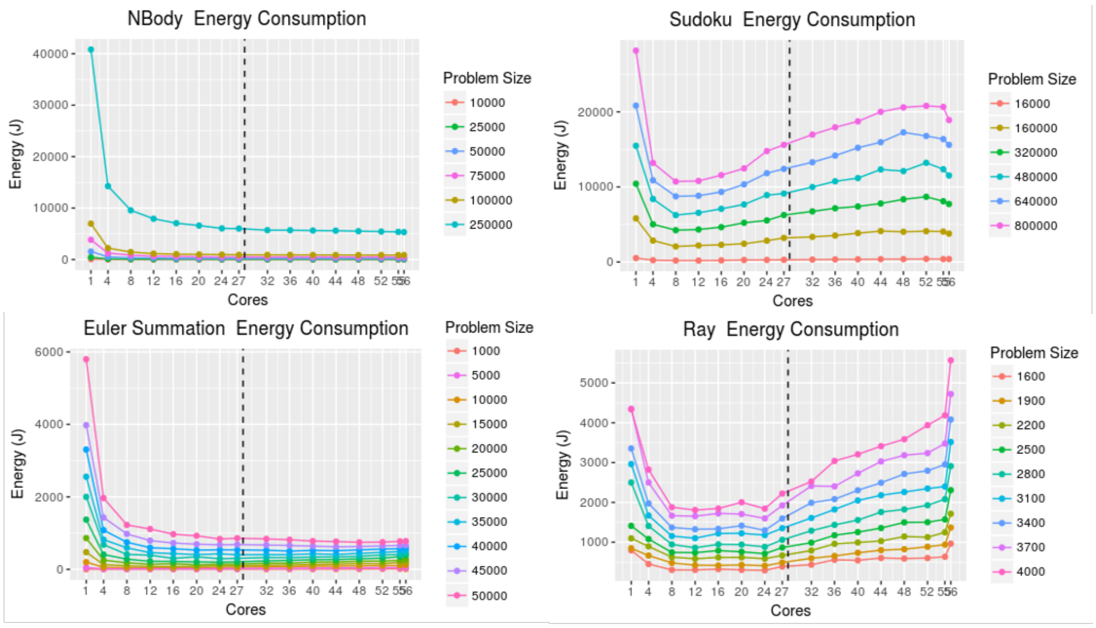


Figure 5.3: Energy consumption of the NBody, Euler Summation, Sudoku and Ray applications on multiple problem sizes.

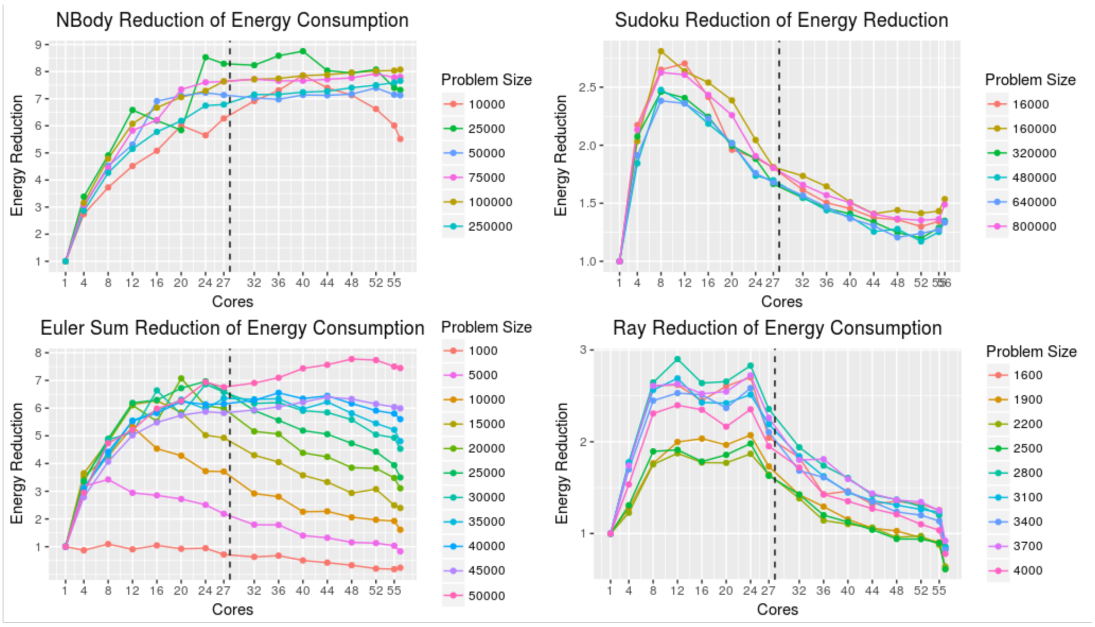


Figure 5.4: Energy reduction in the NBody, Euler Summation, Sudoku and Ray applications on multiple problem sizes.

Another observation is that the maximum reduction point for the right side (Sudoku, Ray) is achieved at the next core milestone, 12 whilst for the left side the energy is reduced while the core number is being increased. In the Euler Summation, we observe the lowest problem sizes not to follow the others. These problem sizes are considered too small because they do not assign enough workload to the available CPU's. Table 5.2 contains the maximum energy consumption values of the benchmarks. It illustrates the maximum energy reduction of each application along with the number of available cores and input size that it was obtained on.

Benchmark	Max Speedup Value	Cores	Input Size
NBody	8.75	56	250000
Euler Summation	7.77	48	50000
Sudoku	2.81	12	160000
Ray	2.90	12	3100
Matrix Mult.	7.52	52	5000
Queens	5.19	40	14
Coins	4.00	12	1800
Mandelbrot	4.15	16	2000
PRSA	3.81	12	4000000

Table 5.2: Maximum energy reduction values along with the information about the problem size and number of cores of all benchmarks.

The energy reduction graph is able to deliver more evidence about the application's energy usage. One of the most interesting results of this thesis is that the speedup and reduction energy curves look very similar. As a consequence, we will try to bring "close" these variables by investigating their relationship in the next sections.

5.4 Execution Time & Energy Consumption

The relationship of time and energy of a program execution is expected up to certain point. Even though the graphs identify this association of these variables we do not possess anything tangible yet in our hands to estimate their correlation level. Because the time and energy data are in different graphs it is difficult to identify visually their association. For a better comparison of the speedup and the energy reduction we created graphs for each benchmark that contain the speedup, the power increment and the energy reduction.

Figure 5.5 comprise four graphs of the applications NBody, Euler Summation, Sudoku and PRSA. Each of these graphs illustrates the amount of speedup with the blue line, the level of power increment with the green line and the energy reduction with the red line. The $x-axis$ illustrates the number of the execution cores and the $y-axis$ the level of reduction, speedup or increment in each case respectively. The benchmarks in the graphs are executed with the largest value in order to make sure that the application has enough amount of workload to avoid significant overheads due to sparks creation[5].

5.4.1 Speedup And Energy Reduction

According to figure 5.5, not all the application behave in similar ways. On the left side, there is the NBody and the Euler Summation benchmarks. In this cases the speedup in linear across the number of physical cores. Furthermore, we see that in the same number of cores the level of energy reduction is also increased almost with the same pace in the first 8 cores and with less in the next ones up to 27 cores.There is only one exception to this statement. Another critical observation is that in the Euler Summation at the transition from 24 to 27 cores there is speedup without energy reduction. We will return to this observation in later sections. Beyond that point, the applications achieve further speedup due to the Hyper-Threading mechanism. However, the increment rate of speedup is significantly lower and not linear for the case of Euler Summation. This case paradox is that the application gain speedup without consuming much more energy. The amount of power usage is almost the same from the 27 cores up to 56 which consists a great win for the Hyper-Threading technology. Finally, in the Euler Summation in the large core numbers, there is a speedup drop which is caused by the relatively small problem input size. These applications behave similarly because they are not affected by the garbage collection mechanism.

In contrast, the applications on the right hand (Sudoku and Ray) are suffering from extensive garbage collection overhead. Every out of four benchmarks behaves in a similar way up to 12 cores. After that point the applications on the right hand reduce the pace of speedup, unveiling a logarithmic association between the scale up and the number of cores. Beyond the 20 and 24 cores drop their performance respectively. This phenomenon is created because of the high number of cores which allocates big memory chunks forcing the garbage collection to create free space.

5.4.2 The Impact of Speedup in Energy Consumption

We already have stated multiple times our expectations, that the less execution time of a program reduces the total energy consumption by the CPU. In the transitions, from the cores 8 to 20, of the Sudoku and Ray apps show that the improved performance does not lead to better energy efficiency, which contradicts with our thoughts and already assumption. In order to comprehend why this is happening, it is necessary to analyse the benchmarks' execution processes.

The CPU power consumption is not the same for the different number of activated cores. Overall, the Intel Xeon E5-2690 v4 processor has 28 virtual cores. We estimated the power that keeps the CPU active and it is approximately 5W or 5 Joules per second. When the psychical cores are activated the CPU power requirements are increased which means that more Joules are consumed per second. Contrary to our logic, the correlation between the number of enabled cores and the CPU power is not linear (Two physical cores will not double the power usage of the CPU). As a result,

[5] Spark creation is a term used for parallel Haskell applications for creating a new thread.

in order to keep the relationship between the speedup and the energy reduction positive correlated the added cores have to improve the performance in a level that the overhead due to increased chip's power usage becomes smaller. We put our efforts to identify this threshold and it seems to be different from application to application. For instance, in the Ray application when a transition (going from 8 to 12 cores) gives at least 10% better performance the energy is reduced. In this point, we have to state that it is not expected this threshold to be the same on different scale ups because the power requirements are not variate the same from 8 to 12 cores and from 20 to 24. To generalize our results, the phenomenon of speedup and increased energy consumption is observed only in more than 8 cores and less than 27. This is expected for higher core numbers because activating virtual cores does not constitute a significant overhead in power thus if execution time is reduced then the energy will also drop and vice versa. Finally, for the cores sizes 1 to 8, because all the benchmarks provide a high speedup values we cannot provide an explanation in this project.

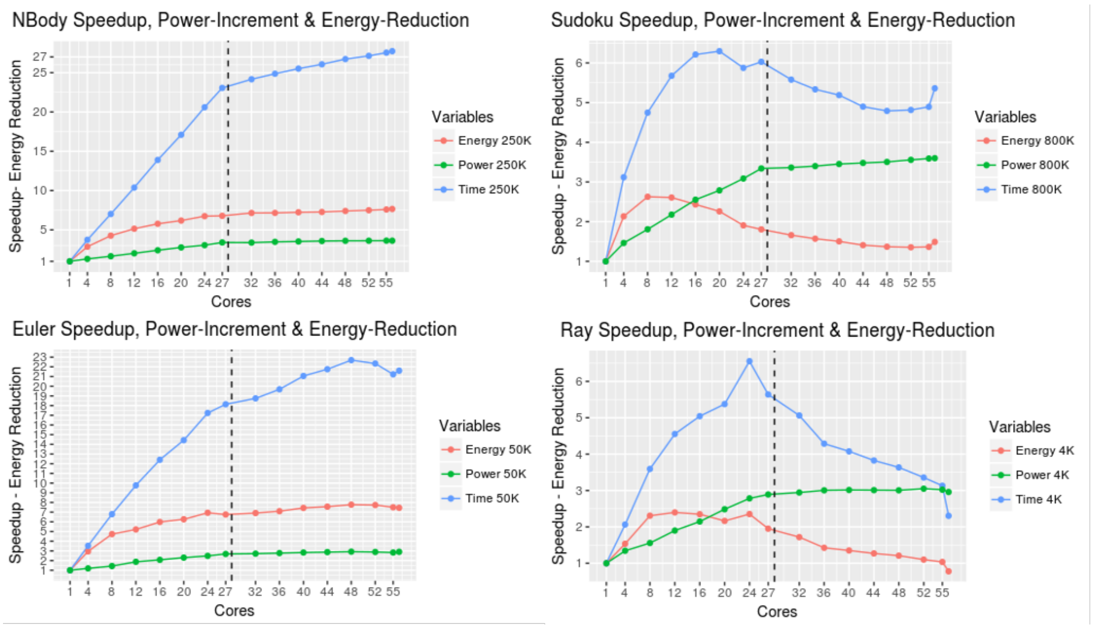


Figure 5.5: Speedup, power increment and energy reduction in the NBody, Euler Summation, Sudoku and Ray applications on the largest input size.

5.5 Euler And PRSA comparison

We investigate the Euler Summation and Parallel RSA application execution behaviour in greater depth in order to extract more information that can lead to better interpretation of the time and energy relationship. The Euler Summation scales better that the PRSA. We will try to identify the different scaling behaviour of these applications based on the information of GHC profiler and *time*[6] application that are displayed in table 5.3.

In the previous sections we claimed that Euler Summation scales better than the Parallel RSA benchmark because of less of garbage collection overhead. In order to support our claims, we performed a small benchmark comparison of the execution results of a particular case. We executed the two application in same number of cores and relatively similar workload[7]. According to table 5.3, both application allocate around 9 gigabytes of memory. However, the size of copied memory during the garbage collection is four times bigger in PRSA. In addition, the amount of time spent in garbage collection is five times bigger in PRSA due to the high number of old generation collections which requires time to collect than the new generation. This is happening because in the collection of new generation it is expected to scan more live objects in memory. Moreover, we expect more IO time spent on PRSA because of the size of memory which was copied during garbage collection. Basically, PRSA has 20% of the execution time wasted on IO while Euler Summation 14%. Finally, we can see that the time that spent of garbage collection in the PRSA case is 82%, 4.52 times bigger than the mutation (useful work) time. In contrast, Euler Summation waste the 45.7% of time to garbage collection and the 14% in IO operations. This explains the difference in the performance which imparts the energy consumption.

	Euler Summation	**Parallel RSA**
Bytes in heap	8.298 G	9.361 G
Bytes copied (GC)	3.22 M	1.365 G
Gen 0 Gen 1	1823 [12.75] (0.29) 3 [0.11] (0.00)	609 [40.79] (0.93) 395 [30.75] (0.70)
Mutation Time	15.27s (0.94s)	15.58s (0.36s)
GC Time	12.86s (0.29s)	71.54s (1.63s)
Productivity	54.3%	17.9%
Real time	1.305 s	2.106 s
User time	24.563 s	71.805 s
System time	3.594 s	15.342 s
Energy	75.86 J	173.83 J
Power	58.28 w	82.68 W

Table 5.3: Profiling Results of Euler Summation and Parallel RSA on 44 cores with 10000 and 1000000 input sizes.

These data are able to highlight the different behaviour of the benchmark appli-

[6] More information about GNU time application on http://man7.org/linux/man-pages/man1/time.1.html

[7] To achieve this, we assign a values to each benchmark that gives similar mutation time.

cations. Even though, we did not have enough time to analyse them in greater depth, we strongly believe that if we analyse more these kind of data we will be able to provide better explanations of time and energy correlation.

5.6 Time Energy Correlation

The graphs unveil a close relationship between time and energy. In order to estimate the degree of the correlation we will use R-Studio to perform different the correlation tests of Pearson [13], Kendall[14] and Spearman[6]. The three correlation tests calculate a coefficient which is comprised between -1 and 1:

- **-1** indicates a strong negative correlation : this means that every time x increases, y decreases (left panel figure 5.6).
- **0** means that there is no association between the two variables (middle panel figure 5.6).
- **1** indicates a strong positive correlation, y increases with x (right panel figure 5.6).

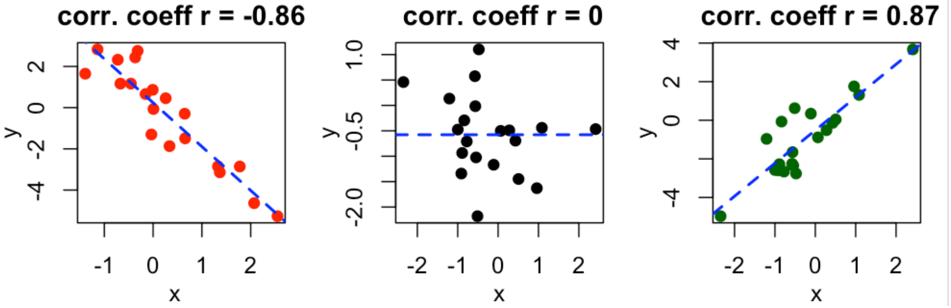


Figure 5.6: Graphical explanation of the correlation coefficients.

To test the validity of the correlation coefficients, we need to check that they are statistically significant. The two hypothesis of these tests are:

H0: The fit of the intercept-only model and regression model are equal.
H1: The fit of the intercept-only model is significantly reduced compared to the regression model.

Each correlation coefficient has a probability value. If this value is lower than the significance level, in our case $a = 0.05$, then it is considered as statistically significant which certifies its value, otherwise it is considered as zero. The benchmarks applications are tested for the energy and time correlation in table 5.4.

Benchmark	Pearson's	Kendall's	Spearman's
NBody	$0.81\ (2.2 \cdot 10^{-16})$	$0.60\ (2.2 \cdot 10^{-16})$	$0.76\ (2.2 \cdot 10^{-16})$
Euler Summation	$0.91\ (2.2 \cdot 10^{-16})$	$0.78\ (2.2 \cdot 10^{-16})$	$0.93\ (2.2 \cdot 10^{-16})$
Sudoku	$0.41\ (2.29 \cdot 10^{-5})$	$0.45\ (8.54 \cdot 10^{-11})$	$0.51\ (9.52 \cdot 10^{-8})$
Ray	$0.68\ (2.2 \cdot 10^{-16})$	$0.54\ (2.2 \cdot 10^{-16})$	$0.73\ (2.2 \cdot 10^{-16})$
Matrix Mult.	$0.95\ (2.2 \cdot 10^{-16})$	$0.82\ (2.2 \cdot 10^{-16})$	$0.95\ (2.2 \cdot 10^{-16})$
Queens	$0.38\ (1.02 \cdot 10^{-4})$	$0.28\ (6.015 \cdot 10^{-5})$	$0.41\ (3.342 \cdot 10^{-5})$
Coins	$0.61\ (1.124 \cdot 10^{-9})$	$0.43\ (1.307 \cdot 10^{-8})$	$0.58\ (1.637 \cdot 10^{-8})$
Mandelbrot	$0.61\ (9.37 \cdot 10^{-13})$	$0.40\ (2.48 \cdot 10^{-10})$	$0.55\ (3.04 \cdot 10^{-10})$
Parallel RSA	$0.72\ (2.2 \cdot 10^{-16})$	$0.64\ (2.2 \cdot 10^{-16})$	$0.78\ (2.2 \cdot 10^{-16})$

Table 5.4: Correlation Test Results. Coefficients and probabilities are provided for each application. There are three different groups based on the correlation level in speedup and reduced energy in each application.

5.6.1 Correlation Tests Results

Table5.4 includes the correlation coefficients that have been estimated by the each test. Next to the coefficients, inside the parenthesis there is the significance level of the **t-test** (p-value or probability) that help us to reject or accept the null hypothesis. The correlations tests were all successful by providing statistically positive significant values(probability values less than 0.05). Hence, the speedup and the level of energy reduction are positively correlated with a significant degree in all the applications.

Strong Correlation
the cases of NBody, Euler Summation, Matrix Multiplication and Parallel RSA encryption the correlation is considered as strong (more than 70%). In these benchmarks, we cite that the time and energy are highly associated.

Medium Strength Correlation
In the Coins, Mandelbrot and Ray application there is a clear, but not very strong association between the speedup and the energy reduction. On average, the correlation degree for these applications is around 60%

Loose Correlation
The Queens and Sudoku programs show a loose association of the variables at the level of approximately 40%.
We tried to identify the reason behind these results. Especially, on the common characteristics of the applications that provide similar coefficient values. Despite our research on the number of garbage collections, the memory requirements and the overall productivity[8] of each program, we were not able to find the exact reason why some applications provide higher correlation results than others.

[8] Productivity is the percentage of program mutation time divided by the total execution time.

5.7 Time-Energy Correlation & Hyper-threading

The graphs and the statistical analysis so far, show that there is a significant relationship between energy and the performance. During our research, different program behaviours drag our attention. On some of the graphs5.5 it seems that there are two different kinds of correlations, one on the left side with cores values less than 28 and one on the right (cores greater than 28). The critical number is the 28 because the 2.6GHz Intel Xeon E5-2690 v4 machine has 28 physical cores and with hyper-threading, the number goes up to 56 virtual cores.

The phenomenon of different application behaviours due to Hyper-Threading was judged as highly interesting and important for our research. As a consequence, we wanted to research the "time and energy" correlation with and without the Hyper-Threading even more. Hence, we split the data based on the size of the number of cores. The benchmarks that executed on less than 28 cores was separated from the rest. We didn't include the size of the 28 because we were not sure if the Hyper-Threading is enabled during that configuration. Having the data separated we expected to find different correlation of time and energy.

Figure 5.9 shows the curve fitting results on all nine applications. Each graph contains eight points that is the average speedup and energy reduction for a large input values across the set of cores (in the case of non Hyper-Threading the core vector is [1,4,8,12,16,20,24,27]). The red curve is the estimated values that come from a multilinear regression taking the logarithmic value of the speedup as the independent variable and the level of reduced energy as the dependent. In all of the cases, the logarithmic values of the speedup give better fit than the original ones which can be explained from the the previous line graphs. This means that for a speedup value 8 the energy consumption is approximately reduced by 50%. Even though the logarithmic model fits better than the linear, it is observed that from 1 and up to 8 cores association is linear with gradient value a bit less than one. This means that a speedup of value 2 reduces the energy by a factor a bit lower than 2. The overall logarithmic behaviour is obtained from 8 cores and beyond.

In general, when we assign more cores to the applications we expect them to speedup. This has the opposite effect on energy, when we assign more cores to the application the CPU consumes more power to supply power to more cores. In the ideal case of zero garbage collection, it would be very useful to define what costs more energy, better performance by using more cores or worse performance in a lower number of cores. Unfortunately, the nature of the applications includes the inevitable effect of garbage collection and other overheads making the relationship between energy and time a lot more complex. Furthermore, this is the reason that in figures 5.9 and 5.10 we included samples only from the same input size in order to keep the garbage collection overhead the same across the different number of cores. The cases that the garbage collection does not affect the execution the results are consistent amongst the various input sizes 5.7 however, the garbage collection in the applications that allocate a lot of memory can affect the execution time thus, the energy consumption may lead to unexpected levels5.8.

Summarizing for the case of curve fitting results, we state that energy reduction is growing along side with speedup(linear relationship with gradient a bit lower than 1). In the core values from 8 to 27, the correlation is logarithmic (natural logarithmic correlation).

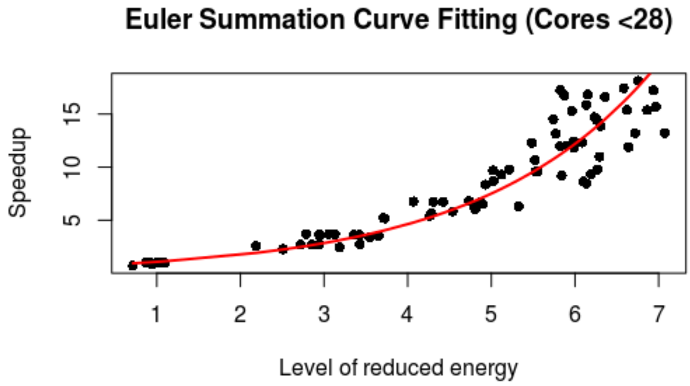


Figure 5.7: Euler Summation. Curve fitting between logarithmic values of speedup and energy reduction is consistent across all the input sizes due to less garbage collection overhead.

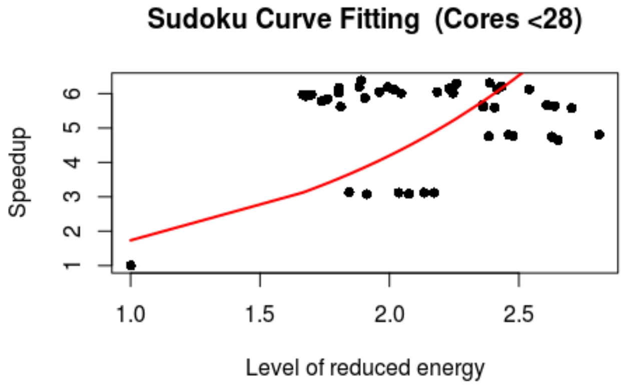


Figure 5.8: Sudoku. Curve fitting between logarithmic values of speedup and energy reduction is not consistent across all the input sizes because of the extensive garbage collection.

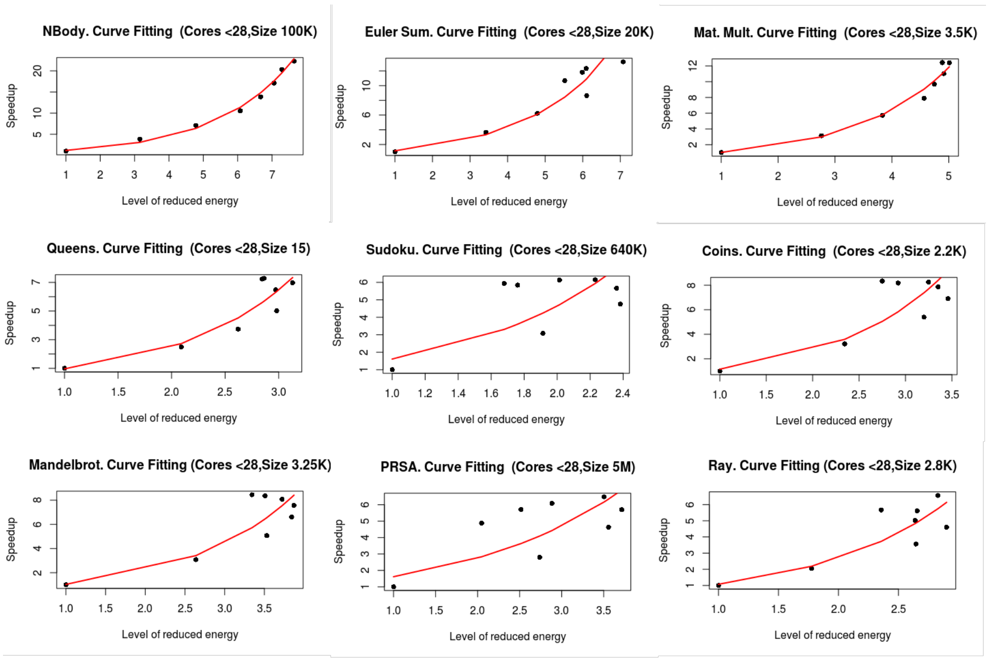


Figure 5.9: Speedup and energy reduction curve fitting..

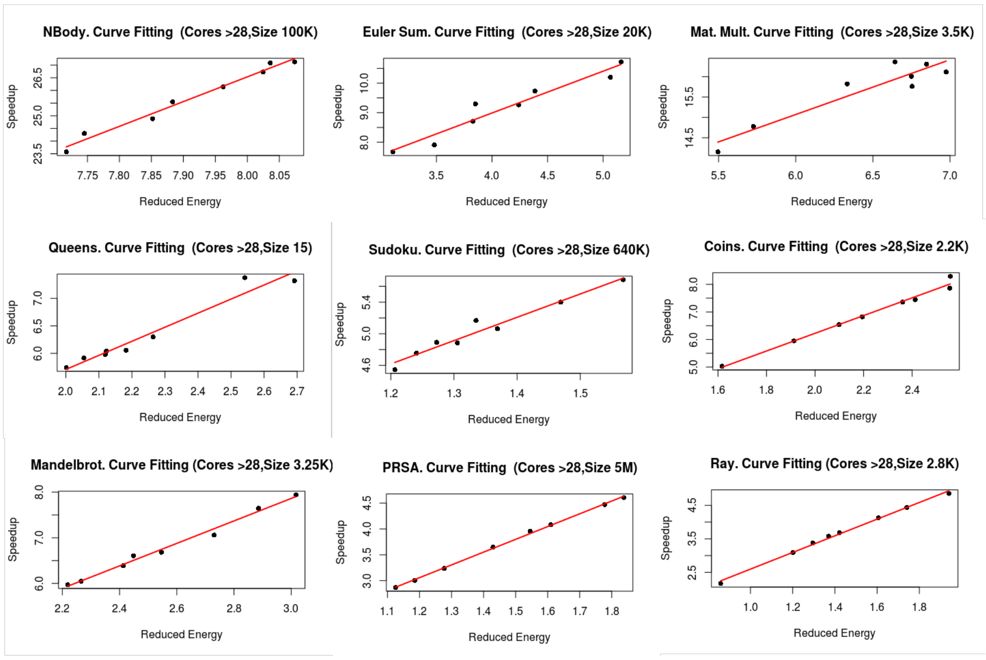


Figure 5.10: Speedup and energy reduction curve fitting with Hyper-Threading.

Regarding the Hyper-Threading, it is easier to identify the relation of the time and energy. The graphs in figure 5.10 unveil an almost perfect linear correlation between the speedup and the energy reduction. All the benchmarks, showed a linear relationship of time and energy even though the speedup in most cases drops after a specific number of cores. The correlation between the two variables is different mainly because the supply of energy does not differ significantly. The Hyper-Threading does not require more amount of power because all the physical cores are already activated and the 56 virtual cores are achieved through physical core sharing between two logical cores[16].

5.7.1 Chapter Summary

In this chapter, we identified the relationship between the time and energy. In a higher abstraction, this relationship is easy to understand, spot and comprehend through simple graphs. If we try to zoom in order to provide more accurate answers and explanations we will discover a big variety of reasons that make this research hard to execute and be scientifically correct due to lot of parameters that take place into the energy consumption equation.

This equation is complicated because it includes many other complicated variables. Despite our efforts to investigate all the possible factors of the equation. Implications of memory allocation and garbage collections are even difficult to be investigated from their own, hence we cite that this chapter can act asan introduction to a more sophisticated research. In this chapter we elaborate on our intuition and personal experience and knowledge to provide reasonable explanations. As a consequence, we cite that there is a significant correlation between the execution time and the energy consumption. This correlation varies from application to application due to their different characteristics. In general, we expect logarithmic time and energy correlation in executions that do not create more threads than the physical number of cores and a linear correlation when the Hyper-Threading is being utilized.

Chapter 6

Applications Energy Models

The final dissertation objective is to use the energy data to create energy models of the nine applications. The energy models were constructed using R-Studio. These models are hardware specific, hence the following models are valid only for the computing machine that was used to generate the benchmark data[1].

The models estimate the energy of the application without executing all possible configurations. It is not expected to be a 100% accurate, but rather give a close prediction of the actual amount. Each application energy model accepts two values as input and calculates the expected energy consumption in Joules (J). The input parameters are the number of available cores and the problem input size.

6.1 Energy Model Construction

Given the execution data for the applications, we had to create a function that given an input gives a specific output. R-Studio made that easy by performing a Multiple Least Square Regression taking the logarithmic value of the dependent variable (Energy) and the independent ones (Core number and Input Size). The variables of cores and input sizes are the input variables of this model, thus they need to be known to the user. The independent variables, are two polynomials of degree five. We used this degree to attain better fit results. The outcome of these actions will be demonstrated for each application separately. The generic formula for the energy models is shown in function 6.1. This function is divided into two for simpler illustration.

$$Energy_i(c, s) = E_c(c) + E_s(s) + e^{a_0} \tag{6.1}$$

$$E_c(c) = e^{a_{c5} \times c^5 + a_{c4} \times c^4 + a_{c3} \times c^3 + a_{c2} \times c^2 + a_{c1} \times c} \tag{6.2}$$

$$E_s(s) = e^{a_{s5} \times s^5 + a_{s5} \times s^4 + a_{s4} \times s^3 + a_{s3} \times s^2 + a_{s2} \times s} \tag{6.3}$$

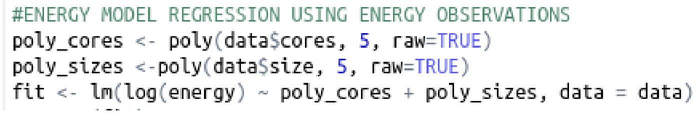
The next step is to define the application energy models is to calculate the coefficients in the exponent of the Euler number, through multiple regressions. The next sections will provide the table with the coefficients for each benchmark. We classified the nine energy models into two groups according to the type of the statistically significant coefficients, keeping always the significance level at $a = 0.05$.

- **Group A**, has energy models with all ten coefficients being statistically significant coming from both the cores and input size functions. This group includes NBody, Sudoku, Euler Summation and the Parallel RSA encryption.
- **Group B**, includes energy models with statistically significant coefficients, all coming from function 6.2 which uses the core number. The benchmarks in this group are the Matrix Multiplication, Coins, Mandelbrot and Queens application.

[1] The machine specifications and can be found in experiment setup section in chapter three

6.1.1 Obtaining The Energy Model Coefficients

In every application energy model coefficients emerged from a multilinear Least Square Regression taking as the dependent variable the logarithm of the energy value and as the independent two polynomials of the Cores vector[2] and the Sizes vector both of fifth degree. For instance, the NBody size's vector is [10000,25000,50000,75000,10000,250000]. Figure 6.1 shows the R-Studio lines of code which performs the regression in each application case.



```
#ENERGY MODEL REGRESSION USING ENERGY OBSERVATIONS
poly_cores <- poly(data$cores, 5, raw=TRUE)
poly_sizes <-poly(data$size, 5, raw=TRUE)
fit <- lm(log(energy) ~ poly_cores + poly_sizes, data = data)
```

Figure 6.1: Regression generic source code in R-Studio.

6.1.2 Guidelines in Regression Quality Assessment

Finally, having the coefficients of the energy model, it is necessary to measure their "goodness of fit". In every benchmark model subsection, there will be a figure which analyses the regression from which theses values occurred. This figure will be explained and in the end, a brief summary will be provided.

The figures that will measure the fitness of the models contain the probability of F-statistics, the R-squared and the residual standard error or the regression. The F-statistics assess the overall fit of the model. The two hypothesis are:

H0: The fit of the intercept-only model and regression model are equal.
H1: The fit of the intercept-only model is significantly reduced compared to the regression model.

If the probability of the F-statistics is less than the significance level, 0.05, the null-hypothesis is rejected. This means the model provides a better fit than the intercept-only model [3].

The R-squared value is the percentage of the response variable variation that is explained by the linear model. In general, measure how the dependent variable responds to the different input values. The max value for this metric is 1 on which the quality of the regression is expected to be good.

Finally the residual standard error (RSE) describes the expected difference between the actual values and the fitted ones from the regression. RSE is the square root, of the fraction of square the sum of the residuals, divided by the degrees of freedom[4].

$$RSE = \sqrt{\frac{\sum \hat{e}^2}{n - 11}} \tag{6.4}$$

The RSE proximity to zero measures how good is the fit of the model. For example, if the distribution of the residuals is normal and the $RSE = 0.2$ then the 65% of the predictions are in the range of ±20% of the actual value and the 95% are in ±40%. Hence, a prediction of 100 Joules claims that the actual energy requirements are between 83.3 (100/1.2) and 125 (100/0.8) Joules with 65% probability.

Because the distribution of the residuals is important for the fit measurement of the model, the regression figure will be accompanied with a plot of the residuals with the mean value represented by a horizontal line. Each plot has on the $y - axis$ the value of the residual error (values close to zero do not include any error) and the $x - axis$ has the observation index[5]. The information that we need to obtain is

[2] The core vector is the same as in the graphs and consisted of: 1, 4, 8, 12, 16, 20, 24, 27, 32, 36, 40, 44, 48, 52, 55, 56 values.

[3] The intercept-only model is the model that uses only constant values none of the coefficients that are generated from the multiple regression.

[4] Degrees of freedom are the number of observations minus eleven. We lose eleven degrees of freedom because we estimate eleven parameters in equation 6.1

[5] For example if the maximum number on the $x - axis$ is 100, that means that we used 100 observations to create our model.

the normality of the residuals. To be statistically correct about the 65% of the prediction with RSE deviation the residuals have to be normally distributed. Thus, the graph is used as a visual normality test. Normality is confirmed when the half of the residuals give larger than the mean values and the rest lower. In addition, the most of the residual values have to be accumulated around the mean, in other words, the horizontal line.

6.2 Energy Models. Group A

This group contains the energy models that are constituted by ten statistically significant coefficients of the core and size equations 6.2 6.3 respectively. In this category belong, the NBody, Sudoku, Euler Summation and the Parallel RSA encryption.

6.2.1 NBody Energy Model

Table 6.1 illustrates the coefficients of the NBody's energy model. Hence, the energy requirements can be estimated with the equation 6.1 by simply giving the set of available cores and the input size as parameters to the function.

	a_{i5}	a_{i4}	a_{i3}	a_{i2}	a_{i1}	a_0
E_c	-9.92×10^{-8}	1.63×10^{-5}	-1.00×10^{-3}	2.91×10^{-2}	-4.02×10^{-1}	–
E_s	5.17×10^{-25}	-2.66×10^{-19}	4.81×10^{-14}	-4.18×10^{-09}	2.13×10^{-04}	–
a_0	–	–	–	–	–	2.906

Table 6.1: Coefficients of NBody application energy model.

Figure 6.2 shows basic information from the regression that NBody's coefficients have emerged.

F-Statistic (prob)	R-Squared	Residual Std. Error
0	0.9983	0.09098

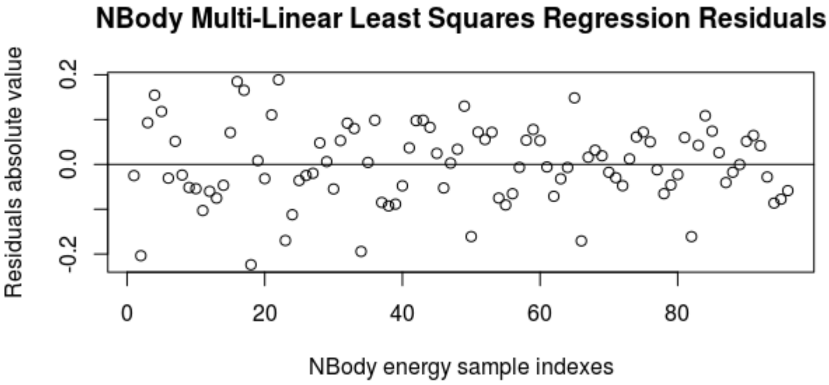


Figure 6.2: Multilinear regression of NBody application.

- **F-Statistics Probability.** Because the f-statistics probability is below 0.05 we reject the null hypothesis. This means that the model provides a better fit than the intercepted only one.
- **R-squared.** Is almost hundred percent which means that all variable variation is explained by the linear model.
- **Residual Plot.** There is not any pattern in the residuals. In addition, they confirm the normality conditions. Consequently, residuals are considered as random.
- **Residuals Std. Error** This value is the expected estimation error on average. Based on residual normality the 65% of the prediction deviates ±9.0% from the real value.

The final conclusion is that all the regression statistics show the good fit of the model with 90% proximity to the actual energy requirements of the application.

6.2.2 Euler Summation Energy Model

Euler Summation application energy model coefficients are illustrated in table 6.2. These coefficients apply to function 6.1 which calculates the estimated energy requirements of the application given the number of available cores and the input size.

	a_{i5}	a_{i4}	a_{i3}	a_{i2}	a_{i1}	a_0
E_c	-8.96×10^{-8}	1.50×10^{-5}	-9.53×10^{-4}	2.82×10^{-2}	-3.78×10^{-1}	–
E_s	2.10×10^{-25}	-3.08×10^{-19}	1.71×10^{-14}	-4.57×10^{-09}	6.64×10^{-04}	–
a_0	–	–	–	–	–	2.32

Table 6.2: Coefficients of Euler Summation application energy model.

The validity of table 6.2 is assessed from the following figure (figure 6.3).

F-Statistic (prob)	R-Squared	Residual Std. Error
0	0.9661	0.3142

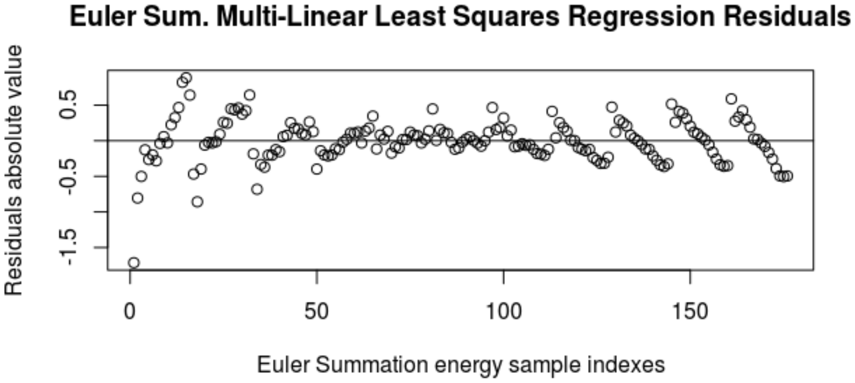


Figure 6.3: Multilinear regression of Euler Summation application

- **F-Statistics Probability.** We reject the null hypothesis. This means that the model provides a better fit than the intercepted only one.
- **R-squared.** The R-squared value is high which means that all variable variation is explained by the linear model.
- **Residual Plot.** The residuals unveil a pattern. However, they still meet the normality conditions. However, the graph is considered following the normal distribution.
- **Residuals Std. Error** Based on residual normality, the 65% of the prediction deviates ±31.4% from the real value.

Even though the high R-squared value, there is a significant error in the fit values of the model. Overall, the model is considered as 69% accurate and is expected to include values that may or may not deviate significantly from the actual consumed energy.

6.2.3 Sudoku Energy Model

Sudoku application energy model coefficients are illustrated in table 6.3. These coefficients apply to function 6.1 which calculates the estimated energy requirements of the application given the number of available cores and input size.

	a_{i5}	a_{i4}	a_{i3}	a_{i2}	a_{i1}	a_0
E_c	-9.57×10^{-8}	1.54×10^{-5}	-9.40×10^{-3}	2.62×10^{-2}	-3.05×10^{-1}	–
E_s	1.32×10^{-25}	-3.23×10^{-19}	3.03×10^{-14}	-1.38×10^{-09}	3.34×10^{-04}	–
a_0	–	–	–	–	–	5.98

Table 6.3: Coefficients of Sudoku application energy model.

The energy model coefficients are examined by the following figure (figure 6.4).

F-Statistic (prob)	R-Squared	Residual Std. Error
0	0.9991	0.04342

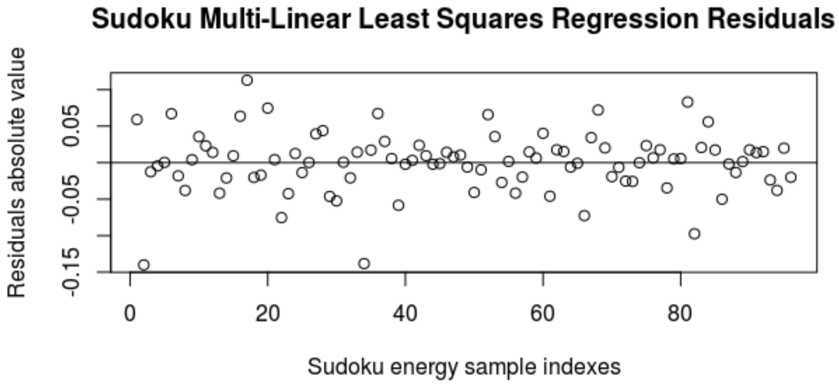


Figure 6.4: Multilinear regression of Sudoku application.

- **F-Statistics Probability.** The null hypothesis is rejected. This means that the model provides a better fit than the intercepted only one.
- **R-squared.** The R-squared value is high which means that all variable variation is explained by the linear model.
- **Residual Plot.** There is not any pattern in the residuals. In addition, they confirm the normality conditions. As a result, the residuals are considered as normal distributed.
- **Residuals Std. Error** This value is the expected estimation error. Based on residual normality the 65% of the prediction deviates ±4.3% from the real value on average.

The regression statistics display the high fit of the model, with estimated quality of 95.7%. The model is expected to produce values that will not deviate significantly from the actual consumed energy values.

6.2.4 PRSA Energy Model

Parallel RSA application energy model coefficients are illustrated in table 6.4. These coefficients apply to function 6.1 which calculates the estimated energy requirements of the application given the number of available cores and input size.

	a_{i5}	a_{i4}	a_{i3}	a_{i2}	a_{i1}	a_0
E_c	-6.69×10^{-8}	1.24×10^{-5}	-8.67×10^{-3}	2.74×10^{-2}	-3.55×10^{-1}	–
E_s	1.35×10^{-34}	-4.28×10^{-27}	5.34×10^{-20}	-3.44×10^{-13}	1.36×10^{-06}	–
a_0	–	–	–	–	–	4.721

Table 6.4: Coefficients of Parallel RSA application energy model.

The energy model coefficients are assessed by the following figure (figure 6.5).

F-Statistic (prob)	R-Squared	Residual Std. Error
0	0.9935	0.06704

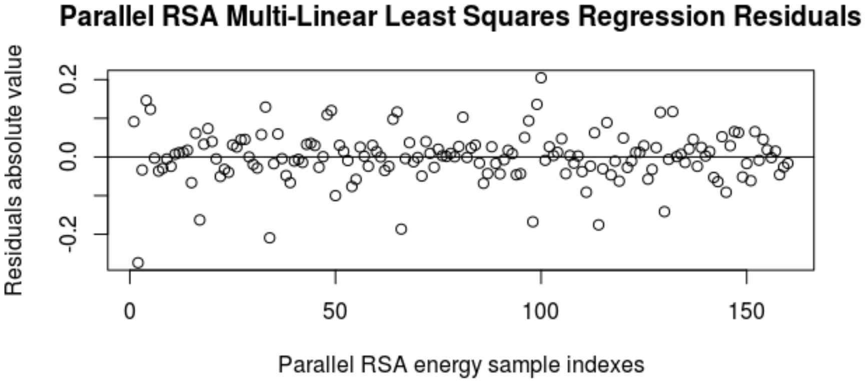


Figure 6.5: Multilinear regression of Parallel RSA application.

- **F-Statistics Probability.** The null hypothesis is rejected., thus we state the model provides a better fit than the intercepted only one.
- **R-squared.** The R-squared value is high which means that all variable variation is explained by the linear model.
- **Residual Plot.** There is no spotted pattern in the residuals. Furthermore, they meet the normality conditions. As a result, we claim that they follow the normal distribution.
- **Residuals Std. Error** This value is the average expected estimation error. Based on residual normality the 65% of the prediction deviates ±6.7% from the real value on average.

The regression statistics illustrate the good fit of the model, rated as 93.3% accurate. It is expected to forecast values that are not going to differ significantly from the actual consumed energy values. The 2/3 of the prediction is going to deviate 6.7% from the original values.

6.3 Energy Models. Group B

This group contains the energy models that are constituted only by the coefficients of the equation 6.2. The equation 6.3 consists of not statistically significant values which are practically equal to zero. In this category are the Matrix Multiplication, Coins, Mandelbrot and Queens applications.

6.3.1 Matrix Multiplication Energy Model

Matrix Multiplication application energy model coefficients are illustrated in table 6.5. These coefficients apply to function 6.1 which calculates the estimated energy requirements of the application given the number of available cores and input size.

	a_{i5}	a_{i4}	a_{i3}	a_{i2}	a_{i1}	a_0
E_c	-8.73×10^{-8}	1.49×10^{-5}	-9.48×10^{-3}	2.75×10^{-2}	-3.63×10^{-1}	–
E_s	0	0	0	0	0	–
a_0	–	–	–	–	–	5.586

Table 6.5: Coefficients of Matrix Multiplication application energy model.

This energy model include the coefficients that are statistically significant. The other values have been constituted with zeros. The model's goodness of fit is assessed by the following figure (figure 6.6).

F-Statistic (prob)	R-Squared	Residual Std. Error
0	0.9882	0.1289

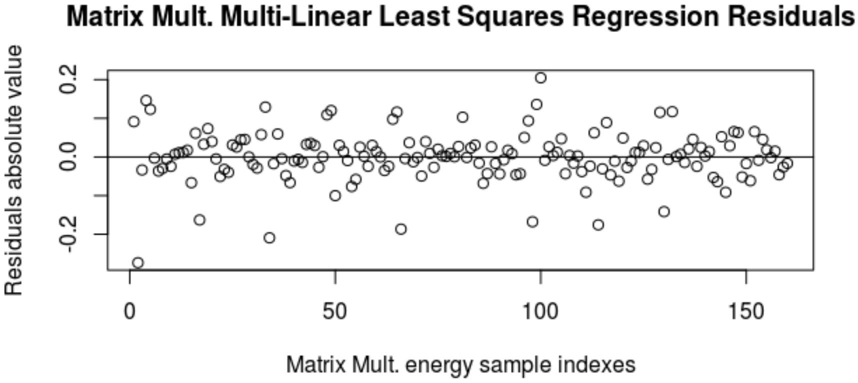


Figure 6.6: Multilinear regression of Matrix Multiplication application.

- **F-Statistics Probability.** The null hypothesis is rejected. This means that the model provides a better fit than the intercepted only one.
- **R-squared.** The R-squared value is close to highest value. Thus, all variable variation is explained by the linear model.
- **Residual Plot.** There is no spotted pattern in the residuals. Moreover, they meet the normality conditions. Hence, we cite that they are random and normal.
- **Residuals Std. Error** This value is the average expected estimation error. Based on residual normality the 65% of the prediction deviates ±12.9% from the real value on average.

The regression statistics highlight the low quality fit of the model, rated as 87.1% accurate. The 2/3 of the fit values deviate less than 13% from the real values.

6.3.2 Coins Energy Model

Coins application energy model coefficients are illustrated in table 6.6. These coefficients apply to function 6.1 which calculate the estimated energy requirements of the application given the number of available cores and input size.

	a_{i5}	a_{i4}	a_{i3}	a_{i2}	a_{i1}	a_0
E_c	-1.17×10^{-8}	1.81×10^{-5}	-1.06×10^{-3}	2.89×10^{-2}	-3.52×10^{-1}	–
E_s	0	0	0	0	0	–
a_0	–	–	–	–	–	1.208

Table 6.6: Coefficients of Coins application energy model.

The coefficients are assessed for their validity below in figure 6.7.

F-Statistic (prob)	R-Squared	Residual Std. Error
0	0.9897	0.07917

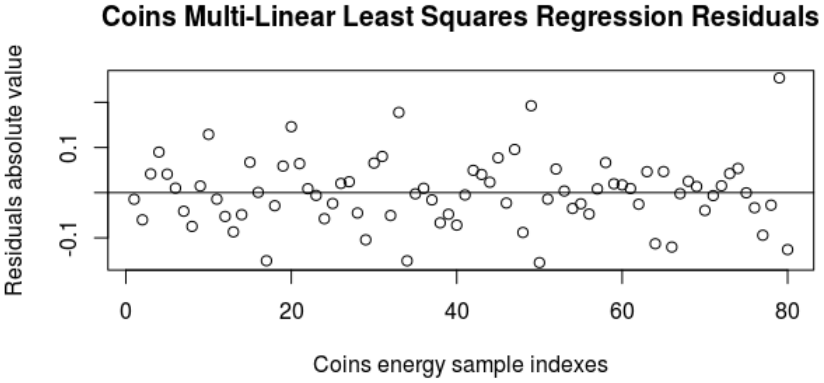


Figure 6.7: Multilinear regression of Coins application.

- **F-Statistics Probability.** The null hypothesis is rejected. This means that the model provides a better fit than the intercepted only one.
- **R-squared.** The R-squared value is close to optimal standards which mean that all variable variation is explained by the linear model.
- **Residual Plot.** There is no spotted pattern in the residuals. Also, they meet the normality specification. Hence, we cite that they are random and normal.
- **Residuals Std. Error** This value is the average expected estimation error. Based on residual normality the 65% of the prediction deviates ±7.9% from the real value on average.

The regression statistics unveil the good fit of the model having the 65% of the estimated values close in the real energy requirement values.

6.3.3 Mandelbrot Energy Model

Mandelbrot application energy model coefficients are illustrated in table 6.7. These coefficients apply to function 6.1 which calculates the estimated energy requirements of the application given the number of available cores and input size.

	a_{i5}	a_{i4}	a_{i3}	a_{i2}	a_{i1}	a_0
E_c	-1.04×10^{-8}	1.68×10^{-5}	-1.02×10^{-3}	2.91×10^{-2}	-3.65×10^{-1}	–
E_s	0	0	0	0	0	–
a_0	–	–	–	–	–	-3.18

Table 6.7: Coefficients of Mandelbrot application energy model.

The energy model coefficients in table 6.7 were calculated from a multilinear regression which statistics are shown below in figure 6.8.

F-Statistic (prob)	R-Squared	Residual Std. Error
0	0.9867	0.05899

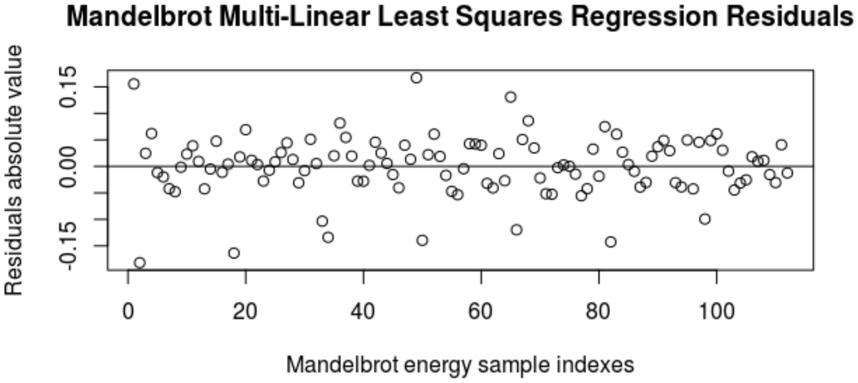


Figure 6.8: Multilinear regression of Mandelbrot application.

- **F-Statistics Probability.** The null hypothesis is rejected. This means that the model provides a better fit than the intercepted only one.
- **R-squared.** The R-squared value is close to optimal standards. Almost all of the variable variation is explained by the linear model.
- **Residual Plot.** There is no spotted pattern in the residuals. In addition, they meet the normality standards. Hence, we cite that they are random and normal.
- **Residuals Std. Error** Based on residual normality, the 65% of the prediction values deviate ±5.8% from the original values on average.

The verdict for the Mandelbrot's energy model is that the produced results are close to the original values. This model is considered having 94.6% quality results.

6.3.4 Queens Energy Model

Queens application energy model coefficients are illustrated in table 6.8. These coefficients apply to function 6.1 which calculate the estimated energy requirements of the application given the number of available cores and input size.

	a_{i5}	a_{i4}	a_{i3}	a_{i2}	a_{i1}	a_0
E_c	-8.41×10^{-8}	1.33×10^{-5}	-7.96×10^{-3}	2.25×10^{-2}	-2.92×10^{-1}	–
E_s	0	0	0	0	0	–
a_0	–	–	–	–	–	3.720

Table 6.8: Coefficients of Queens application energy model.

Queen's energy model is tested below by the figure 6.8.

F-Statistic (prob)	R-Squared	Residual Std. Error
0	0.9964	0.1975

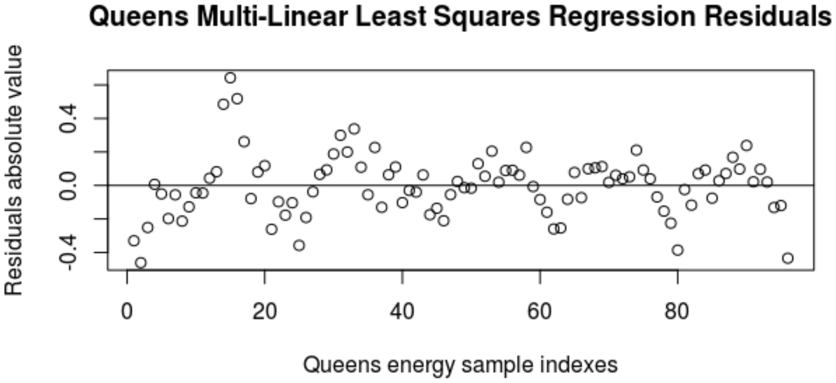


Figure 6.9: Multilinear regression of Queens application.

- **F-Statistics Probability.** The null hypothesis is rejected. This means that the model provides a better fit than the intercepted only one.
- **R-squared.** The R-squared value is close to optimal (one) which mean that all variable variation is explained by the linear model.
- **Residual Plot.** There is no spotted pattern in the residuals. Furthermore, they meet the normality conditions. As a result, we state that they are random and normal.
- **Residuals Std. Error** This value is the expected estimation error on average. Based on residual normality the 65% of the prediction deviates ±19.7% from the real value on average.

The regression statistics show the low fit of the Queens model. The predictions of this model are not considered to be close to real values.

6.3.5 Ray Energy Model

Ray application has only two coefficients that are statistically significant. Ray's energy model coefficients are illustrated in table 6.9. These coefficients apply to function 6.1 which calculate the estimated energy requirements of the application given the number of available cores and input size.

	a_{i5}	a_{i4}	a_{i3}	a_{i2}	a_{i1}	a_0
E_c	7.47×10^{-8}	-1.60×10^{-5}	0	0	0	–
E_s	0	0	0	0	0	–
a_0	–	–	–	–	–	6.218

Table 6.9: Coefficients of Ray application energy model.

This, last energy model is tested below in figure 6.10.

F-Statistic (prob)	R-Squared	Residual Std. Error
827.3 ($< 2.2 \cdot 10^{-16}$)	0.9842	0.08856

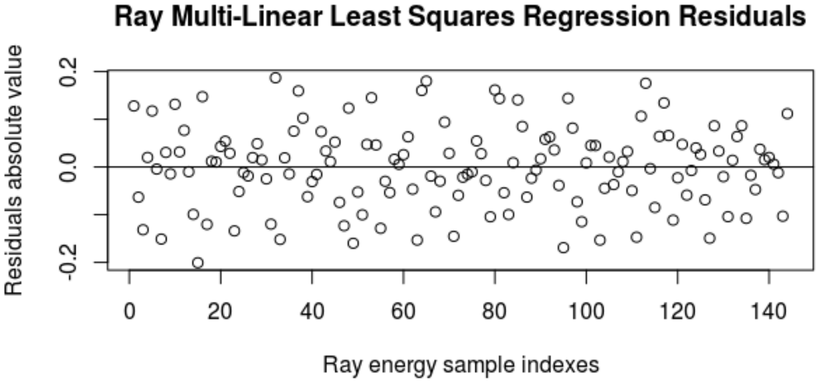


Figure 6.10: Multilinear regression of Ray application.

- **F-Statistics Probability.** The null hypothesis is rejected. This means that the model provides a better fit than the intercepted only one.
- **R-squared.** The R-squared value is close to one which means that all variable variation is explained by the linear model.
- **Residual Plot.** There is no spotted pattern in the residuals. Moreover, they pass the visual normality test. They considered random and normal.
- **Residuals Std. Error** Based on residual normality the 65% of the prediction deviates ±8.8% from the real value on average.in the distribution of the residuals.

The regression statistics display the good fit of the Ray model with score 91.2%. We state that Ray's energy model is expected to provide estimated values that are close to real ones.

6.4 Chapter Summary

This chapter presented the constructed energy models of the parallel Haskell applications with were used as benchmarks. Overall, we divide the energy models into two groups based on the type of statistically significant coefficients.

All the energy models rejected the null hypothesis of the f-stat (analysed in the beginning of this chapter) and had high R-squared values which indicate the good fit of the model. However, for the cases of the Euler Summation (31% error), Queens (19% error) and the Matrix Multiplication (12% error) the difference from the original values is considered significant (greater than 10%). In these cases, there were made an effort to identify the reasons creating this phenomenon. We focused our investigation the structure of these three applications and what makes them different from the other six. Despite our search, we did not find any unique combination of characteristics that makes them special. As a consequence, we are unable to provide a reasonable explanation and we let this question open for further research.

This was the last objective of this thesis. By providing these models we enable the approximation of those nine benchmark applications energy requirements, on the "Corryvreckan", University of St. Andrews, machine. These energy models are specific for this machine and should not be considered as generic.

Chapter 7

Future Work

The future work includes extension of practices used by this disseration and continuation of the energy and time relationship on the memory level.

7.1 Core Affinity

The conducted experiment of the CPU pinning can be extended by increasing the number of benchmarks and the repetition of each in order to obtain more data and better quality in the statistical analyses. Moreover, it can be studied how the Linux Kernel system calls achieve the core affinity and what are the implications of core affinity in applications written in different programming languages and systems.

7.2 Greater Analysis on the Application Behaviour

By running the nine benchmarks, we generated a few hundreds of megabytes of data. We did our best to provide scientifically correct explanations and highlighting important findings. In some cases and despite the extensive data analysis, it was not able to provide valid answers and reasonable explanations. These cases were highlighted and they are left for future researchers to solve them. Most of these, were about the benchmarks' behaviour and the impact of the garbage collection in the program execution. This is considered as highly important because if we are able to comprehend the garbage collection impact on execution, the time and energy variables become tightly coupled. This would eradicate the need for treating time and energy separately, saving time and effort.

7.3 Energy Consumption in Memory

The research on energy provided on this thesis is measured by the RAPL interface that is supported by the CPU architecture. All the amount of energy comes from the CPU. However, RAPL interface also supports measuring the energy that is used from the DRAM. Hence, the research can be transferred on the memory using the same interface to obtain application energy data and perform similar data analysis to extract the impact of speedup to memory energy consumption. We are going to motivate the future researchers by commencing this research immigration to DRAM level.

Firstly, we searched the online source code repositories for applications that utilize RAPL interface to extract energy information on the machine's memory. Finally, we decided to use an application called "rapl-plot"[1] which read the necessary registers from memory and displays the power of the whole DRAM package. In contrast, with the AppPowerMeter application which was used in the CPU experiment, the rapl-plot had to be executed along side with the benchmark application. Because of this, we expect the energy measurement to contain some error. This was not evaluated as important to fix because this last section is more or less a suggestion.

[1] The application is available on https://github.com/deater/uarch-configure

The experiment includes only the NBody application as a benchmark. Nbody was considered a really good candidate for this experiment because of the low level of garbage collection impact on the overall execution. In all cases, NBody achieves productivity more than 95%. Having the execution and energy data in a text file with several thousands number of lines, just import into R-Studio and copied some of the written source code in order to do the analysis. Figure 7.1 includes a line chart that illustrates the energy reduce level of all test cases across the different number of cores. This figure constitutes the starting point of our recommended way to commence the data analysis of DRAM power usage. It a clear that the graph is almost identical to the one with energy values coming from the CPU package. The difference is the level of the reduced energy. in the case of memory is 16.5, exactly double than the one obtained from CPU (8.75). Figure 7.2 isolates the speedup and the energy reduce of a single input problem size to illustrate their relation. As previous, the graphs are similar with the ones from CPU with the same exception, the double energy reduce values. In addition, the different correlations caused by the effect of Hyper-Threading are also visible. Finally in the case of DRAM, we expect a more linear relation than CPU between the time and energy requirements in the lower number of cores.

Even though the first results look similar situation in DRAM, we cannot generalize our early findings. This is left as question that has to be answered by future work.

In conclusion, this chapter provides a set of ideas that can guide other researchers effort to investigate in greater depth and breadth the correlation of time and energy in multi-core architectures.

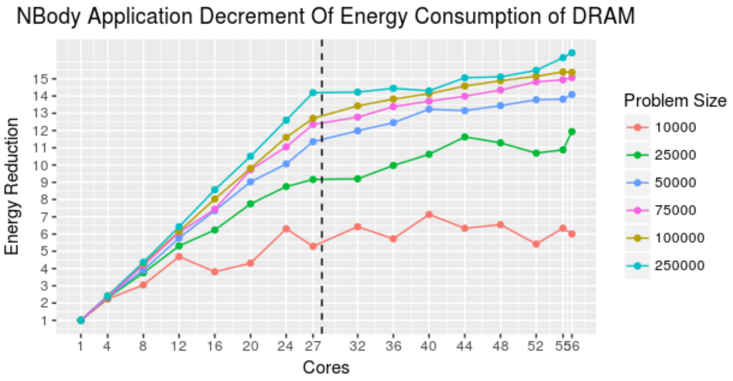


Figure 7.1: Energy reduce on DRAM, in the NBody application

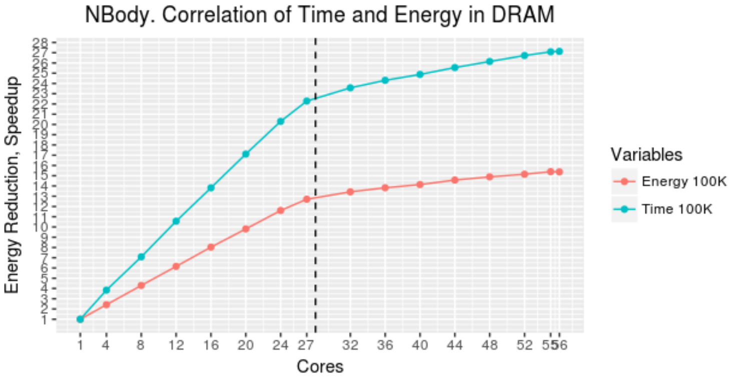


Figure 7.2: Energy reduce on DRAM, in the NBody application

Conclusion

The major goal of this dissertation was to identify the relationship between parallel application's performance and energy usage. Our findings state that the time variable is strongly correlated with the CPU's power usage meaning that the less time the application takes to executes the less energy it consumes due to better CPU utilization. Even though higher CPU utilization leads to higher energy consumption per second, overall we benefit from the less execution time. This consists the most simple explanation of our findings that we can provide. These results make unnecessary the separate investigation of time and energy because the energy requirements of an application can be predicted by its performance and vice versa.

It would be naive to state that these are our final result. Even though the performance and power usage are strongly associated, our findings showed that under certain circumstances the performance can be improved while energy consumption is increased. These circumstances can be highly complicated because they bring different factors in the equation. The most important factor is the impact of garbage collection mechanism in the execution of a program. Having this mechanism in the equation, we need to take into account the application's workload and the strategies which were used for parallelization. This phenomenon constitutes a barrier on how accurate can be the prediction of energy requirements based on performance and vice versa.

Despite these obstacles, our thesis accomplishes to generate a unified equation that is able to forecast energy requirements of nine applications with relatively high accuracy.

Bibliography

[1] Haskell in Green Land: Analyzing the Energy Behavior of a Purely Functional Language. *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 517–528, 2016.

[2] RedHat Linux Enterprise 7.4. Turbostat documentation, performance monitoring tools. 2008.

[3] Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), pages 483–485, New York, NY, USA, 1967. ACM.

[4] Adam D Barwell and Kevin Hammond. Obliterating Obstructions : Discovering Potential Parallelism by Slicing for Dependencies in Recursive Functions.

[5] Carlos Bernal-Cárdenas. Improving energy consumption in android apps. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, pages 1048–1050, New York, NY, USA, 2015. ACM.

[6] Spearman C. The proof and measurement of association between two things. *The American Journal of Psychology*, 15(1):72–101, 1904.

[7] T. Ilsche D. Molka J. Schuchart D. Hackenberg, R. Schöne and R. Geyer. An energy efficiency feature survey of the intel haswell processor. In *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*, pages 896–904, May 2015.

[8] S. Das, J. R. Doppa, D. H. Kim, P. P. Pande, and K. Chakrabarty. Optimizing 3d noc design for energy efficiency: A machine learning approach. In *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 705–712, Nov 2015.

[9] Spencer Desrochers, Chad Paradis, and Vincent M. Weaver. A validation of dram rapl power measurements. In *Proceedings of the Second International Symposium on Memory Systems*, MEMSYS '16, pages 455–470, New York, NY, USA, 2016. ACM.

[10] Intel Open Source Group. Powertop software utility. 2007. Available at https://01.org/powertop.

[11] Intel Open Source Group. Thermal management for intel® xeon® processors. 2017. Available at https://www.intel.co.uk/content/www/uk/en/support/processors/000006710.html6.

[12] Abram Hindle. Green mining: A methodology of relating software change to power consumption. In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*, MSR '12, pages 78–87, Piscataway, NJ, USA, 2012. IEEE Press.

[13] Pearson K. Note on regression and inheritance in the case of two parents. pages 240–242, 1895. Available at http://dx.doi.org/10.1098/rspl.1895.0041.

[14] M. G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1-2):81–93, 1938.

[15] Laurent Lefevre. Impact of application and service knowledge for energy efficiency in hpc. In *Proceedings of the 2013 Workshop on Energy Efficient High Performance Parallel and Distributed Computing*, EEHPDC '13, pages 1–2, New York, NY, USA, 2013. ACM.

[16] Deborah T. Marr, Frank Binns, David L. Hill, Glenn Hinton, David A. Koufaty, Alan J. Miller, and Michael Upton. Hyper-Threading Technology Architecture and Microarchitecture. *Intel Technology Journal*, 6(1):4–15, February 2002.

[17] Adel Noureddine, Aurelien Bourdon, Romain Rouvoy, and Lionel Seinturier. A preliminary study of the impact of software engineering on greenit. In *Proceedings of the First International Workshop on Green and Sustainable Software*, GREENS '12, pages 21–27, Piscataway, NJ, USA, 2012. IEEE Press.

[18] Gustavo Pinto. Do language constructs for concurrent execution have impact on energy efficiency? In *Proceedings of the 2013 Companion Publication for Conference on Systems, Programming, &#38; Applications: Software for Humanity*, SPLASH '13, pages 121–122, New York, NY, USA, 2013. ACM.

[19] V. Tiwari, S. Malik, and A. Wolfe. Power analysis of embedded software: A first step towards software power minimization. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 384–390, Nov 1994.

[20] Wanghong Yuan and Klara Nahrstedt. Energy-efficient soft real-time cpu scheduling for mobile multimedia systems. *SIGOPS Oper. Syst. Rev.*, 37(5):149–163, October 2003.

UNIVERSITY OF ST ANDREWS
TEACHING AND RESEARCH ETHICS COMMITTEE (UTREC)
SCHOOL OF COMPUTER SCIENCE
PRELIMINARY ETHICS SELF-ASSESSMENT FORM

This Preliminary Ethics Self-Assessment Form is to be conducted by the researcher, and completed in conjunction with the Guidelines for Ethical Research Practice. All staff and students of the School of Computer Science must complete it prior to commencing research.

This Form will act as a formal record of your ethical considerations.
Tick one box

□ **Staff Project**
☑ **Postgraduate Project**
□ **Undergraduate Project**

Title of project

PARALLEL RESOURCE MONITORING IN HASKELL

Name of researcher(s)

THEODOROS CHRISTOS DIMOPOULOS

Name of supervisor (for student research)

KEVIN HAMMOND

OVERALL ASSESSMENT (to be signed after questions, overleaf, have been completed)

Self audit has been conducted **YES** ☑ **NO** □

There are no ethical issues raised by this project

Signature Student or Researcher

Print Name

THEODOROS CHRISTOS DIMOPOULOS

Date

13/4/17

Signature Lead Researcher or Supervisor

K Hammond

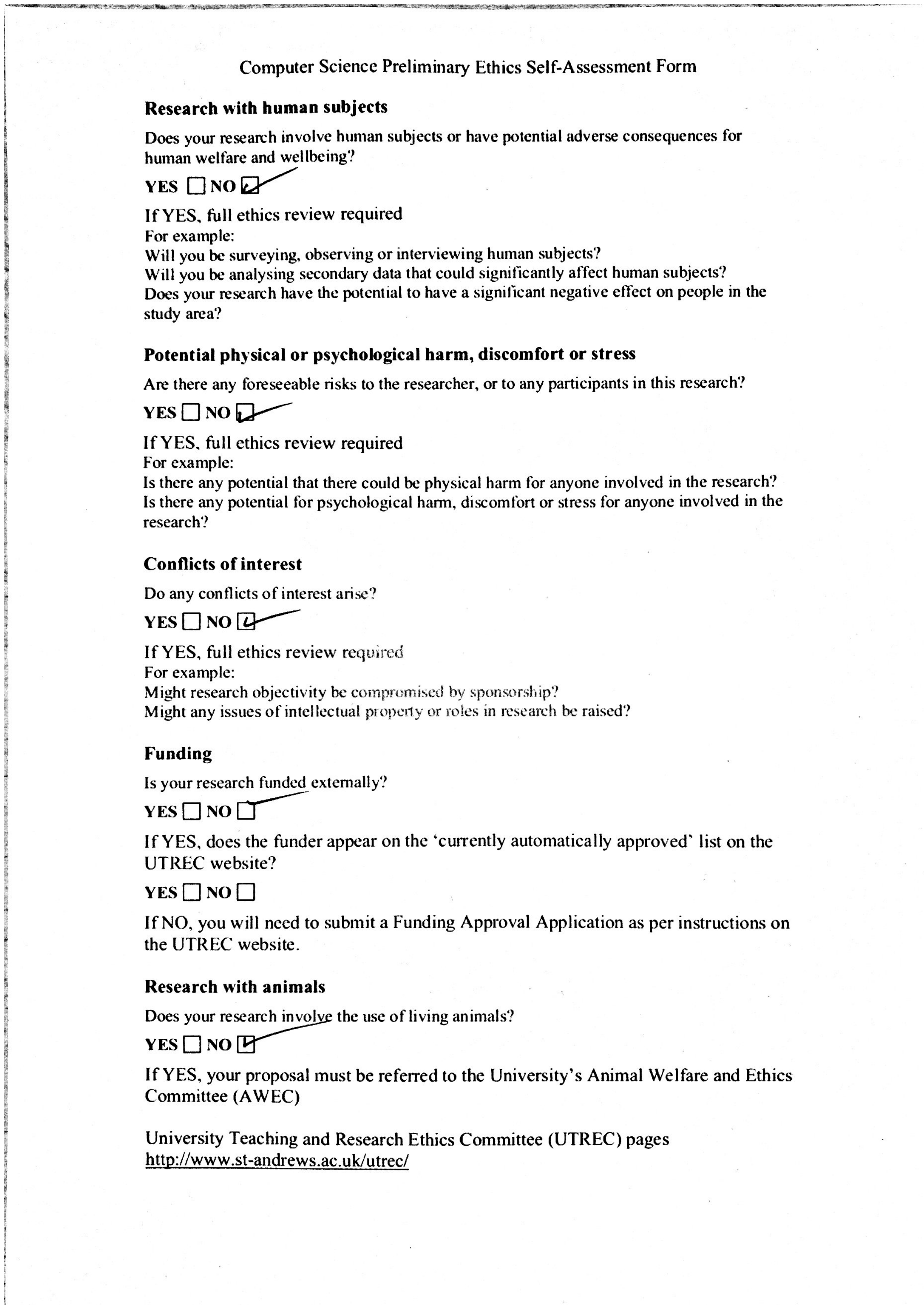
Print Name

KEVIN HAMMOND

Date

13/4/17

This form must be date stamped and held in the files of the Lead Researcher or Supervisor. If fieldwork is required, a copy must also be lodged with appropriate Risk Assessment forms. The School Ethics Committee will be responsible for monitoring assessments.



Computer Science Preliminary Ethics Self-Assessment Form

Research with human subjects

Does your research involve human subjects or have potential adverse consequences for human welfare and wellbeing?

YES □ **NO** ☑

If YES, full ethics review required
For example:
Will you be surveying, observing or interviewing human subjects?
Will you be analysing secondary data that could significantly affect human subjects?
Does your research have the potential to have a significant negative effect on people in the study area?

Potential physical or psychological harm, discomfort or stress

Are there any foreseeable risks to the researcher, or to any participants in this research?

YES □ **NO** ☑

If YES, full ethics review required
For example:
Is there any potential that there could be physical harm for anyone involved in the research?
Is there any potential for psychological harm, discomfort or stress for anyone involved in the research?

Conflicts of interest

Do any conflicts of interest arise?

YES □ **NO** ☑

If YES, full ethics review required
For example:
Might research objectivity be compromised by sponsorship?
Might any issues of intellectual property or roles in research be raised?

Funding

Is your research funded externally?

YES □ **NO** ☑

If YES, does the funder appear on the 'currently automatically approved' list on the UTREC website?

YES □ **NO** □

If NO, you will need to submit a Funding Approval Application as per instructions on the UTREC website.

Research with animals

Does your research involve the use of living animals?

YES □ **NO** ☑

If YES, your proposal must be referred to the University's Animal Welfare and Ethics Committee (AWEC)

University Teaching and Research Ethics Committee (UTREC) pages
http://www.st-andrews.ac.uk/utrec/