UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

EDSON LUIZ PADOIN

Energy-aware load balancing approaches to improve energy efficiency on HPC systems

Thesis presented in partial fulfillment
of the requirements for the degree of
Doctor of Computer Science

Advisor: Prof. Dr. Philippe Olivier Alexandre Navaux
Advisor during Ph.D.
Internship: Prof. Dr. Jean-François Méhaut

Porto Alegre
April 2016

CIP — CATALOGING-IN-PUBLICATION

Padoin, Edson Luiz

Energy-aware load balancing approaches to improve energy efficiency on HPC systems / Edson Luiz Padoin. – Porto Alegre: PPGC da UFRGS, 2016.

153 f.: il.

Thesis (Ph.D.) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2016. Advisor: Philippe Olivier Alexandre Navaux; Advisor during Ph.D. Internship: Jean-François Méhaut.

1. Load Balancing. 2. DVFS. 3. Energy Efficiency. I. Navaux, Philippe Olivier Alexandre. II. Méhaut, Jean-François. III. Título.

“Whatever you are, be a good one”

SIR A. LINCOLN

ACKNOWLEDGMENTS

First, I would like to thank my advisor Philippe Olivier Alexandre Navaux for having accepted me as his PhD. student. For all the ideas and motivation during this PhD period. Thank, also by opportunities, meetings, conferences and projects and mainly by encouraging the internship in France where I had the opportunity to work in LIG team. I also would like to thank sandwich-advisor Jean-François Méhaut, for all suggestions and by scientific support on our work research.

The CAPES-Brazil through PDSE program that sponsored my internship abroad and French-Brazilian HPC-GA project that provided opportunity for travelling attending meetings and conferences.

I would like to all friends and colleagues members of the GPPD and LIG research teams by their support and help during the PhD. A main and special thanks to all of my family by comprehension and patience in my absence: thank you for understanding and supporting me.

ABSTRACT

Current HPC systems have made more complex simulations feasible, yielding benefits to several research areas. To meet the increasing processing demands of these simulations, new equipment is being designed, aiming at the exaflops scale. A major challenge for building these systems is the power that they will require, which current perspectives reach the GigaWatts. To address this problem, this thesis presents an approach to increase the energy efficiency using of HPC resources, aiming to reduce the effects of load imbalance to save energy.

We developed an energy-aware strategy, called ENERGYLB, which considers platform characteristics, and the load irregularity and dynamicity of the applications to improve the energy efficiency. Our strategy takes into account the current computational load and clock frequency, to decide whether to call a load balancing strategy that reduces load imbalance by migrating tasks, or use Dynamic Voltage and Frequency Scaling (DVFS) technique to adjust the clock frequencies of the cores according to their weighted loads. As different processor architectures can feature two levels of DVFS granularity, per-chip DVFS or per-core DVFS, we created two different algorithms for our strategy. The first one, FG-ENERGYLB, allows a fine control of the clock frequency of cores in systems that have few tens of cores and feature per-core DVFS control. On the other hand, CG-ENERGYLB is suitable for HPC platforms composed of several multicore processors that do not allow such a fine-grained control, i.e., that only perform per-chip DVFS. Both approaches exploit residual imbalances on iterative applications and combine dynamic load balancing with DVFS techniques. Thus, they reduce the clock frequency of underloaded computing cores, which experience some residual imbalance even after tasks are remapped.

We evaluate the applicability of our approaches using the CHARM++ parallel programming system over benchmarks and real world applications. Experimental results present improvements in energy consumption and power demand over state-of-the-art algorithms. The energy savings with ENERGYLB used alone were up to 25% with our FG-ENERGYLB algorithm, and up to 27% with our CG-ENERGYLB algorithm. Nevertheless, residual imbalances were still present after tasks were remapped. In this case, when our approaches were employed together with these load balancers, an improvement in energy savings of up to 56% is achieved with FG-ENERGYLB and up to 36% with CG-ENERGYLB. These savings were obtained by exploiting residual imbalances on iterative applications.

By combining dynamic load balancing with the DVFS technique, our approach is able to reduce the average power demand of parallel systems, reduce the task migration among the available resources, and keep load balancing overheads low.

Keywords: Load Balancing. DVFS. Energy Efficiency.

Abordagens de Balanceamento de Carga ciente de energia para melhorar a eficiência energética em sistemas HPC

RESUMO

Os atuais sistemas de HPC tem realizado simulações mais complexas possíveis, produzindo benefícios para diversas áreas de pesquisa. Para atender à crescente demanda de processamento dessas simulações, novos equipamentos estão sendo projetados, visando à escala exaflops. Um grande desafio para a construção destes sistemas é a potência que eles vão demandar, onde perspectivas atuais alcançam GigaWatts. Para resolver este problema, esta tese apresenta uma abordagem para aumentar a eficiência energética usando recursos de HPC, objetivando reduzir os efeitos do desequilíbrio de carga e economizar energia. Nós desenvolvemos uma estratégia baseada no consumo de energia, chamada ENERGYLB, que considera características da plataforma, irregularidade e dinamicidade de carga das aplicações para melhorar a eficiência energética. Nossa estratégia leva em conta carga computacional atual e a frequência de clock dos cores, para decidir entre chamar uma estratégia de balanceamento de carga que reduz o desequilíbrio de carga migrando tarefas, ou usar técnicas de DVFS par ajustar as frequências de clock dos cores de acordo com suas cargas computacionais ponderadas. Como as diferentes arquiteturas de processador podem apresentam dois níveis de granularidade de DVFS, DVFS-por-chip ou DVFS-por-core, nós criamos dois diferentes algoritmos para a nossa estratégia. O primeiro, FG-ENERGYLB, permite um controle fino da frequência dos cores em sistemas que possuem algumas dezenas de cores e implementam DVFS-por-core. Por outro lado, CG-ENERGYLB é adequado para plataformas de HPC composto de vários processadores multicore que não permitem tal refinado controle, ou seja, que só executam DVFS-por-chip. Ambas as abordagens exploram desbalanceamentos residuais em aplicações interativas e combinam balanceamento de carga dinâmico com técnicas de DVFS. Assim, eles reduzem a frequência de clock dos cores com menor carga computacional os quais apresentam algum desequilíbrio residual mesmo após as tarefas serem remapeadas. Nós avaliamos a aplicabilidade das nossas abordagens utilizando o ambiente de programação paralela CHARM++ sobre benchmarks e aplicações reais. Resultados experimentais presentaram melhorias no consumo de energia e na demanda potência sobre algoritmos do estado-da-arte. A economia de energia com ENERGYLB usado sozinho foi de até 25% com nosso algoritmo FG-ENERGYLB, e de até 27% com nosso algoritmo

CG-ENERGYLB. No entanto, os desequilíbrios residuais ainda estavam presentes após as serem tarefas remapeadas. Neste caso, quando as nossas abordagens foram empregadas em conjunto com outros balanceadores de carga, uma melhoria na economia de energia de até 56% é obtida com FG-ENERGYLB e de até 36% com CG-ENERGYLB. Estas economias foram obtidas através da exploração do desbalanceamento residual em aplicações interativas. Combinando balanceamento de carga dinâmico com DVFS nossa estratégia é capaz de reduzir a demanda de potência média dos sistemas paralelos, reduzir a migração de tarefas entre os recursos disponíveis, e manter o custo de balanceamento de carga baixo.

Palavras-chave: Balanceamento de Carga, DVFS, Eficiência Energética.

LIST OF ABBREVIATIONS AND ACRONYMS

AMPI *Adaptive Message Passing Interface*

API *Application Programming Interface*

APM *Advanced Power Management*

ARM *Advanced RISC Machine*

BMC *Baseboard Management Controller*

BRAMS *Brazilian developments on the Regional Atmospheric Modeling System*

BRGM *Bureau de Recherches Géologiques et Minières*

CFD *Computational Fluid Dynamics*

ChaNGa *Charm N-body GrAvity*

DARPA *Defense Advanced Research Projects Agency*

DVFS *Dynamic Voltage and Frequency Scaling*

EDP *Energy-Delay Product*

FEM *Finite Element Method*

FLOPS *Floating-Point Operation per Second*

FVM *Finite Volume Methods*

GPU *Graphics Processing Unit*

HPC *High Performance Computing*

HPCGA *High Performance Computing for Geophysics Applications*

HPL *High Performance Linpack*

IPMI *Intelligent Platform Management Interface*

JLESC *Joint Laboratory for Exascale Computing*

LICIA *Laboratoire International en Calcul Intensif et Informatique Ambiante*

LIG *Laboratoire d'Informatique de Grenoble*

MSR *Model-Specific Registers*

NAMD *NAnoscale Molecular Dynamics*

NUMA *Non-Uniform Memory Access*

PAPI *Performance API*

PCU *Power Control Unit*

PDP *Power-Delay Product*

PPL *Parallel Programming Laboratory*

PSU *Power Supply Unit*

RAMS *Regional Atmospheric Modeling System*

RAPL *Running Average Power Limit*

RISC *Reduced Instruction Set Computer*

RTS *RunTime System*

TCO *Total Cost of Ownership*

LIST OF FIGURES

Figure 1.1 Instantaneous power demand in different scenarios. 20
Figure 1.2 Accumulated processor occupation measured during execution of the application using GREEDYLB load balancer with different processors counts. 20

Figure 2.1 Application regularity. 27
Figure 2.2 Application dynamicity. 29

Figure 4.1 Overview of the fine-grained algorithm. 63
Figure 4.2 Overview of the coarse-grained algorithm. 67
Figure 4.3 ENERGYLB implementation. 70

Figure 5.1 Evaluation of FG-ENERGYLB with *lb_test* benchmark. 81
Figure 5.2 Instantaneous power measured during execution of the imbalanced *lb_test* benchmark using different load balancers. 83
Figure 5.3 Load balancing overhead and total task migration to *lb_test* with different load balancers. 86
Figure 5.4 Evaluation of FG-ENERGYLB with *ComprehensiveBench* benchmark. 88
Figure 5.5 Load imbalance relation between processors at initial and final execution. 89
Figure 5.6 Total energy consumption of *ComprehensiveBench* in each processor. 90
Figure 5.7 Evaluation of FG-ENERGYLB with *Ondes3D* application. 92
Figure 5.8 Comparison of Instantaneous Relative Load with Instantaneous Clock Frequency for *Ondes3D* when used FG-ENERGYLB alone. 93
Figure 5.9 Load balancing overhead and Total task migration to *Ondes3D* with different load balancers. 93
Figure 5.10 Evaluation of FG-ENERGYLB with *Lulesh* application. 94
Figure 5.11 Comparison of Instantaneous Relative Load with Instantaneous Clock Frequency for *Lulesh* when used FG-ENERGYLB alone. 95
Figure 5.12 Load balancing overhead and Total task migration to *Lulesh* with different load balancers. 96
Figure 5.13 Energy spend with load balancing over total energy consumption for each real application. 97
Figure 5.14 FG-ENERGYLB comparison with different threshold value on *Ondes3D*. 100
Figure 5.15 FG-ENERGYLB comparison with different threshold value on *Lulesh*. 101
Figure 5.16 Evaluation of CG-ENERGYLB with *Ondes3D* application. 104
Figure 5.17 Instantaneous power demand and clock frequency measured during the execution of *Ondes3D* with CG-ENERGYLB. 105
Figure 5.18 Average load balancing overhead to *Ondes3D* application. 107
Figure 5.19 Evaluation of CG-ENERGYLB with *Lulesh* application. 109
Figure 5.20 Average load balancing overhead to *Lulesh* application. 110
Figure 5.21 Energy spend with load balancing over total energy consumption for each real application. 111
Figure 5.22 CG-ENERGYLB comparison with different threshold value on *Ondes3D*. 112
Figure 5.23 Power evaluation to different threshold value on *Ondes3D* 113
Figure 5.24 CG-ENERGYLB comparison with different threshold value on *Lulesh*. 114
Figure 5.25 Power evaluation to different threshold value on *Lulesh* 115

Figure A.1 Evaluation of FG-ENERGYLB with *kNeighbor* benchmark. 137
Figure A.2 Load balancing overhead and Total task migration to *kNeighbor* load balancer with different load balancers. 138

Figure A.3 Evaluation of FG-ENERGYLB with *stencil 4D* benchmark. 140
Figure A.4 Load balancing overhead and Total task migration to *stencil 4D* load balancer with different load balancers. 141
Figure A.5 Evaluation of FG-ENERGYLB with *Lassen* application. 144
Figure A.6 Comparison of Instantaneous Relative Load with Instantaneous Clock Frequency for *Lassen* when used FG-ENERGYLB alone. 145
Figure A.7 Load balancing overhead and Total task migration to *Lassen* with different load balancers. 146
Figure A.8 Energy spend with load balancing over total energy consumption for each real application. 147
Figure A.9 FG-ENERGYLB comparison with different threshold value on *Lassen*. ... 148

Figure B.1 Evaluation of CG-ENERGYLB with *Lassen* application. 149
Figure B.2 Average load balancing overhead to *Lassen* application. 150
Figure B.3 Energy spend with load balancing over total energy consumption for each real application. 152
Figure B.4 CG-ENERGYLB comparison with different threshold value on *Lassen*. ... 153
Figure B.5 Power evaluation to different threshold value on *Lassen* 153

LIST OF TABLES

Table 2.1 Examples of scientific application and dynamicity. 32
Table 2.2 Strategies characteristics. 43
Table 2.3 Load balancing algorithms comparison in terms of make decision criteria. ... 44

Table 3.1 DVFS strategies comparison in terms of applicability. 59

Table 4.1 Description of variables used in algorithms. 63
Table 4.2 Comparison of the proposed algorithms with the state of the art in terms of decision criteria. 74

Table 5.1 Benchmarks and real world applications characteristics. 78
Table 5.2 Summary of the input parameters of benchmarks and real applications. 79
Table 5.3 List of load balancers abbreviations used in the presentation of results. 80
Table 5.4 Total execution time, energy consumption and average power demand measured to *lb_test* benchmark. 84
Table 5.5 Average load balancing duration in seconds for each real application. 97
Table 5.6 Summary of the input parameters of real applications. 103
Table 5.7 Total energy consumption and improvements for CG-ENERGYLB and other load balancers. 106
Table 5.8 Average load balancing duration in seconds for each real application. 111

Table A.1 Average load balancing duration in milliseconds for each benchmark. 142
Table A.2 Percentage of energy spend with load balancing over the total energy consumption for each benchmark. 143
Table A.3 Average load balancing duration in seconds for each real application. 146
Table A.4 Percentage of energy spend with load balancing over the total energy consumption for each real application. 147

Table B.1 Average load balancing duration in seconds for each real application. 151
Table B.2 Percentage of energy spend with load balancing over the total energy consumption for each real application. 152

CONTENTS

1 INTRODUCTION 16
1.1 Problem Statement and Motivation 18
1.2 Objectives and Contributions 21
1.3 Scientific Context 22
1.4 Text Organization 23
2 LOAD BALANCING OVERVIEW 25
2.1 Characterization of Computational Load 25
2.1.1 Scientific Applications 26
2.1.1.1 Regular and Irregular Applications 27
2.1.1.2 Static and Dynamic Applications 28
2.1.1.3 Iterative and Non-Iterative Applications 28
2.1.2 Scientific Application Classes 29
2.2 Load Balancing Heuristics 32
2.2.1 Load Balancing Decisions 33
2.3 Related Work on Load Balancing 34
2.3.1 Centralized Strategies 35
2.3.1.1 Basic Centralized Load Balancing Strategies 36
2.3.1.2 Centralized Load Balancing Strategies using Communication Data 37
2.3.1.3 Centralized Topology-Aware Load Balancing Strategies 37
2.3.1.4 Centralized Temperature-Aware Load Balancing Strategies 38
2.3.1.5 Centralized Energy-Aware Load Balancing Strategies 39
2.3.2 Distributed Strategies 40
2.3.2.1 Basic Distributed Load Balancing Strategies 41
2.3.3 Hierarchical or Multi-Level Strategies 41
2.3.3.1 Basic Hierarchical Load Balancing Strategies 42
2.4 Discussion 43
3 ENERGY CONSUMPTION OVERVIEW 46
3.1 Energy Consumption Metrics 47
3.1.1 Power Demand 47
3.1.2 Energy Consumption 48
3.2 Energy Consumption Evaluation 49
3.2.1 Power and Energy Measurement 49
3.2.1.1 External Measurement 49
3.2.1.2 Integrated Measurement 50
3.3 Related Work on Frequency Scaling 52
3.3.1 Frequency Scaling on Idle Resources 54
3.3.2 Frequency Scaling on Power or Energy Budget 55
3.3.3 Frequency Scaling on Communication Periods 55
3.3.4 Frequency Scaling on Kernel Governors 56
3.3.5 Frequency Scaling on Performance Prediction 57
3.4 Discussion 58
4 PROPOSED ENERGY-AWARE LOAD BALANCERS 61
4.1 EnergyLB: Energy-Aware Load Balancing 62
4.1.1 FG-EnergyLB: Fine-Grained EnergyLB 63
4.1.1.1 FG-EnergyLB Algorithm 64
4.1.1.2 FG-EnergyLB Algorithm Properties 65
4.1.2 CG-EnergyLB: Coarse-Grained EnergyLB 66
4.1.2.1 CG-EnergyLB Algorithm 66

4.1.2.2 CG-ENERGYLB Algorithm Properties 68
4.2 Implementation Details 68
4.2.1 CHARM++ Parallel Programming 69
4.2.2 Energy Modules 70
4.2.3 ENERGYLB Phases 71
4.3 Discussion 72
5 LOAD BALANCERS EVALUATION 75
5.1 Evaluation Methodology 75
5.1.1 Experimental Environment 75
5.1.2 Benchmarks and Irregular Applications 76
5.1.3 Load Balancers 78
5.1.4 Power and Energy Monitoring 78
5.1.5 Test Execution Details 79
5.2 FG-ENERGYLB Evaluation 80
5.2.1 Evaluation on Benchmarks 81
5.2.2 Evaluation on Real Applications 91
5.2.2.1 Percentage of Energy Spent on Load Balancing 96
5.2.2.2 Threshold Evaluation 98
5.3 CG-ENERGYLB Evaluation 102
5.3.1 Evaluation on Real Applications 103
5.3.1.1 Percentage of Energy Spent on Load Balancing 110
5.3.1.2 Threshold Evaluation 112
5.4 Overall Results 116
6 CONCLUSIONS AND FUTURE WORK 119
6.1 Contributions 119
6.2 Perspectives of Future Work 121
6.3 Published Papers 122
REFERENCES 125
APPENDIX A — FG-ENERGYLB EVALUATION 137
A.1 Evaluation on Benchmarks 137
A.1.1 Percentage of Energy Spent on Load Balancing 141
A.2 Evaluation on Real Applications 144
APPENDIX B — CG-ENERGYLB EVALUATION 149
B.1 Evaluation on Real Applications 149

1 INTRODUCTION

Parallel scientific applications have been influencing the way science is done for decades. These applications have ever-increasing demands in terms of performance and resources, due to their great complexity and large data-sets. In order to meet these demands, the performance of High Performance Computing (HPC) platforms has been growing exponentially for years (LAROS et al., 2009; DONG; CHEN; TANG, 2010).

Current PFlops (10^{15} floating point operation per second) systems allow reaching increasingly accurate results for several scientific applications, such as climate modeling, oil exploration and atomic simulation. However, this exponential increase in computational performance also leads to an exponential growth in power demand, i.e., the rate of power (watt) consumed directly of a power supply (HSU; FENG, 2005; LAROS et al., 2009; DONG; CHEN; TANG, 2010; ALVES et al., 2010; PADOIN et al., 2013a).

The main focus of HPC systems has been performance, and today the HPC community works toward building Exascale systems (i.e., EFlops), which will provide unprecedented computational power, allowing to solve even larger scientific problems. Conceiving Exascale supercomputers by scaling the current technology, these systems could demand over a GigaWatt of power (HSU; FENG; ARCHULETA, 2005; FENG; CAMERON, 2007), which is equivalent to the entire production of a medium size nuclear power plant (WEHNER; OLIKER; SHALF, 2009). In this way, a global research effort has risen to try to break this barrier while avoiding such high power demands, since the electric company must provide to customers this power when they turn the equipment on.

In recent decades, supercomputers were compared almost exclusively by their computing performance, also defined by runtime or total execution time, as in the context of this thesis. For instance, the Top500 list was established to rank supercomputers regarding processing speed (DONGARRA; MEUER; STROHMAIER, 2014). Nonetheless, given the exponential growth in power demand of HPC systems (LAROS et al., 2009; DONG; CHEN; TANG, 2010), and responding to the energy efficiency problem, the HPC community started new initiatives that take into account both performance and power demand (REN; SUDA, 2010). An example is the Green500 list. It considers the ratio between performance and power (Flops/W) to define the energy efficiency of parallel machines (GE et al., 2007; FENG; LIN, 2010; SUBRAMANIAM; FENG, 2010; CAMERON, 2010; SCOGLAND; SUBRAMANIAM; FENG, 2012).

In this context, the first challenge to the development of energy efficient HPC

systems and applications lies on increasing their performance while reducing their power demand. Indeed, reducing power demand and saving energy have become one of the main concerns of the HPC community. Thus, to build future systems we need to take into account power demand and energy consumption constraints and ways to improve energy efficiency (BARKER et al., 2009; YOUNGE et al., 2010; PADOIN et al., 2013b; PADOIN et al., 2014). In this context, current processors have incorporated solutions to improve energy efficiency. Some processor models have temperature sensors to avoid overheating and reduce cooling costs. Others also feature mechanisms that allow controlling voltage through Dynamic Voltage and Frequency Scaling (DVFS). By using DVFS, the processor's clock frequency can be reduced and, consequently, reduces its instantaneous power demand and the total energy consumption. However, this reduction in processor's clock frequency also causes a reduction on application performance.

To attend to the increasing demand for processing, a second challenge arises when HPC systems grow in number of processors. This complicates the efficient use of all resources at the hardware level. To run scientific applications on these complex parallel systems, the application's work (or computing load) is divided into tasks or processes. However, as the computational load of a scientific applications may not be equally divided due to dynamic or irregular characteristics, the application may be executed and end up with an imbalanced load. These characteristics are present in several scientific applications and contributes to a reduced energy efficiency on parallel systems, since the most heavily loaded processor determines the application's performance. Several load balancing strategies have been used to improve the load distribution across processors and to achieve an efficient use of all available resources of a parallel machine. However, these strategies work with a NP-Hard problem, doing exhaustive searches and can take processing time and may even degrade the performance if they are not well applied (LEUNG, 2004; BRUCKER; BRUCKER, 2007).

Therefore, the challenge of designing more energy efficient applications is to improve the use of all hardware resources available in the parallel systems, providing near optimal performance for applications and at the same time improve the energy efficiency of the systems. This pursuit is also motivated by the current energy cost of HPC systems. Some supercomputers demand more than 10 MW of power, which costs several million dollars per year in energy, i.e., the power demand integrated over a specified time period. Beyond that, it generates emissions of millions metric tons of carbon dioxide. Furthermore, the total cost of equipment maintenance, will overcome in a few years the cost

of the hardware infrastructure acquisition (TORRELLAS, 2009; STEIGERWALD et al., 2011; PERAZA et al., 2013; RAJOVIC et al., 2014).

In this context, state-of-the-art research focuses on either power demand or load balancing strategies separately. Several proposal have used DVFS, however this technique may cause performance degradation and an increase of the total execution time of parallel applications. Other proposals have used load balancing strategies to reduce the overall execution time, and, consequently, save energy. However, these strategies have used only computational load ignoring power demand or total energy consumption.

In the context of this thesis, we improve the energy efficiency of parallel systems when running load imbalanced applications. We provide new load balancing strategies that collect system information from each node of the parallel machine and collect information from tasks of the application, in order to take decision more accurately and to manage power demand while the imbalanced application is running. Our approach combines dynamic load balancing with the DVFS technique in order to reduce the clock frequency of underloaded cores, which have some residual imbalance.

1.1 Problem Statement and Motivation

Several scientific applications take advantage of the parallelism offered by large HPC systems to simulate different natural phenomena. However, to satisfy their increasing processing demand, we end up increasing the power demand of these systems. This is an issue, since they have reached near of the limit specified by DARPA report (KOGGE et al., 2008).

Taking into account power constraints, issues such as power demand and energy consumption are increasingly discussed in the HPC community. Nowadays, the energy efficiency of parallel systems is a major challenge in scientific research. In this context, **our research problem is to efficiently use all hardware resources available in parallel machines to provide the best performance possible and, at the same time, reduce the power demand to improve the energy efficiency of parallel HPC systems**.

Generally, approaches to reduce the power demand are used separately of research for increasing application's performance. Several approaches have been applied to mitigate the imbalanced workloads and achieve an efficient use of all parallel resources. However, most of these approaches focus on reducing the execution time of the applications by improving the load distribution, but neglecting power consumption. Other approaches

have used power controls of current microarchitectures to change the voltage level, reducing then their operating frequency and saving power, without taking into account the load distribution.

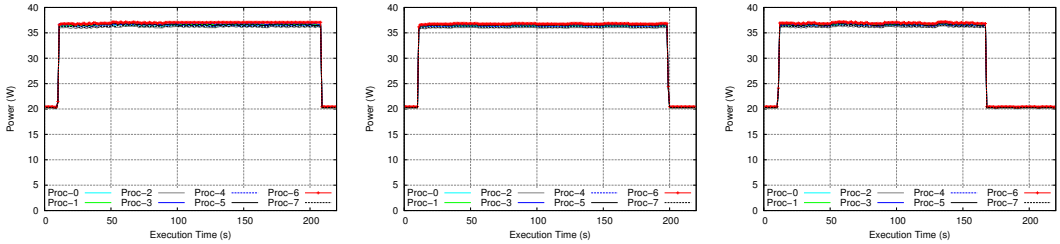
Load balancing strategies exploit the load imbalances of tasks and achieve reductions in the total execution time. These reductions are relevant in the perspective of energy consumption, since energy is saved when hardware resources are used for a shorter time. However, some imbalance may still remain after making decisions and task migrations. In this thesis, we call this remaining load imbalance left by the load balancers of *residual imbalance*. In this context, it is possible to achieve even greater energy savings if the runtime system is able to exploit the residual imbalances to fine tune the voltage and frequency of cores accordingly. In this case, the challenge lies in reducing the energy consumption of the application while maintaining its performance.

In this context, iterative imbalanced applications are potential candidates for energy consumption improvements. These parallel applications are present on different scientific fields. Molecular dynamics, structural dynamics, weather forecast (MICHALAKES; VACHHARAJANI, 2008), cosmological modeling simulation (DIKAIAKOS; STADEL, 1996; JETLEY et al., 2008), seismic wave propagation simulations (DUPROS et al., 2008), physics (SHIERS, 2007), and oil exploration (PANETTA et al., 2009), are examples of applications that can benefit from load balancing schemes to improve performance. Our proposal is that these schemes still leave space for energy optimization.

In an experimental evaluation, tests were performed using 8 and 24 cores (one on each processor) of the parallel machine presented in Section 5.1. We ran a small test with *Ondes3D* (DUPROS et al., 2008), a seismic wave simulation that uses the CHARM++ parallel programming framework (KALÉ; KRISHNAN, 1993; CHARM++, 2014). Figures 1.1 presents the power demand measured on 8 processors during the execution of the iterative application in different scenarios. In the first scenario (Figure 1.1(a)), the application executes a balanced workload, without any load balancer. In the second scenario (Figure 1.1(b)), the same workload is executed with GREEDYLB (KALÉ; KRISHNAN, 1993), a load balancer that uses a greedy approach further discussed in Section 5.1.3. As it can be observed, GREEDYLB had almost no effect on both execution time and power demand, since the workload is originally balanced.

In the third scenario the application executes an imbalanced workload (Figure 1.1(c)). In this case, GREEDYLB was able to improve the performance of the application, reducing its execution time by 22% when compared to the same workload without a load

Figure 1.1: Instantaneous power demand in different scenarios.



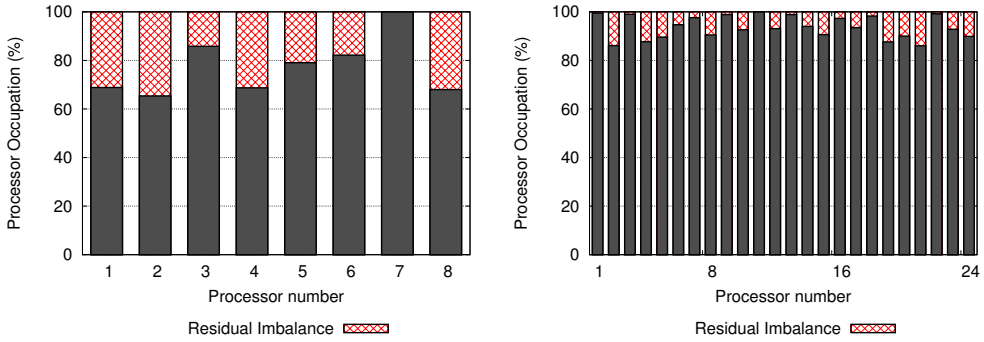
(a) Balanced application without load balancer. (b) Balanced application with GREEDYLB. (c) Imbalanced application with GREEDYLB.

Source: The author

balancer. We also observe that even though the power demand remained the same. It's important to observe that the application consumed less energy in this case, due to its shorter execution time.

Load balancers improve the performance of imbalanced applications by making a better load distribution among the available processors. However, they can take sub optimal decisions, which may result in some load remaining imbalanced after task migrations. This may happen due to characteristics of the application that prevent a perfectly balanced mapping to be achieved, or due to limitations of the load balancing heuristics, as the problem that they are trying to solve is NP-Hard (LEUNG, 2004).

Figure 1.2: Accumulated processor occupation measured during execution of the application using GREEDYLB load balancer with different processors counts.



(a) 8 processors. (b) 24 processors.

Source: The author

The accumulated occupation (usage percent) of each processor used during the execution of the application with GREEDYLB is shown in the Figure 1.2. When running the application in 8 processors a residual imbalance of up to 34% is achieved (Figure 1.2(a)). Using the same workload to each task and running the application in 24 processors, (Figure 1.2(b)), a different load distribution is achieved, where a residual imbalance of up to 14% is present. We notice that this residual imbalance is considerably high in several scenarios, for example, in irregular or dynamic applications, in which the processors are

underused. In scenarios like this, DVFS techniques can be used to reduce its frequency and, consequently its power demand to save energy.

Based on **the state-of-the-art research that focuses on either increasing the application's performance or reducing its power demand, we make the case that both strategies, dynamic load balancing and DVFS, could be used together. Our hypothesis to achieve a better energy efficiency in parallel systems when running load imbalanced applications is that precise tasks and machine information can help load balancing algorithms in their decisions, in order to manage power demand considering residual imbalance while the application is running.**

1.2 Objectives and Contributions

The objective of this thesis is **to enhance the global load balancing distribution of applications, reduce the idleness and increase the energy efficiency of HPC systems using information about the load and power demand of processors.** We intend to unite two areas, dynamic load balancing and DVFS, following the hypothesis that system information, for example, processor occupation or residual imbalance of the parallel platform and load information of imbalanced applications can be used to improve the efficacy of load balancing strategies. Considering this objective, our main contributions are as follows:

- We introduce a new energy-aware load balancing algorithm called **ENERGYLB**, to save energy on iterative applications that present imbalanced loads. Our strategies can reduce the power demand during execution, saving energy and improving the application performance according to its load characteristic. We have implemented two algorithms for our proposed ENERGYLB, using the existing CHARM++ framework, which implements a model of migratable object. The first one, called *Fine-Grained EnergyLB (*FG-ENERGYLB*)*, is suitable for platforms composed of few tens of cores and that allow per core DVFS. The second one, *Coarse-Grained EnergyLB (*CG-ENERGYLB*)*, is suitable for platforms composed of several processors and that do not allow per core DVFS. Our approaches can be applied alone or attached to other load balancers to save energy. When it is used attached, in a first step, a load balancer runs its algorithm to achieve a better load distribution. After

that, ENERGYLB computes the residual imbalance and adjusts the clock frequency according to residual load imbalance, to save energy (PADOIN et al., 2014).

- We evaluate our proposed approaches and demonstrate the improvements in performance and energy savings. For this purpose, we run a set of benchmarks and real world applications. The results were compared to other state-of-the-art algorithms and show a reduction in power demand and in the amount of the energy saving incurring in very low overhead (PADOIN et al., 2014).
- We proposed and developed a tool named **EMonDaemon** to collect power and energy consumption on homogeneous and heterogeneous systems. This contribution is motivated due to current platforms having different interfaces to collect power and energy data from different equipment or tools. In some of these tools, the collected data can be analyzed only after the execution of the application. This makes it difficult to correlate performance with power and energy consumption. Another difficulty is that some equipment have sensors that enable the measurement only of the processor with a low granularity, while others allow only the measurement of entire machine. For example, recent Intel Sandy-Bridge-EP processors have MSR registers that enable the measurement of the accumulated energy of each processor, core, RAM and/or GPU. On the other hand, MPSoCs like Odroid, feature processors and have sensors to measure the instantaneous power of each cluster. These power sensors, not present in older systems, make it possible to analyze energy consumption with a fine granularity. Our solution can be applied to measure performance, instantaneous power and energy consumption on heterogeneous systems with Intel and ARM big.LITTLE processors, correlating performance with power and energy consumption during runtime (PADOIN et al., 2015).

1.3 Scientific Context

This research is being developed in the Institute of Informatics of the *Universidade Federal do Rio Grande do Sul* (UFRGS) in cooperation with the *Laboratoire d'Informatique de Grenoble* (LIG) of the *Université Grenoble Alpes*. In the context of this thesis, the work has been conducted within two research groups. In UFRGS, the research has been developed in the Parallel and Distributed Processing Group (GPPD), and in LIG, during internship period, the research was conducted in the Nanosimulations and

Embedded Applications for Hybrid Multicore Architectures (NANOSIM) group.

As result of the cooperation and collaboration between the two research groups, an associate laboratory called of *Laboratoire International en Calcul Intensif et Informatique Ambiante* (LICIA) was created. Besides that, there was some cooperation of the two groups as part of the *High Performance Computing for Geophysics Applications* (HPC-GA) project. So, our research has benefited from the experience of researchers of the two groups and their common research areas, more specifically, dynamic load balancing algorithms, task migration, dynamic process control and scientific real world applications.

On dynamic load balancing algorithms, Pilla (PILLA et al., 2012; PILLA et al., 2014) developed load balancing algorithms that work at runtime level and combine application information about the machine topology gathered during execution. This approach improves the distribution of tasks over a parallel platform in order to mitigate load imbalance and costly communications.

Rodrigues (RODRIGUES et al., 2009; RODRIGUES et al., 2010) proposed a load balancing approach to reduce the execution time of applications by reducing the load imbalance and costly communications in a real world application, BRAMS. In same context, Tesser (TESSER et al., 2014b) evaluated the use of dynamic load balancing in CHARM++ to improve the performance of *Ondes3D*, a seismic wave propagation simulator developed by BRGM. *Ondes3D* was ported to AMPI and a performance evaluation was done using four load balancers available on CHARM++ and two topology aware load balancers, the NUCOLB (PILLA et al., 2012) and HWTOPOLB (PILLA et al., 2014), both strategies developed by Laércio L. Pilla.

Our work also has a collaboration with the Parallel Programming Laboratory (PPL) of the University of Illinois at Urbana-Champaign (UIUC) within the Joint Laboratory for Exascale Computing (JLESC) (PADOIN et al., 2014b) and (PADOIN et al., 2014a). This collaboration is very important to our research since we used CHARM++, the runtime system developed by PPL, to implement and test our proposed energy-aware load balancing algorithms. Besides that, the results were presented in 12th Workshop on Charm++ and its Applications (PADOIN et al., 2014c).

1.4 Text Organization

The remaining chapters of this thesis are structured as follows:

- **Chapter 2** reviews the basic concepts about load balancing strategies, which are relevant to this thesis. We discuss how the load balancing impacts the use the resources available in parallel machines, since it can interfere in the performance of parallel applications and on the energy consumption of entire system. We detail the behavior of parallel applications, characterize load balancing, and describe the most known strategies for load balancing. We also evaluate scientific applications in the context of load balancing;
- **Chapter 3** presents a review of basic concepts and metrics to measure energy consumption, showing the quantities involved in the measurement that are relevant to the development of this thesis. In this context, we provide a discussion about energy evaluation. We also present a review of the state of the art, contextualizing the addressed issue, research and technological advances related to energy consumption;
- **Chapter 4** proposes the Energy-Aware Load Balancer (ENERGYLB) to improve the energy efficiency of parallel systems when running load imbalanced applications. In this chapter, we described in detail our two proposed load balancing strategies, divided into motivation and contribution. We also discuss algorithms and implementation details. Our approaches propose combining dynamic load balancing and DVFS considering the characteristics of the application and of the system to make decisions;
- **Chapter 5** presents an experimental evaluation of our Energy-Aware Load Balancing approach. In this chapter, we discuss the methodology of the validation of our proposed load balancers. We present the parallel system, the benchmarks, the real world applications, and other available load balancers. These were used in a set of tests to measure the runtime, power demand and energy consumption improvements and compare them with our proposed strategies;
- **Chapter 6** summarizes our conclusions and contributions as well as outlining perspectives for the future work;
- **Appendix A** presents an experimental evaluation of our Centralized Energy-Aware Load Balancer using benchmarks and real world applications; and
- **Appendix B** presents an experimental evaluation of our Hierarchical Energy-Aware Load Balancer using benchmarks and real world applications.

2 LOAD BALANCING OVERVIEW

Applications has been distributed across all system processors aiming achieve high performance and attend the computational demands of scientific simulations. Since several applications suffer from load imbalance, which can reduce performance and scalability, the workload is partitioned according to specific load balancing strategies aiming efficiently use all parallel resources of HPC systems. Load imbalance also contributes to the inefficiency of parallel applications, since the runtime of an application is defined by the most loaded processor. Therefore, achieving a balanced load is one of the main issues to reach high performance, especially when dynamic and irregular problems are computed.

Load balancing (LB) strategies are techniques widely used to achieve a better distribution of tasks or computational load across processors of a parallel machine. They aim to reduce the load imbalance, avoid overloaded processors and decrease the contention in communication resources. Load balancing has been studied extensively, since it is present in several applications from different scientific fields. Therefore, scientific applications are potential candidates for performance improvements through load balancing. Several approaches and combinatorial algorithms that have been proposed to reduce the execution time focusing mainly in increasing resources usage at the hardware level.

In this chapter, we present the state of the art on load balancing strategies for scientific applications. We review the previous fundamental results related to the topic of this thesis. In the first section, we detail the main characteristics of computational load and scientific applications. Load balancing heuristics and load balancing categories are presented in Sections 2.2 and 2.3, respectively. Finally, we present the concluding remarks of this chapter, highlighting some points regarding the context of this thesis.

2.1 Characterization of Computational Load

Parallel scientific applications have allowed addressing grand challenges in science. Scientific applications such as molecular dynamics simulations, natural phenomena forecast, seismic models and weather forecasting are run in HPC systems aiming at more accurate results and shorter execution times.

Research on the HPC field aims to provide faster supercomputer systems to run even bigger simulations creating a better understanding and predicting natural phenom-

ena. However, these applications are constantly increasing the amount of data to be processed aiming a higher precision, demanding a larger number of processing cores, and consuming large amounts of energy to run. To attend to this demand, supercomputers are scaling their performance exponentially over the years leading to an exponential growth in power demand (DONG; CHEN; TANG, 2010; GERARDS et al., 2014).

Furthermore, several scientific applications are complex and have tasks with different or dynamic computational loads that often result in load imbalance and poor scaling. Therefore, in order to determine if an application can benefit from load balancing strategies, their behavior needs to be analyzed.

In the context of this thesis, the load balancing problem can be defined as follows: given a parallel machine with a set of processors and the set of tasks of a parallel application, each task with its own computational load and communication needs, we want to find the mapping that best equalizes the tasks. An important issue for load balancing strategies is how to obtain the information about computation and communication, which will be used for making load balancing decisions. Another issue is that load balancing can incur in an increase in the overall execution time due the time spent on the load evaluation, decision making and task migration (WATTS; TAYLOR, 1998; CHARM++, 2014; TALLENT; ADHIANTO; MELLOR-CRUMMEY, 2010; PEARCE et al., 2012).

Currently, most load balancing strategies make decisions leaving aside the energy consumption of the computer system. Given the high power demand of current systems, however, the energy costs need to be taken into account during load balancing. In this scenario, there is a demand for load balancing strategies able to better manage and reduce the energy consumption of newer systems. An interesting approach would be to investigate the use of power demand information together with computational loads to guide scheduling decisions in order to increase the energy efficiency of parallel systems.

2.1.1 Scientific Applications

Scientific applications have been used in simulations enabling research in several areas of knowledge. The computational load of large scientific applications are divided into a number of tasks to be processed in parallel by an HPC system. Depending of the programming language used, these tasks may be implemented as processes in MPI, threads in OpenMP, and active objects in CHARM++. However, this may not generate the expected performance gains, due to application characteristics, such as load imbalance

and excessive communication between tasks. To deal with such characteristics, these applications may require special strategies to make efficient use of all available resources.

In this context, load balancers can help to detect imbalance and to migrate tasks in order to redistribute the workload and enhance the use of computing resources.

In several applications, the computational load, also called *load* in this thesis context, cannot be equally divided. In this way, they can be ranked in two groups: Regular and Irregular Applications, as discussed in the subsection following.

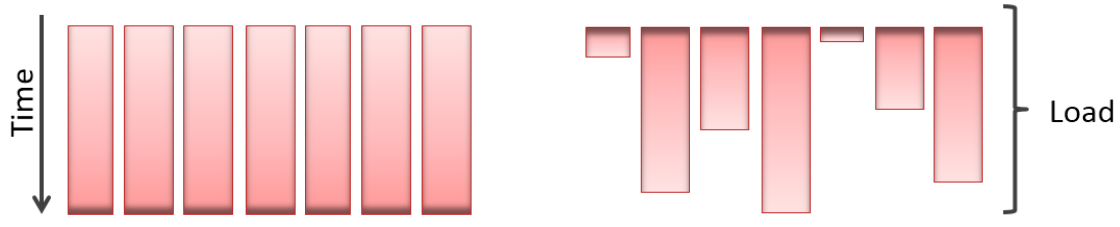
2.1.1.1 Regular and Irregular Applications

Regular applications are those whose tasks have similar execution times in a given application period. During the task creation, their computational load can be equally divided among all tasks. When running this type of application on an HPC system, it is easier to achieve a high processor occupation or resource usage, therefore reaching a higher computational efficiency at the hardware level. For applications with regular loads, static load balancing can easily reach an effective work distribution.

On the other hand, Irregular Applications are those whose computational load cannot be divided in a way that their tasks have similar loads. This is the case of many scientific applications and in addition, for many of these scientific applications, task loads cannot be predicted at start time. This way, it becomes harder to achieve a high processor occupation and an efficient use of the available resources. Furthermore, scaling these applications on current HPC systems is more complex, due to the increased difficulties in scheduling. Examples of irregular applications are state-space search, combinatorial optimization, and recursive parallel codes (LIFFLANDER; KRISHNAMOORTHY; KALÉ, 2012).

Figure 2.1 illustrates an example of regularity present in the computational load during the application's task creation.

Figure 2.1: Application regularity.



(a) Regular Load. (b) Irregular Load.

Source: The author

Other applications have a data-set, which can cause load variations during their execution, making it difficult to predict their behavior. Based on these characteristics, applications can be classified as Static and Dynamic Applications.

2.1.1.2 Static and Dynamic Applications

Some applications have a computational load defined at creation time and it usually remains constant or static throughout the execution. Load balancing algorithms can be applied in these static applications to achieve a better performance using application information gathered during execution time (ICHIKAWA; YAMASHITA, 2000; LEGRAND et al., 2004; MARTÍNEZ et al., 2011; PILLA, 2014).

On the other hand, a significant number of applications have a behavior in which the computational load of tasks varies during execution, making it difficult to predict the load of their tasks. We call them dynamic applications. In this case, we say they have a dynamic load distribution.

Information gathered before execution is not important to load balancing algorithms when they are applied in dynamic applications. Their decisions take into account the current workload of each task periodically, every time a certain number of iterations has elapsed. This behavior is present in a large number of parallel scientific applications, making it even more difficult to run them on heterogeneous or non-dedicated systems (CYBENKO, 1989; HUMMEL et al., 1996; BAHI; CONTASSOT-VIVIER; COUTURIER, 2005; CARIÑO; BANICESCU, 2008; GALINDO; ALMEIDA; BADÍA-CONTELLES, 2008).

Figure 2.2 presents an example of dynamicity present in the computational load during the application's execution.

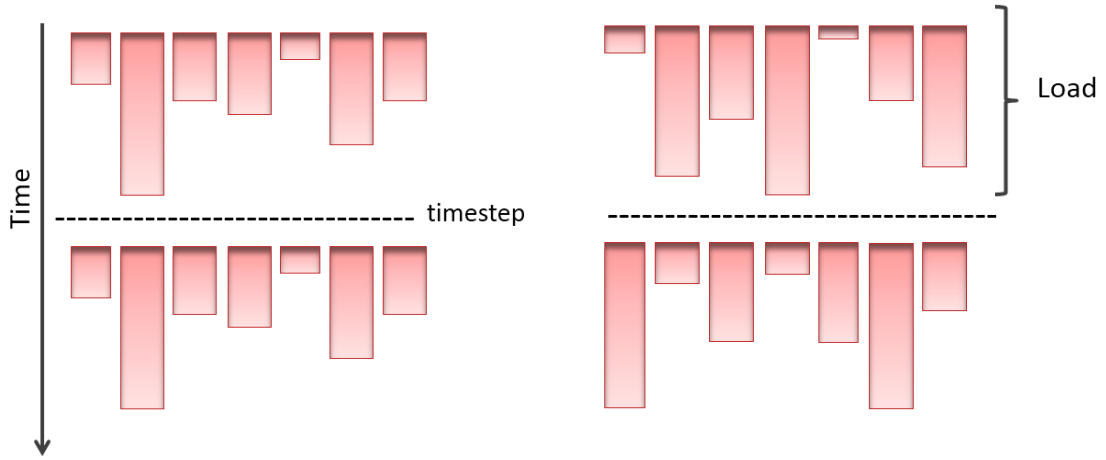
According to their regularity and dynamicity, scientific applications can also be classified as Iterative and Non-Iterative. We will discuss these two in the next subsection.

2.1.1.3 Iterative and Non-Iterative Applications

Iterative applications are those where consecutive iterations have a similar behavior in computation load. That is, these iterations, also known as time steps, have a similar pattern of communication and computation, which can be used to make decisions in load balancing (MENON; KALÉ, 2013).

In iterative applications, the computational load of each task can be evaluated in

Figure 2.2: Application dynamicity.



(a) Static Load. (b) Dynamic Load.

Source: The author

the first iterations and reevaluated every time a certain number of time steps is completed. This allows the current load to be used for predicting the future load. In these strategies, information about load and communication is usually saved in databases to later be used in load balancing decisions.

This kind of application is predominant in real world simulations. For instance, simulations of atoms moving in a structural molecular, NAnoscale Molecular Dynamics (NAMD) (PHILLIPS et al., 2002), climate or weather simulation, cosmological and seismic wave simulation are all examples of iterative applications.

Different from the first type, non-iterative applications do not have any correlation between time steps. This behavior is harder to predict. The difficult increases even more when some of these parallel applications create new tasks in runtime. Thus, creating new computational load and unbalancing the load of the whole system (ZHENG, 2005).

2.1.2 Scientific Application Classes

We selected different classes of scientific applications and examples of applications to verify the regularity and dynamicity of their computational load and the communication. These classes are:

- **Cosmological Simulations**

  The first class of scientific application we analyzed is cosmological simulations. Applications of this class have a unique communication pattern. For example, solvers used in gravitational simulations for the N-body problem generally use

methods such as hybrid codes, particle-mesh methods and tree-methods (BHATELÉ et al., 2011). Examples of cosmological simulations include Parallel K-D tree GRAVity code (PkdGRAV) (DIKAIAKOS; STADEL, 1996) and Charm N-body GrAvity solver (ChaNGa) (JETLEY et al., 2008).

PkdGRAV's code contains a data tree structured where the root-cell of this tree represents the entire simulation volume and ChaNGa's code represents N-body with boundary conditions of isolated stellar systems.

- **Molecular Dynamics**

  The second class of scientific applications we selected is molecular dynamic (MD). These applications focus on the simulation of bio molecular systems involving the computation of forces on a system of N-atoms (BHATELÉ et al., 2011; MEI et al., 2011). Several parallel implementations have been developed for scaling MD codes on current HPC systems by dividing of atoms into cells. So, in each iteration, the force is computed for all pairs of atoms that are within a specified distance, resulting in a computational load proportional to the number of atoms in each cell. These algorithms are considered dynamic due to their load imbalance. This behavior is caused by the variable number of atoms in each cell and by movement of atoms between cells over time.

  LeanMD is a scalable parallel application for MD that simulates the behavior of atoms based on the Lennard-Jones potential (PHILLIPS et al., 2002). The algorithm, implemented using the CHARM++ runtime system, performs a computation similar to the force calculation in NAnoscale Molecular Dynamics (NAMD) (NELSON et al., 1996; BHATELÉ; KALÉ; KUMAR, 2009; MENON; KALÉ, 2013).

- **Seismic Wave Simulations**

  The third class of scientific applications we studied is seismic wave propagation models. Currently, these applications are one of the research topics on HPC due its high demand of processing power. Seismic wave applications can be used to simulate the propagation of waves in a region and predict the consequences of future earthquakes.

  Ondes3D is an example of the seismic wave simulation class. Its algorithm was implemented using several different frameworks, including MPI and CHARM++ (TESSER et al., 2014b). Their main characteristics are the high memory consumption and processing cost. In addition, the data access pattern during the stages of the application that is regular (DUPROS et al., 2008).

- **Unstructured Grid**

  Unstructured grid implementations have been one of the most efficient ways to solve large computational fluid dynamics (CFD) problems and are the fourth class we analyzed. Grid partitioning is a widely used method to decompose computational problems into parallel tasks. It is used mainly in applications where the computational load of the grid or the grid organization changes during the application's execution. The ability of dynamically organize the grid is attractive for increasing the usage of the available resource and achieving higher performance. In this case, dynamic load balancing techniques can be applied to partitioning of unstructured grids, aiming at improving the computational load balanced (BHATELÉ et al., 2011).

  Finite Volume Methods (FVM) and Finite Element Methods (FEM) have been commonly employed to solve problems in this class. However, the cost of their data structures results in more complex implementation (HIRSCH, 2007).

- **Weather Forecasting**

  Weather forecasting is another scientific applications class we analyzed. These applications are employed to predict the state of the atmosphere in a location at a specific time (MICHALAKES; VACHHARAJANI, 2008). Models such as the Regional Atmospheric Modeling System (RAMS) (WALKO et al., 2000), and its Brazilian variant, (BRAMS, 2011; RODRIGUES et al., 2010), are implemented by dividing the globe and the atmosphere into a three-dimensional mesh.

  The computational load of these models is considered irregular with load imbalance. This happens because tasks may present different workloads depending on input data (MENON; KALÉ, 2013). Dynamicity in load is also present due to phenomena moving through the simulated area. The communication of this application class generally has a regular and static behavior, similar to seismic wave simulations (PILLA, 2014).

Table 2.1 summarizes some examples of the scientific applications according to the dynamicity of their computational load and communication, which some applications are used in thesis.

Table 2.1: Examples of scientific application and dynamicity.

Class	Application	Load	Communication
Cosmological Simulations	ChaNGa	Dynamic	Static
Molecular Dynamics	NAMD	Dynamic	Static
Seismic Wave Simulations	*Ondes3D*	Dynamic	Static
Unstructured Grid	CFD	Dynamic	Dynamic
Weather Forecasting	RAMS	Static mainly	Static

Source: The author

2.2 Load Balancing Heuristics

Load imbalance is one of the challenge problems in parallel applications. Load imbalance is caused by a distribution of load that forces some processes to be idle between synchronizations (TALLENT; ADHIANTO; MELLOR-CRUMMEY, 2010). In this context, load balancing techniques are used to achieve a better computational load distribution, and to avoid load imbalance in parallel applications. Their main goal is to increase the resource utilization. This is usually achieved by avoiding processor overloading, and also mapping tasks in a way that decreases the communication cost.

Load balancing is considered a NP-Hard problem, due the application and platform characteristics and also due the hard solutions to achieve a balanced mapping (LEUNG, 2004; EL-REWINI; ABD-EL-BARR, 2005; BRUCKER; BRUCKER, 2007).

For this reason, currently we have several different load balancing heuristics. These heuristics can use several kinds of information, being currently processor load the most used. Besides the load of the processor, heuristics can use the load of individual core, the processor speed, the memory hierarchy, communication costs, network topology and migration costs (CHARM++, 2014; EL-REWINI; ABD-EL-BARR, 2005).

Current load balancer strategies uses a relation to compute the load imbalance (L_ι) present in the system (PEARCE et al., 2012; TALLENT; ADHIANTO; MELLOR-CRUMMEY, 2010). Most of them compute L_ι as the ratio between the maximum load and average load, as presented in Equation 2.1:

$$L_\iota = \frac{L_{max}}{L_{avg}} - 1 \tag{2.1}$$

where L_{max} is the load of the most loaded processor, and L_{avg} is the average load of all the processors. So, periodically, the load balancer strategy obtains the load information and computes the load imbalance.

This equation is used in some load balancing strategies, since a parallel program only finishes its execution when the most loaded processor completes all its tasks. This way, the total execution time of an application is generally defined by the most loaded processor. This means that all other processors will remain waiting for it to complete its work. So, L_{max} represents the current execution time of the application. On the other hand, L_{avg} represents the ideal execution time of the application when the load of all processors is balanced (LEUNG, 2004).

In some parallel programming models, for example CHARM++, the load balancing is based on a heuristic known as the *principle of persistence* for iterative applications. This principle states that in some types of applications, the computational loads and communication patterns tend to persist over time, even in dynamically evolving computations. This behavior allows to use recent history as an indication for predicting load or communication in near future iterations (KALÉ, 2002; KALÉ; KRISHNAN, 1993; CHARM++, 2014; ZHENG et al., 2011; MENON et al., 2013). The principle of persistence is valid for a large class of iterative HPC applications (SAROOD; MENESES; KALÉ, 2013). It can be used as an automatic method to obtain load information without manually predicting the load (ZHENG et al., 2010; ZHENG et al., 2011).

2.2.1 Load Balancing Decisions

Making the right load balancing decisions is one of the most important problems when trying to achieve an optimal load balance. However, making this decision generates overhead, since in each load balancing step is necessary to get the load information from all processors, make the decisions, and, if necessary, migrate the tasks between processors. Choosing the right interval between calls to the load balancer, is decisive to reduce the load balance overhead.

Some load balancing strategies run the algorithm to make decisions in periods of time specified previously by the user. However, if the strategy is performed very frequently, it may incur in a reduction of performance, because the load balancing overhead or also called load balancing times, may exceed its benefits. On the other hand, if the load balancer is invoked in long time periods, the load imbalance may increase too much and also result in loss of performance. So, even when the strategy detects that there is load imbalance, $L_\iota > 0$, according to Equation 2.1, it may not yet be useful to invoke the load balancer due to their own overhead.

Considering this, some current strategies are designed aiming at dynamicity. They analyze the application issues and the platform behavior to make decisions on runtime, without specified parameters. As a solution to decrease the load balancing overhead, several recent strategies have adopted a *threshold* to determine if load balancing must be performed or not. That is, the load balancing heuristics will only execute if $L_\iota >$ *threshold*. Most current strategies use heuristics together with *threshold* to define the load imbalance.

Zheng (ZHENG et al., 2010; ZHENG et al., 2011) for example, uses a similar model to the one shown the Equation 2.1 to compute the load imbalance (L_ι). To define the *threshold* they have used a benchmark to create tasks (t) on available processors (p) where $t > p$.

In each load balancing invocation, the strategy gets load information of each task on every processor and defines the current load of the most overloaded processor. The current load before load balancing is called L_{max} and after load balancing is called as L'_{max}. This way, the *threshold* is defined as the gain from load balancing ($L_{max} - L'_{max}$) in relation to the cost of load balancing (C_{lb}). So, the load balancer will only execute when:

$$gain(L_{max} - L'_{max}) > C_{lb} \tag{2.2}$$

This is, the load balancing strategy will only be executed when its estimated (the gains *threshold*) are higher than its estimated own execution time.

2.3 Related Work on Load Balancing

Load balancing strategies can be classified into two categories. In the first category, the approaches are applied when new tasks are created in the applications. In these strategies, also called of task scheduling, the tasks are the basic unit of work for load balancing. Thus, the load balancer makes decisions on task pool. These applications are typically *non-iterative*, as described in Subsection 2.1.2.

The second category, called periodic load balancing are the strategies suitable for *iterative* applications, such as NAMD, FEM, climate simulation and others. In these applications, the computation typically consists of a number of iterations or time steps. Wherein consecutive iterations have similar behaviors. Therefore, load balancing strategies in this category make load balancing decisions every time a certain number of itera-

tions is executed.

This means that, in iterative applications, the load balancing heuristics is performed with a defined frequency. After performing a specific number of iterations, the application is stopped and the load balancer strategy gets information, based on which it decides if it is necessary to migrate some of the tasks, in order to improve the load balance. As discussed in Subsection 2.2.1, this decision making process, performed by the load balancer can be expensive in terms of application execution time.

Periodic load balancing strategies have been used on iterative applications that have persistent load patterns. These strategies make predictions of future computation load based on past information. Their decisions assume that the load will not change significantly in the following iterations, as discussed in Subsection 2.1.1.3.

Several strategies have been proposed to address the load balancing problem. They can be classified based on where the decisions are made. Some strategies employ a centralized approach, where load information is collected on a single processor, in which the decisions can be made sequentially. Other strategies aim at a better scalability, when used in larger scale systems, with a large number of processors. These strategies adopt distributed approaches where distributed processors make decisions using their local view of the system. A third kind of strategies, seeks to reduce the excessive data collection by using a hierarchical structure. In this case, the decisions are made at each level of the hierarchy. These three classes of strategy are briefly discussed in the following sections.

2.3.1 Centralized Strategies

Centralized strategies make load balancing decisions on a single processor. Load and communication data of the entire machine is accumulated in a specific processor, which runs a sequential decision process based on this information. In these strategies, the process of collecting information from all processors can impose a high overhead in the system. So, centralized approaches are only recommended to system with smaller amounts of processors. Otherwise, the load balancer may run into problems of scalability and memory usage (ICHIKAWA; YAMASHITA, 2000; LEGRAND et al., 2004; PHILLIPS et al., 2002; GALINDO; ALMEIDA; BADÍA-CONTELLES, 2008; BHATELÉ et al., 2008; CARIÑO; BANICESCU, 2008; MARTÍNEZ et al., 2011; ZHENG et al., 2011).

2.3.1.1 Basic Centralized Load Balancing Strategies

In this subsection, we will present examples of basic centralized load balancers. We call them basic, because they only consider the load on their decisions. They are:

- **GREEDYLB** is an aggressive scheduling algorithm. This strategy employs a greedy heuristics to make its decisions using only computational loads information. This algorithm does not take into account other information and also does not considers the current place of tasks into processors to decide the migrating tasks (KALÉ; KRISHNAN, 1993).

  The algorithm sorts the tasks in decreasing load order, and iteratively maps the task with the highest load to the least loaded core. Thus, this centralized strategy can be used to quickly mitigate load imbalance not limiting the number of objects migrated (ZHENG et al., 2011).

- **REFINELB** - makes decisions considering the current load distribution, differently from GREEDYLB, which makes load balancing decisions from scratch. REFINELB improves the load balance by incrementally adjusting the existing mapping. The refinement has a limit, or threshold, on the number of objects that can be migrated. This is done to diminish the overhead in the application. So, this approach migrates only a fraction of objects, therefore reducing data migration. This makes REFINELB be more efficient than GREEDYLB in some applications (KALÉ; KRISHNAN, 1993).

  In its first step, REFINELB's algorithm divides the cores in two groups according to a previously defined load threshold. So, a core is considered overloaded when its load is greater than the threshold. In the second step, the algorithm verifies the possible migrations from the overloaded cores to non-overloaded cores. REFINELB tries to reach a load distribution that leaves the core loads close to the average. This algorithm is less aggressive than GREEDYLB, because it works over the current task mapping. The load imbalance is incrementally reduced every time these steps are repeated.

- **RANDCENTLB** - differently from GREEDYLB and REFINELB, this load balancer algorithm does not take into account any information about the application or platform to make its decisions. It randomly assigns all tasks to all processors available (KALÉ; KRISHNAN, 1993).

2.3.1.2 Centralized Load Balancing Strategies using Communication Data

Besides the computational load, some centralized load balancers also consider communication data on their decisions. In this subsection, we will present some load balancers available in this category:

- **GREEDYCOMMLB**, which extends the GREEDYLB with the addition of communication loads to its decision processes. The algorithm computes the communication cost of each task, based on the number of messages sent to tasks mapped to other cores. So, instead of simply mapping of the task with the biggest load to the least loaded core, as done by GREEDYLB, it also considers all other cores that have tasks that communicate with it, making this strategy more expensive (ZHENG et al., 2011).
- **REFINECOMMLB**, similarly to GREEDYCOMMLB, extends the REFINELB algorithm by including communications costs in the decision process. The algorithm considers the communication cost between cores used by the application to compute the load imbalance. The communication data is also considered to map tasks into a different processor (KALÉ; KRISHNAN, 1993).
- **SCOTCHLB** is a static mapping algorithm based on the dual recursive bi-partitioning. It allows map efficiently using both the source process graph and the target architecture graph to map process. It can also use any weighted source graph onto any weighted target graph (PELLEGRINI; ROMAN, 1996; SCOTCH, 2015).

2.3.1.3 Centralized Topology-Aware Load Balancing Strategies

More recently, some load balancers began including network, and memory topology information in their heuristics. Some of the load balancers of this category are listed in following:

- **TOPOAWARELDB** is a load balancing strategy that uses information about the topology to map applications on 3D torus/mesh architectures (BHATELÉ; KALÉ; KUMAR, 2009). This strategy has been applied in several applications, among them NAMD, as discussed in Subsection 2.1.1.3.
- **NUCOLB** is a strategy developed for parallel platforms with non uniform topologies. It takes into account the memory topology of NUMA systems. Besides trying to mitigate the load imbalance, also tries to reduce communications costs by keep-

ing communicating tasks in the same NUMA node (PILLA et al., 2012; PILLA et al., 2014).

- **HWTOPOLB** extends the NucoLB, aiming at improving its performance by decreasing the costs of communication. This load balancing strategy takes into account the communication costs of the whole system's topology and information from the application, as workload and communication patterns captured during its execution. The communication costs are measured through a selected benchmark (PILLA et al., 2012; PILLA et al., 2014).

2.3.1.4 Centralized Temperature-Aware Load Balancing Strategies

Newer centralized load balancers have considered system temperature to avoid overheating and reduce cooling costs. We present, in this subsection, some load balancers available in this category:

- **TEMPLDB**, different from most load balancers, does not use computation load to take its decision. Instead, it limits the temperature of processors to a temperature barrier. Its goal is to reduce the energy spent in cooling (SAROOD et al., 2012; SAROOD; MENESES; KALÉ, 2013).
  The TEMPLDB algorithm collects the temperature of each core in a distributed mode. Its decision is made in a specific processor, using a barrier to control the temperature. A DVFS technique is applied to reduce the processor's temperature. Temperature changes to a processor affect all of its cores (MENON et al., 2013).
- **METATEMPCONTROLLER** is an improvement of the TEMPLDB. This temperature-aware strategy automatically control the processors temperature, avoiding hot spots without any support from the user (MENON et al., 2013). The algorithm is implemented using a *threshold* value. This way, automating the decision making and avoiding the overhead caused by load balancing calls. METATEMPCONTROLLER computes an ideal period between calls of the load balancing. It takes into account information about the application and the cost to execute the load balancer on system.
- **Merkel *et al.*** employ a similar strategy to TEMPLDB. It is a task scheduling algorithm to avoid overheating in multiprocessors systems. Merkel uses data from monitoring counters to implement a mechanism to assign tasks to CPUs. This monitoring data is also used to tasks migrate to avoid hot spot in systems (MERKEL;

BELLOSA, 2006).

- **Goel *et al.*** present a model that uses CPU performance counters and CPU temperature data to estimate an accurate per-core power. This model provide information that can be used to guide scheduling decisions in power-aware managers (GOEL et al., 2010).
- **Hartog *et al.*** developed an approach that uses the relationship between CPU temperature and energy consumption to provide a method for estimating the power usage of the system (HARTOG; DEDE; GOVINDARAJU, 2013).
- **Kodama *et al.*** use a similar approach to Hartog. They state that homogeneous nodes have different power consumption due their positions. They implement a relationship between CPU load and fan speed once these behavior generate an imbalance in the power consumption of fans. This way, they also introduces a new metric that based on the power and heat of each node, aims to reduce the power demand of fans (KODAMA et al., 2013).
- **Seyedmehdi *et al.*** implement a task scheduling approach to cloud computing. It aims to benefit both cloud providers and their customers. The model regulates the execution speeds of real-time tasks in a way that a host reaches the optimum level of utilization instead of migrating its tasks to other hosts (HOSSEINIMOTLAGH; KHUNJUSH; HOSSEINIMOTLAGH, 2014).

2.3.1.5 Centralized Energy-Aware Load Balancing Strategies

Several strategies and algorithms have been proposed focusing mainly in increasing resources usage and reducing the execution time. However, few approaches have made efforts to further improve the energy consumption on load balancing context. In this subsection, we present a load balancer available in this category:

- **Aupy *et al.*** propose an energy-aware scheduling model to schedule tasks under reliability and makespan constraints. This approach has been evaluated using simulations with different heuristics based on the probability of failure, the task weights, and the processor speeds. These heuristics aim at minimizing the energy consumption while enforcing reliability and deadline constraints considering an expected energy consumption for a second execution of tasks (AUPY; BENOIT; ROBERT, 2012).

Centralized load balancing strategies are able to achieve good load balance. As such, several scientific applications have used them to increase their performance. NAMD (PHILLIPS et al., 2002) and ChaNGa (JETLEY et al., 2008) are examples of these applications (ZHENG et al., 2011). However, the centralized approach generally presents low scalability in large scale systems, due to large amount of information used in the decision process. For example, Zheng *et al.* measured the increase in memory overhead when using centralized strategies in a HPC system with 65,536 cores (ZHENG et al., 2010). Future Exascale system will probably have a few thousands of processors. It would hardly be possible to make a centralized approach to scale to this level. As an alternative solution, the research community has proposed distributed load balancing strategies, which will be discussed in the next subsection.

2.3.2 Distributed Strategies

Distributed load balancing strategies aim to improve the performance of load balancers on large scale systems. In these strategies, processors exchange load balancing data only among their neighbors, as a way to decentralize the load balancing process (CYBENKO, 1989; BAHI; CONTASSOT-VIVIER; COUTURIER, 2005; ZHENG et al., 2011). This way, different from centralized approaches, distributed strategies do not have a global load balancing point. The drawback is that achieving a load balanced may take more time, since each processor has a limited set of information about the entire system. Therefore, these strategies present a slow convergence. However, working with a smaller set of data improves scalability, since there is less to process.

On the other hand, distributed strategies are more scalable than centralized strategies, since processors make decisions using only local information of the system (CYBENKO, 1989). It's important to point that in very large systems, even with these strategies, which are developed to be scalable, may still suffer from scalability problems. The cause for this issue is the information exchange among the nodes that compose the system. To avoid this problem, current strategies have used some communication info, similarly to GREEDYCOMMLB and REFINECOMMLB strategies.

2.3.2.1 Basic Distributed Load Balancing Strategies

In this subsection, we will present examples of basic distributed load balancers. We call them basic, because they only consider the load on their decisions. They are:

- **GRAPEVINELB** is a distributed strategy, which has the goal of presenting less load balancing overhead than similar centralized strategies. Its algorithm makes load balancing decisions in two stages. In the first, which is performed in parallel, the average load of each processor is computed. This average value is used to define the global state of the system. In the second stage, the overloaded processors receive the information about the underloaded processors, and make decisions about whether or not to transfer some of their tasks to underloaded processors (MENON; KALÉ, 2013).
- **GRAPEPLUSLB** is an improvement of GRAPEVINELB. The main modification is to avoid that underloaded processors receive the information about the overloaded processors after the transference of tasks (MENON; KALÉ, 2013).

Distributed strategies do not have a global synchronization point, which has a positive impact in their scalability. However, with the expected scale of futures systems, which will be much large than current large scales systems, it will become harder for these strategies to collect load balancing data that are distributed throughout the system. Therefore, we need new strategies to solve this load balancing data distribution problem. One of these strategies is called, hierarchical load balancing, which will be discussed in the next subsection.

2.3.3 Hierarchical or Multi-Level Strategies

Hierarchical load balancing strategies divide the processors in groups, which are organized in hierarchical levels, according to system organization. To make decisions, the hierarchical algorithms aggregate the load data in lowest level of the hierarchy. This allows them to decentralize the decision making and minimizing the communication among the processors. This way, these strategies have a reduced time to make decision and also need less memory to save load data only of the sub-group (ZHENG et al., 2011).

Most implementations of hierarchical strategies use only two independent hierarchical levels, due to overhead concerns. In the highest level is the whole platform. This

level, also called of root or group leader, is composed of several sub-levels organized in a sub-tree. In the second level, also called leaves, a centralized load balancing strategy can be performed to balance the load of the processors of their sub-tree. This way, hierarchical strategies allow to execute load balancing algorithms in parallel on different sub-tree. So, they can exploit the parallelism available in these systems, which helps them to scale in systems with a very large number of processors, without reducing the efficiency of the load balance.

2.3.3.1 Basic Hierarchical Load Balancing Strategies

In this subsection are presented some examples of basic hierarchical load balancers. Similar to centralized and distributed load balancing strategies these also only consider the load on their decisions.

- **HYBRIDLB** is a periodic hierarchical load balancing developed for the CHARM++ environment. It divides the processors into independent groups organizing them in a hierarchical tree. The strategy creates a tree based on the machine topology, aiming to reduce the communication overhead and improve locality. This hierarchical implementation reduces the amount of data in the tree, since data are aggregate to sub-tree, avoiding the communication among them. Similarly, load balancing decisions are made in each sub-tree to minimize the cost of tasks migration (ZHENG et al., 2011).
- **HIERARCHICALLB** aims to reduce the load balancing overhead by dividing the load balancing decisions in hierarchical levels. It allows the use of different load balancers in different levels. At platform level, it employs HwTopoLB, a centralized load balancing, to distribute tasks over nodes, and at node level, it can run either HWTOPOLB or NUCOLB to map tasks inside a node. Applying task mapping in two levels, this organization takes into account the machine topology, different mechanisms can be used for communication, as interconnection network between system nodes and memory hierarchy inside a node (PILLA et al., 2012).

As seen in this subsection, hierarchical strategies aims to overcome issues present in centralized and distributed strategies. These strategies provide good load balancing performance and good scalability in large scale systems, while requiring a smaller amount of communications. Table 2.2 summarizes these approaches considering load balancing convergence, scalability and load balancing overhead.

Table 2.2: Strategies characteristics.

Strategy	**Load balancing**		
	convergence	**scalability**	**overhead**
Centralized	*fast*	*poor*	*high*
Distributed	*slow*	*good*	*low*
Hierarchical	*fast*	*good*	*relative to levels*

Source: The author

2.4 Discussion

HPC systems have been continuously increasing their performance to meet the computational demands of scientific applications. Unfortunately, this also increases their energy consumption. This strategy has made the energy consumption to become a limiting factor to the scalability of future systems. So, in this chapter we presented a characterization of computational load of scientific applications and a load balancing discussion.

Besides the increased energy consumption, HPC systems also grew in number of computational nodes, it gets harder to achieve a high level of efficiency in the use of the available resources. A high resource usage is essential for achieving the best performance possible on these systems. To mitigate this problem, parallel scientific applications need to employ load balancing strategies, which try to equally divide their workload across all processors.

Load balancing is a challenging problem, which has been widely studied and used as a way to improve the performance of parallel applications (KALÉ; BHANDARKAR; BRUNNER, 1998; EL-REWINI; ABD-EL-BARR, 2005; ZHENG et al., 2011). In this context, different strategies have been proposed aim to overcome the load balancing problem. These strategies generally are classified based on where the decisions are made, being called of centralized, distributed or hierarchical strategy.

Most of these load balancing strategies are mainly based on application characteristics and architectural aspects of the machine. They analyze the application information and the platform behavior to make decisions on runtime, being computational load, processor speed, communication cost and hardware topology the most data used for making load balancing decisions.

More recently, some load balancers began including network, and memory topology information to make its decisions and nowadays some strategies start include decisions based on system temperature on their heuristics. However, few approaches have

made efforts to further improve the energy consumption on load balancing context. A few recent strategies started including decisions based on system temperature on their heuristics (ZHENG et al., 2011).

A subset of the current load balancing algorithms is compared in Table 2.3, according to the criteria they use in their decisions.

Table 2.3: Load balancing algorithms comparison in terms of make decision criteria.

Algorithm	**Category***	**Load**	**Communication**	**Topology**	**Temperature**	**Energy**
GREEDYLB	C	•				
REFINELB	C	•				
GRAPEVINELB	D	•				
GRAPEPLUSLB	D	•				
Seyedmehdi *et al.*	C	•				
GREEDYCOMMLB	C	•	•			
REFINECOMMLB	C	•	•			
TOPOAWARELDB	C			•		
NUCOLB	C	•	•	•		
HWTOPOLB	C	•	•	•		
HIERARCHICALLB	H	•	•	•		
HYBRIDLB	H	•	•			
TEMPLDB	C				•	
METATEMPCONTROLLER	C				•	
Merkel *et al.*	C				•	
Kodama *et al.*	C				•	
Goel *et al.*	C				•	
Hartog *et al.*	C				•	
Aupy *et al.*	C					•

*Category: (**C**)entralized (**D**)istributed (**H**)ierarchical

Source: The author

During a long period of time, research in load balancing strategies focused in increasing performance and scalability. Power demand or total energy consumption were not take into account. Instead, most load balancers implemented on several different programming models, focused mainly on reducing the execution time or communication cost. Some of these approaches seek to improve the performance avoiding imbalance by improving the distribution of the load among processors. The main result would be a reduction in the total execution time. It's worth noticing that these strategies may also reduce the application's energy consumption, since they will be using the machine for a shorter period of time.

Today, power and energy consumption have become critical in the HPC field. These are the main design constraints to build the future Exascale systems, which will include thousands of processors. In this context, the main challenge is to improve performance (or to keep the same) while reducing the power demand and the energy consumption of these large scale HPC systems. Despite the existence of several load balancers

focused on performance optimization, few works have made efforts to further improve the energy consumption in the context of load balancing. Thinking in future Exascale machines, that will have a large amount processors and will have high power demands, it becomes necessary to develop of load balancing approaches that take into account energy consumption and power demand characteristics in their decisions.

Our research also aims to fill this gap by proposing and implementing a new load balancer strategy aimed at reducing the average power demand and the total energy consumption of the system while running applications.

Differently from other load balancers, Aupy *et al.* proposed a centralized energy-aware strategy that consider the number of failures on HPC system aiming to avoid the loss in reliability. They propose a model of scheduling with fault-tolerance. This model uses re-execution of task while reducing the energy consumption, considering an expected energy consumption for a second execution of tasks (AUPY; BENOIT; ROBERT, 2012). This approach is different from ENERGYLB once that our proposed load balancers aim reduce the average power demand during the runtime when imbalanced applications are executed.

In addition, besides the variables used by current and traditional algorithms, our proposal takes into account the power demand of the system. In this way, load balancing decisions are based on performance models considering the computational load coupled with power demand constrains. Our proposed approaches focus on dynamic load balancing, allowing the application improve the usage of available resources of parallel HPC machines as well as increasing the energy efficiency during their execution. These approaches will be presented in Chapter 4, however, ways to obtain information about the energy and power of parallel machines will discussed in the next chapter.

3 ENERGY CONSUMPTION OVERVIEW

Considering the global rate of increase in energy consumption and the rate of increase of the number of HPC systems, it is estimated that by the year 2050 the total power demand of the planet will reach 30 Terawatts (EIA, 2012). This would exceed the planet capacity of generation (ENERGIA, 2011; INPE, 2010). Based on these projections, we see why the energy cost has become an integral part of the *Total Cost of Ownership* (TCO) of these systems. Thus, making companies and organizations begin to recognize the importance of *green* policies as an alternative to reduce equipment maintenance costs.

According to Younge *et al.*, supercomputers's performance increased more than 3000 times in 20 years, while the performance per watt increased only 300 times and the performance by m^2 increased only 65 times (YOUNGE et al., 2010). In this context, the efficient use of HPC systems can represent a significant reduction in power demand. Therefore, approaches that aims to improve the energy efficiency are being widely adopted. Furthermore, new approaches are being proposed with significant frequency.

Therefore, it is important to improve the energy consumption of these systems both in the hardware as in the software level. For this purpose, it is necessary to understand the system power demands and application's behavior, at the entire system level as well as at the processor level. In this context, it is important to study the energy efficiency of these systems, to overcome the current challenges and build Exascale systems without exponentially increasing power consumption. Thus, there are several studies being developed with the goal of keeping the power demand below the 20 MW limit specified by DARPA (BECKMAN et al., 2011).

In 2008, specialists alerted on an official DARPA report (KOGGE et al., 2008; BECKMAN et al., 2011) that the acceptable power budget to reach Exascale is 20 MW. In this context, an Exascale system has a limit of $\frac{1\ EFlops}{20\ MW}$, i.e., 50 GFlops/W. Tianhe-2, the current number 1 in the Top500 rank, produces 33.8 PFlops spending 17.8 MW, thus Tianhe-2 energy efficiency is 1.91 GFlops/W. Using Tianhe-2 as reference, we would have an increase the energy efficiency 26-fold to match the DARPA recommendation. Thus, to get to Exascale we need to find alternatives to solve the power demand problem (BARKER et al., 2009; YOUNGE et al., 2010; PADOIN et al., 2013a).

In this thesis, we frequently use technical terms related to energy. Therefore, in this chapter we review the fundamental aspects related to power demand and energy consumption of HPC systems. In Section 3.1, we present the basic concepts and different

issues related to the characterization of energy consumption and energy efficiency metrics. We also seek to better define the characteristics of this consumption in HPC systems, as well as elucidating the units involved in the measurement of both the energy consumption and performance of these systems. Solutions for power and energy evaluation are presented in Section 3.2. In Section 3.3, we discuss the state of the art on frequency scaling approaches, which are a solution to reduce power demand present in current processors. Finally, we present the chapter's discussion, highlighting some points related to the context of this thesis.

3.1 Energy Consumption Metrics

In this section, we review some definitions and concepts related to units considered in the measurement of power demand and energy consumption.

In the state of the art, units of measurement are defined arbitrarily as a reference standard. The main units are defined in International System of Units. The understanding of these units is necessary for the analysis and measurement of energy consumption in HPC systems (FOWLER, 1992; JOHNSON; HILBURN; JOHNSON, 2000; HEWITT, 2002; STEIGERWALD et al., 2011). The units, in this context, are discussed in the two next subsections:

3.1.1 Power Demand

Power or power demand (in watt) represents the rate at which work is done. In the context of energy use, power demand describes the rate at which the energy is consumed or transferred. So, power demand means that the electric utility company must have ready or provide to customers when they turn the equipment on. Generally power is expressed as a function of the time, so it is equivalent to an amount of energy conversion or energy consumed (in joule) per unit time (in second) as shown in the Equation 3.1. Time, in several works, is been also called runtime, performance and *total execution time*, as in the context this thesis.

$$Power = \frac{Energy}{Time} \tag{3.1}$$

For integrated circuits made with CMOS technology, the power demand can be

defined as proportional to the square of the voltage (HSU; FENG, 2005), as is given in Equation 3.2:

$$P = f \times C_L \times V_{dd}^2 \times \alpha \tag{3.2}$$

where f is the clock frequency, C_L is the circuit load capacitance, V_{dd} is the supply voltage, and α is the number of switches per clock cycle (CHANDRAKASAN; SHENG; BRODERSEN, 1992; KRISHNA; LEE, 2000; KIM et al., 2012).

Current processors are manufactured with the lowest possible supply voltage technologies. This approach results in energy savings, however, it incurs in time constraints, which will be discussed in the Section 3.3.

3.1.2 Energy Consumption

Energy is a physical quantity that represents the ability or capacity to perform work. Energy consumption is usually measured in Joules, may also be used KWh unit when it comes to energy bills. So, in the computational field, to run some task, an equipment or a processor consumes a determined amount of power over a determined period of time. In this context, energy consumption can be defined as the amount of joules spent to execute a task and it is obtained by solving the integral of the function of power over a time domain (KRISHNA; LEE, 2000).

Several approaches simplify the calculation of energy consumption, by computing it as the average power over a period of time, also called Power-delay product (PDP), as shown in the Equation 3.3:

$$E = P_{avg} \times t \tag{3.3}$$

where P_{avg} is the average power, and t is the time period.

Similar to PDP metric, the Energy-delay product (EDP) metric is also used, which considers delay^2 instead of delay.

3.2 Energy Consumption Evaluation

Nowadays, there is an increasing interest in improving the energy efficiency of high performance system and applications. Accurate power and energy consumption measurement is of great importance to achieve this goal. One approach to evaluate the energy consumption of an application is to use devices that measure the power supplied to the system on which the application is being executed, which are discussed in next subsections.

Another approach is to estimate the energy consumption based on theoretical models. These estimation models, however, are out of the scope of our research. Therefore, they will not be addressed in this work.

3.2.1 Power and Energy Measurement

There are two well-known methods to measure the power and energy consumption of HPC systems, depending on the system's organization. The first is based on use of devices that are connected to components located inside or outside of system. The second uses sensors integrated directly in the system's hardware. The latter are becoming more common in currently systems.

Once the energy data is acquired, the energy consumption can be analyzed from two points of view. One looks at each component's individual consumption. The other, at the consumption of the whole system. We separated our analysis of the measurement approaches in two subsections. The first one deals with External Measurement and the second deals with Integrated Measurement.

3.2.1.1 External Measurement

Several external equipment have been used for measuring the power demand and energy consumption of HPC systems. Generally, these power meters are installed between the power plug and the system. This approach provides data about the equipment as a whole. It is widely employed, mainly because these devices are easy to deploy, since they get their measurements directly from the alternate current (AC), which is important in system's designs.

Different studies have used this approach to measure the power demand on HPC

systems. For this purpose, several different devices have been applied. Steigerwald in (STEIGERWALD et al., 2011) asserts that there are still very few tools to measure power demand in large and complex HPC systems. In this context, the equipment used for this measurement include Amperemeter, Voltmeter, Wattmeter, Multimeter, Oscilloscope, Power and Energy Analyzers and Power Distribution Units (PDU).

A different approach has been adopted by the Green500 list. It uses a general method for measuring the power and determine the performance per watt metric of each system. Aiming to minimize the variations in the measurement of energy consumption, its approach assumes that the Linpack benchmark's execution is balanced among all the nodes of system and that all nodes are identical. Consequently, all nodes have the same power demand. Based on these assumptions, only one of the nodes has its power demand measured. To obtain the total demand, the measured value is multiplied by the total number of nodes in the system (GE et al., 2007).

External measurement devices allow measurements with a high level of granularity, due to the fact they measure the consumption of the whole system. Depending on the type of device, the measurement can be different in some points. For example, in some devices display the metric for active power is the average power calculated from the energy consumed over one second. Other devices measure the instantaneous power, obtained by measuring the current and the voltage of machine. This difference in approaches incurs in different accuracies and measurement frequencies.

Furthermore, the use of external equipment, which are connected between the power source and the system, generally provides data in different format or frequency making it difficult to integrate with data from other devices. This forces a synchronization of the different data, thus increasing the potential inaccuracy of the results, as discussed in Section 4.2.

3.2.1.2 Integrated Measurement

Another kind of measurement equipment are those installed internally to systems. These usually work with direct current. This approach is normally implemented through sensors that provide more complex information than external devices. With these devices, it is possible for a system to allow the acquisition of power or energy data from each individual component.

Current system architectures implement sensors that can be accessed through a software interface, in order to obtain different kinds of information, among them power

demand and energy consumption. These sensors enable the separate measurement of different system components with a low granularity. Besides that, in some cases, this information can be managed.

Modern processors, like Intel, AMD and ARM, provide power and energy information. This is done through sensors integrated into the processor's chip itself. Intel processors, for example, feature a Package Control Unit (PCU) and integrated microcontrollers that allow power management. The PCU collects information such as temperature and power, and stores it in Model-Specific Registers (MSRs) included in the processor. These MSR can be mapped to pseudo files by Linux kernel module. This file then can be used to monitor the execution of a program.

The software interface used to access the MSRs in Intel processors is Running Average Power Limit (RAPL) (INTEL, 2009). Its infrastructure provides an estimate of the current energy usage based on a model driven by hardware counters, temperature, and leakage models (WEAVER et al., 2012). The version of RAPL introduced in the Intel Sandy Bridge architecture provides a set of registers with consumption statistics and mechanisms to set power limits (SUBRAMANIAM; FENG, 2013). The data generated by this model is available to the user, with an update frequency on the order of milliseconds. RAPL is capable of providing power and energy information from the processor package, core subsystem, and DRAM (ROTEM et al., 2012).

Similarly, with AMD processors, the Application Power Management (APM) interface can be used. It provides very similar features to Intel RAPL. APM provides power management and access to an estimation of the instantaneous power dissipated by the processor.

Several of the current machines feature a specific microcontroller embedded on their motherboard called Baseboard Management Controller (BMC). This microcontroller captures data from sensors present on the motherboard. This data may include temperature, system status, voltage, power mode, power demand, fan speed, among others.

BMC is present in a large amount of HPC systems and can be connected through different interfaces, which allow greater flexibility in monitoring and management. Intelligent Platform Management Interface (IPMI) is an interface widely used to access data from BMC (IPMI, 2013; INTEL, 2009). The current IPMI version defines a set of hardware and firmware interfaces that can be used to monitor energy consumption in real time. In some computer models, it also allows the management of power supplies, in order to operate with better efficiency, thus saving power.

While RAPL provides measurement of energy consumption only by the processor, IPMI provides the total consumption of the system. However, IPMI's interface measures the power demand inside the machine, but after it passes through the power supply. So, these measurements ignore the power dissipated by the power supply unit. Depending on the power supply unit's efficiency, this ignored consumption can represent a significant portion of the total energy spent.

An important issue with these approaches is that they are available only in new architectures. So, their portability is reduced. For instance, RAPL and APM are available only in new models on Intel and AMD processors, while IPMI is found in several server hardware.

Moreover, these approaches are considered intrusive solutions, once is need frequent accesses to a register, to read the power demand data. Another problem is that the execution of an application to read data also consumes energy and this measurement can also change the measured values.

One point in which internal measurement represent an advantage over external devices is that there is no need to install additional hardware in the system. Another point is that manufactures offer software interfaces to access their data with a higher level. Therefore, they represent better convenience compared to the external meters.

Weaver *et al.* (WEAVER et al., 2012) extended the Performance API (PAPI) analysis library to measure and report energy and power values. They describe in detail the types of energy and power readings available through PAPI. Similarly, Hartog *et al.* (HARTOG; DEDE; GOVINDARAJU, 2013) explore the possibility of power-efficient scheduling without the need for expensive power monitors on every node. To do so, they consider that homogeneous cluster are heterogeneous with respect to power demand according to several aspects. Using another approach, Goel *et al.* (GOEL et al., 2010) presented a methodology for deriving per-core power models using the values of performance counters and temperature sensor readings.

3.3 Related Work on Frequency Scaling

Many research projects with the aim to reduce power demand and energy consumption in processors architectures and parallel HPC systems have been developed. Despite that, the processor remains as the component with the highest energy consumption (FENG; GE; CAMERON, 2005). From this premise, recent micro architectures fea-

ture several power controls and support dynamic power management. Processors that incorporate this technology allow to change the processor voltage level, to reduce their frequency of operation, thus saving power (CHANG; PEDRAM, 1996). This technique is called *dynamic voltage and frequency scaling* (DVFS).

Several of the current processors incorporate these DVFS techniques. Processor models, including multicore SoCs (MPSoC), that today are present in the HPC area, incorporate DVFS to reduce their power demand and energy consumption. This technique is present in architectures like Intel's Sandy Bridge, Ivy Bridge and Haswell, AMD's Opteron, NVIDIA's Tegra, Qualcomm's Snapdragon, Samsung's Exynos and Texas Instrument's OMAP Cortex and big.LITTLE family, among others. Beyond reducing the energy consumption by the processor, decreasing its clock frequency and voltage, they generate less heat. Therefore, this technique can also reduce the amount energy consumed for cooling the system.

The processor's clock frequency has a direct impact on its power demand. Based on this relationship, several models of currently processors manufactured enable the management of power through the technique of DVFS. DVFS approaches can provide significant energy savings due this direct relation between power demand and clock frequency (KAXIRAS; MARTONOSI, 2008). As discussed in Section 3.2, Equation 3.2, power demand is directly proportional to the square of the core voltage and linear to its clock frequency (CHANDRAKASAN; SHENG; BRODERSEN, 1992).

The current processors generally run tasks using the highest frequency allowed for a given voltage. In this context, reductions in the clock frequency of processors allow a reduction of supply voltage, which implies a linear decrease in power demand. However, this approach used for energy savings leads to an increase in execution time of tasks.

Therefore, these increase in the execution time, may cause an increase in total energy consumption of the application. Besides that, due to this relation, processors cannot achieve the best energy efficiency for an application by simply executing it at the highest available frequency (ROTEM et al., 2012).

An important consideration is the overhead of clock frequency switching. It takes a few microseconds to change the processor frequency from one level to another. Thus, the DVFS overhead is on the order of microseconds. According to Park *et al.* (PARK et al., 2013), cited by Gerards *et al.* (GERARDS et al., 2014) the time and energy overhead of DVFS is comparable with those of context switching. Particularly on the Intel Core2 Duo E6850 the maximum transition overhead is of 62.68 microseconds (GERARDS et

al., 2014).

Other point is the technology used to manufacture the current multicore processors. Some multicore processor models only allow their cores to run at the same clock frequency. Among them, include some Intel processors (PARK et al., 2013). On the other hand, some AMD processor models, allow each one of its cores to run at a different frequency (DORSEY et al., 2007; SPILIOPOULOS et al., 2013). Therefore, on newer processors DVFS is implemented using one of these two approaches, also called of *per-chip or global DVFS* or *per-core or local DVFS* (GERARDS et al., 2014).

DVFS techniques have been used to improve power demand and energy efficiency of parallel systems, while running several different applications. In this context, in next sections, we present the state of the art in power demand and energy consumption. Some areas that have been employed DVFS techniques include:

3.3.1 Frequency Scaling on Idle Resources

DVFS strategies have been used when detected idle state and scale frequency down to save energy. Hosseinimotlagh *et al.* use DVFS for this purpose. They point out that an idle host can consume up to 70% of its peak power demand in cloud systems. In this work, the authors propose a task scheduling approach that regulates the execution speeds of task, instead of migrating them to other hosts. Their results show a reduction up to 41% of the total energy consumption, however this approach incurs in an impact on turnaround times of real time tasks up to 85% (HOSSEINIMOTLAGH; KHUNJUSH; HOSSEINIMOTLAGH, 2014).

In the same context, Younge *et al.* present a study on the reduction of the clock frequency on multicore architectures. Their best result were achieved with a reduction of 18% in the frequency, using DVFS, with a reducing of only 5% in performance and of up to 20% in power demand (YOUNGE et al., 2010). Still on idle, Lee *et al.* implemented an Energy-saving DVFS Scheduling of Multiple Periodic Real-time Tasks on Multicore Processors containing more processing cores than running tasks (LEE, 2009). Their simulated method saves energy up to 64% executing each task on a separate core.

3.3.2 Frequency Scaling on Power or Energy Budget

Some works employ DVFS aiming to meet an energy budget. Springer *et al.* propose a technique to find a near optimal scheduling that satisfies an energy consumption limit when DVFS should be run for a particular application. Their scheduling technique combines performance modeling, performance prediction, and program execution. The scheduler tries to minimize the timing penalty for a given power limit. Using NAS benchmarks and energy limits of 95% of the value spent for each program on 2, 4 and 8 nodes, the authors achieved, in the worst case, 6.1% of the optimal scheduling (SPRINGER et al., 2006).

Similarly, Isci *et al.* proposed to fine-tune the clock frequency of processors in order to maintain the power demand below a specified budget (ISCI et al., 2006). Considering both static and dynamic configurations of multithreaded applications, the results of their proposed approach are within 1% of the performance and meeting a given power budget.

In the same context, Hernandez *et al.* propose a thread mapping strategy for NoC based CMPs that considers performance and energy consumption constraints. Their approach taking into account the assigned threads to cores in the chip to adjust the frequency and voltage of the cores in the selected region. More specifically, it uses the location of memory controllers and the concurrent execution. Their results achieve reductions of the execution time up to 23% while reducing the energy up to 24% (HERNANDEZ; SILLA; DUATO, 2011).

Rountree *et al.* also quantify the variation of the performance and power efficiency on Intel's Sandy Bridge family. Using instances of the NAS Parallel Benchmarks on single-processor, they show that, in the lack of a power bound, DVFS can be used as a tool for balancing power and performance in HPC systems (ROUNTREE et al., 2012).

3.3.3 Frequency Scaling on Communication Periods

Other works also reduce the clock frequency during communication phases of applications. Lim *et al.* propose to use DVFS strategy to change the frequency of the cores during the communication phase of MPI applications. They developed an MPI runtime that dynamically identifies communication phases with a high concentration of MPI calls and reduces CPU performance, aiming to minimize the energy-delay product

of MPI applications. They achieved a reduction of 12% in energy consumption while running the NAS benchmarks, while the execution time increased only up to 2.1% (LIM; FREEH; LOWENTHAL, 2006; LIM; FREEH; LOWENTHAL, 2011).

In this same context, Tan *et al.* proposed the Adaptively Aggressive Energy Efficient DVFS Scheduling (A2E) for Data Intensive Applications that can also be applied in memory and disk periods (TAN et al., 2013). When utilized A2E on a 64-core cluster the energy savings were an average of 32.6% with performance loss of 6.2% on average.

3.3.4 Frequency Scaling on Kernel Governors

Power management has become also an important consideration in operating systems. Current operating systems have subsystem governors that allow to scale up or down the clock frequency of processors. For this purpose, power saving and frequency scaling approaches have been integrated in some governors of operating system kernels.

Current operating systems kernels support DVFS through the *cpufreq* device driver. This driver allows governors to control the voltages and frequencies of the system's processors (PALLIPADI; LI; BELAY, 2007). When this device driver is used, an available governor can change the voltage and frequency in which the processor operates.

Linux kernel implements static and dynamic governors (BRODOWSKI, 2014). The static governors available are *Powersave* and *Performance*. The first sets the processor's frequency at minimum value available within the range supported by the processor. The second keeps the frequency always at the highest value supported by the processor. The kernel also includes dynamic governors, such as *Ondemand* and *Conservative*, which can switch clock frequency depending on the current processing demand. The Ondemand governor sets the processor frequency based on its utilization, using mainly the minimum and maximum level supported by the processor. The Conservative governor changes the clock frequency one level at a time based on threshold values.

Besides of static and dynamic governors, some kernels have other governors, such as the *Userspace* governor, which lets users to set a specific frequency in user space. In this way, an application can control the frequency of the processor. There is also the *Interactive* governor, which is similar to Ondemand, in which the frequency is also changed based on processor utilization. However, this last governor is available only in some ARM processors, like big.LITTLE models.

Other studies using DVFS in simulations aiming to control the system's power de-

mand as kernel governors. Spiliopoulos *et al.*, developed an approach using software and hardware issue. They extended the gem5 simulator to support full-system DVFS modeling and developed a framework with specifications and complies of the hardware and software conventions. Comparing their extended version to different DVFS governors, as interactive, on-demand and performance, the authors concluded that the interactive governor is faster than on-demand, achieving better performance at about the same energy consumption (SPILIOPOULOS et al., 2013).

In a similar study, Semeraro *et al.* describe an approach in which the processes are divided into several clock domains. Within each domain, independent voltage and frequency scaling can be performed. They evaluate their design using the SimpleScalar simulator with four clock domains. Using benchmark suites, they obtained an energy-delay product improvement of 20%, compared to 3% savings from voltage scaling a single clock and voltage system (SEMERARO et al., 2002).

Gerards *et al.* use DVFS strategies to decrease the energy consumption in the context of multicore processors. This is a complex problem since these processors adopt a global DVFS, where the voltage and clock frequency is shared among the cores in the same chip. Using the amount of parallelism of applications, they proposed a theoretical method to transform the problem of finding an optimal clock frequency on global DVFS systems to a single core problem. Their proposal was validated in gem5 simulator using performance, ondemand and interactive governors (GERARDS et al., 2014).

3.3.5 Frequency Scaling on Performance Prediction

DVFS strategies are also employed together with performance prediction. Kim *et al.* proposed a realistic DVFS performance prediction method and a practical DVFS control policy (eDVFS). They compare the eDVFS Linux's ondemand governor, while running the SpecCPU 2006 benchmark. Defining a static optimal frequency for each program performing all combinations of each available frequencies, eDVFS can save up to 20% energy with voltage variation of 32% (KIM et al., 2012).

Freeh *et al.* developed a static framework that allows to perform each phase of a single application in different processor frequencies. In a first execution, it divides the program into phases, and measures the energy consumption and execution time of each phase using a specific heuristic. In a second execution, it uses a heuristic to choose the best frequency for each phase of application allowing then to save energy. The results

achieved over NAS benchmarks show save savings of up to 16% with an overhead of 1% (FREEH; LOWENTHAL, 2005).

Raghu *et al.* introduced the Power Aware Algorithm for Scheduling (PAAS). PAAS predicts an optimal clock frequency or processor's voltage for each task using information obtained from previous executions of HPC applications (RAGHU; SAURAV; BAPU, 2013). The authors evaluated the proposed PAAS with the set of scientific applications achieving reductions on energy spent between 12.6% and 13.5%.

Still on DVFS performance prediction, Peraza *et al.* introduce the Green Queue framework for generation and analysis of traces. By examining traces generated during the previous executions, their framework is able to instrument the application with DVFS controls. Tests realized on 1024 cores with this framework achieved energy savings of up to 21% for the intratask and 32% for intertask DVFS strategies (PERAZA et al., 2013).

3.4 Discussion

Due the expensive and increasing cost of the energy and also the power demand constraints, saving power has become one of the main concerns of the HPC community. Thus, to build future systems, power demand and energy consumption limits need be taken into account, since the total cost of equipment maintenance can overcome in a few years its cost of acquisition. Therefore, in this chapter we presented a review of the fundamental aspects related to power demand and energy consumption of HPC systems.

State-of-the-art research has proposed several techniques to reduce the power demand and improve the energy efficiency of parallel platforms. DVFS is one of these techniques, which is used both at hardware level, as well as in the context of operating systems.

Currently, DVFS techniques have been employed on different areas. Among them, the more relevant are idle resources, power or energy budget, communication periods, kernel governors and performance prediction, as discussed in this chapter. However, most of the cases, these techniques are employed to reduce the total cost of operation of the machine itself, as well as the cost of energy used for cooling. As much as these approaches can save energy, however when employed they may incur in degradation of the performance of the system and increase in the total execution time of applications.

In this context, to correctly apply DVFS techniques, it is good to have a clear notion of the concepts we reviewed in this chapter. It is also important to know the variables

involved in the measurement of energy consumption in modern processors architectures, since they are directly related to the determination of the energy efficiency of systems. On the other hand, research in this field focuses on power demand separately of load balancing strategies.

To correlate the state of the art in DVFS strategies presented in this chapter this with their area involved, we organized the Table 3.1. In this comparison, we depicted where each strategy was employed according to their main focus.

Table 3.1: DVFS strategies comparison in terms of applicability.

Author	Idle resource	Energy budget	Communication periods	Simulator/Kernel governor	Performance	Trace/ prediction
Hosseinimotlagh *et al.*	•				•	
Younge *et al.*	•				•	
Lee *et al.*	•			•		
Springer *et al.*		•			•	•
Isci *et al.*		•			•	
Hernandez *et al.*		•			•	
Rountree *et al.*		•			•	
Lim *et al.*			•			
Tan *et al.*			•			
Spiliopoulos *et al.*				•		
Semeraro *et al.*				•		
Gerards *et al.*				•		
Kim *et al.*				•	•	•
Freeh *et al.*				•	•	•
Raghu *et al.*					•	•
Peraza *et al.*					•	•

Source: The author

Some these presented approaches scaling down the frequency when the system is detected to be idle, in order to save energy. Works like the one proposed by Peraza *et al.* use an offline instrumentation of DVFS. First, it identifies the characteristics of each section of the applications. Then, in a second stage, it instruments the application with DVFS. Freeh *et al.* also proposes a static framework execute a single application at different processor frequencies and in a second execution choose the best frequency for each phase aiming saving energy. Other works have developed new governors to be loaded on kernel systems. These governors are developed based on a premise of ondemand or powersave governors.

Other works employ DVFS aiming to meet an energy budget, while that our proposed load balancers aim to reduce the average power demand during the execution of imbalanced applications on parallel machines. Works of Springer, Similarly, Isci *et al.* using DVFS to change the frequency of processors in order to maintain the power demand below a specified budget, while ENERGYLB combine load balancing and DVFS to compute the residual load imbalance among cores, after the execution of the load bal-

ancer. It adjusts the clock rate of underloaded cores and increases that of overloaded cores during the execution. Our proposed load balancers are also different from Hernandez *et al.* research, since they use a thread mapping strategy that considers performance and energy consumption constraints.

Based on these research, the first highlight with respect the power and energy consumption of this thesis is that current kernels governors do not recognize the load imbalance among cores used by a parallel application. Therefore, these issues could be explored with the use of DVFS techniques to provide significant energy savings without performance loss.

Another point is that DVFS strategies generally have been used separate of load balancing strategies. Several works have used DVFS, however this technique may cause performance degradation with an increase in the total execution time of parallel applications, which consequently increases the total energy consumption.

In this way, differently from other related works, we propose a dynamic energy-aware approach that focuses on increasing application's performance and reduce the average power demand. To implement this, we use both strategies, dynamic load balancing and DVFS together, which will be discussed in the next chapter.

4 PROPOSED ENERGY-AWARE LOAD BALANCERS

Parallel scientific applications allowed big advances in science. Scientific applications, however, demand an ever increasing performance of HPC systems. The vast majority of these applications have an irregular computational load and tasks with different processing demands. This makes the load distribution among the processors difficult. In this context, different approaches have been used to mitigate these imbalanced workloads and achieve an efficient use of all available parallel resources. However, most of these load balancing strategies focus only on reducing the execution time. Only a few recent strategies began to use the power demand and energy consumption information in their decision making process. Based on this premise and in the need to search for alternatives to reduce power demand without decreasing computing rate, **our thesis focuses on reducing the power demand and total energy consumption of imbalanced applications through a combination of *Dynamic Load Balancing* and *Dynamic Voltage and Frequency Scaling* (DVFS) without resulting in performance degradation**.

After an overview and an evaluation of the related works, and considering i) the application behavior; ii) the applications load imbalance; iii) the residual imbalance after load balancing calls; and iv) the power demand of processors in parallel machines; we propose a new energy-aware load balancing strategy named **ENERGYLB** (PADOIN et al., 2014). This load balancer strategy recognizes the irregular load of applications and has the best energy-performance tradeoff than state-of-the-art algorithms. It maintains the original execution time of the application while reducing the average power demand of the system during its execution, reducing the total energy consumption.

The remaining sections of this chapter are organized as follows. The main concepts of our energy-aware load balancer are presented in Section 4.1. The details of the centralized load balancing algorithm are discussed in Section 4.1.1. In Section 4.1.2, we detail the hierarchical algorithm and the approach used for its implementation. Following this, in Section 4.2 we describe implementation details and the parallel programming framework that supports the development of our proposed energy-aware strategies. Finally, we conclude this chapter, analyzing and presenting some considerations about the proposed ENERGYLB load balancer.

4.1 EnergyLB: Energy-Aware Load Balancing

In several HPC systems, the processor is the hardware component that has the highest power demand. When imbalanced applications are executed on parallel platforms, processors or cores in charge of lightweight tasks finish first, but remain consuming energy without performing any actual work for the application. In this context, load balancer strategies have been applied to reduce load imbalance on several parallel applications. However, even after load balancing, residual imbalances may still remain, leaving room for energy consumption improvements. Modern processors implement DVFS mechanisms, through which energy consumption may be reduced by reducing the processor's frequency and voltage. This mechanism is based in the fact that power demand is a quadratic function of the voltage, while the speed is a linear function (KRISHNA; LEE, 2000). In this case, energy savings can be achieved by decreasing the clock frequency of the underloaded processors or cores, through DVFS, in such a way that they can still end their tasks before or at the same time of the most loaded ones.

As discussed in Section 3.3, DVFS can be implemented, mainly, at two levels of granularity. Some processors feature *per-chip DVFS*, using the same power delivery network to reach every core, and consequently, bind each core to the same DVFS schedule. On the other hand, other processor models allow *per-core DVFS*, which uses a separate voltage regulator for each core, therefore allowing every core to have an independent DVFS schedule.

Taking these two levels of granularity in mind, we propose two versions of our energy-aware load balancer. The first version, called ***Fine-Grained EnergyLB*** **(FG-ENERGYLB)**, is suitable for platforms composed of few tens of cores that allow per-core DVFS. The second one, called ***Coarse-Grained EnergyLB*** **(CG-ENERGYLB)**, is suitable for current HPC platforms composed of several multicore processors that feature per-chip DVFS (PADOIN et al., 2014). Both ENERGYLB versions allow the selection of a specific load balancer to make load balancing decisions. At each load balancing round, our strategy decides whether to execute the load balancer or apply DVFS to reduce the power demand of underloaded cores.

In next two subsections, we present both load balancing approaches we are proposing. In their descriptions, we have adopted the notation described in Table 4.1, which shows the used variables and their descriptions.

Table 4.1: Description of variables used in algorithms.

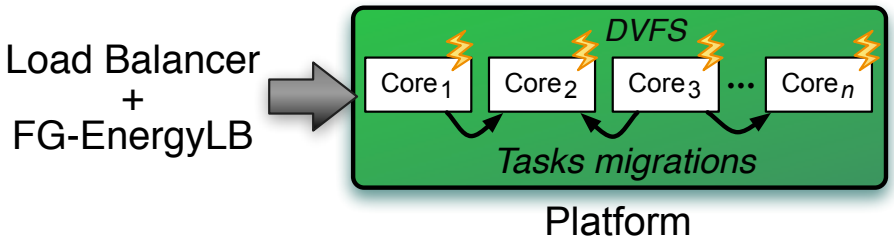
Variable	Description
p	individual processor
c	individual core
lb	load balancer
$load$	current load of each core
$freq$	current frequency of each core
$Wload$	weighted load (load / frequency)
min_Wload	lowest weighted load
max_Wload	highest weighted load
min_freq	lowest frequency
max_freq	highest frequency
$thrld$	threshold
$tasks$	total number of application tasks

Source: The author

4.1.1 FG-EnergyLB: Fine-Grained EnergyLB

FG-EnergyLB is a centralized energy-aware load balancer that considers a flat view of the underlying platform, balancing the load among all cores and performing DVFS on each individual core (Figure 4.1). FG-EnergyLB aims at mitigating load imbalance and also tries to save energy by reducing power demand. When the load imbalance is greater than range of clock frequency of the processors used, the use of the FG-EnergyLB alone is not able to avoid the total imbalance. Therefore, it relies on currently available load balancers to improve the task distribution among cores. On the other hand, when migrating tasks is not beneficial, it adjusts the clock frequency of underloaded cores, individually, to save energy. Since it only performs DVFS in these cases, it does not considerably impact the overall performance of the system.

Figure 4.1: Overview of the fine-grained algorithm.



Source: The author

As discussed in Section 2.2.1, most current strategies have used heuristics together with thresholds as a solution to decrease the load balancing overhead. In this way, these strategies adopt a threshold value to determine when load balancing must be performed or

not. In other words, the load balancer is only executed when load imbalance overcomes the threshold value.

Considering this issue, our strategy also adopts a threshold. FG-ENERGYLB takes into account both application load imbalance and processors power demand (clock frequency) to compute load imbalance and decide, on runtime, whether to call the load balancer or to perform DVFS. So, in each call of the ENERGYLB, it verifies if the weighted load of each processor exceeds or not the threshold, making decisions to adjust the frequencies (determining so that the frequency will be decreased or increased) or invoke other load balancer to migrate tasks.

This functionality is only allow to current processor that implement different clock frequency levels. This range of levels allow to users to vary the clock frequency of the processor from low level (power save) up to high level (power hungry), as discussed in Section 3. In this way, FG-ENERGYLB is able to save energy through the reduction of the power demand during the execution, as discussed in following.

4.1.1.1 FG-ENERGYLB *Algorithm*

The main steps performed by FG-ENERGYLB are described in Algorithm 1, which takes three input parameters. The first one, *lb*, is the load balancer that should be used when the imbalance is high. The second one, *max_freq*, is the maximum frequency that can be set to a core. The third one, *thrld*, is used as a *threshold* to decide whether FG-ENERGYLB will call the load balancer or perform DVFS.

At each load balancing step, FG-ENERGYLB starts by gathering the current load (*load*) and clock frequency (*freq*) of each core *c* (lines 2–3). The load of a core *c* is determined by the sum of the execution times of the tasks currently assigned to it since the last load balancing step.

Having completed this phase, it calculates the weighted load (*Wload[c]*) of each core *c* (line 4). This is necessary since the time needed to compute the tasks assigned to a core will depend on its current clock frequency. Then, FG-ENERGYLB determines the minimum (*min_Wload*) and maximum (*max_Wload*) weighted loads of all cores (lines 6–7).

Finally, it decides whether it is necessary to call a load balancer *lb* to migrate tasks among the cores or not. This decision is taken based on the threshold *thrld*. FG-ENERGYLB considers that the load imbalance is high if the ratio between *max_Wload* and *min_Wload* is equal or greater than *thrld*, thus calling the load balancer (line 9). This

Algorithm 1: FG-ENERGYLB's algorithm.

input : A load balancer *lb*, the maximum frequency *max_freq* and a threshold *thrld*.
output: A new task distribution or the frequency of each core *c* adjusted.

```
1  foreach core c do
2  |  load ← getLoad(c)
3  |  freq ← getFrequency(c)
4  |  Wload[c] ← load / freq
5  end
6  min_Wload ← min(Wload)
7  max_Wload ← max(Wload)
8  if (max_Wload / min_Wload ≥ thrld) then
9  |  call(lb)
10 else
11 |  foreach core c do
12 |  |  freq ← round (Wload[c] / max_Wload ∗ max_freq)
13 |  |  setFrequency(c, freq)
14 |  end
15 end
```

way, ENERGYLB benefits from other available load balancers. In this case, reducing only the frequency would not be enough to balance the load, so calling other load balancing strategies helps by migrating tasks. Otherwise, it adjusts the clock frequencies of the cores according to their *relative loads*, i.e. their loads in relation to more overload core (lines 11–14) to save energy if residual imbalance is present.

By updating the frequency of the cores according to their relative loads, ENERGYLB reduces the average power demand of the system and, consequently, saves energy. This way, using this approach does not impact on the overall execution time of the application.

4.1.1.2 FG-ENERGYLB *Algorithm Properties*

FG-ENERGYLB aims to adjust the clock frequency of underloaded processors through DVFS when weighted loads are less than a *threshold*, otherwise, the strategy calls a load balancer. The execution of this load balancing can incur in an overhead to application. This occurs when the load balancer is called too frequently, or the time it takes to make load balancing decisions exceeds its benefit. This way, is important to determine when and how frequently to call the load balancer.

Strategies such as GREEDYLB, GREEDYCOMMLB, REFINELB, REFINECOMMLB,

and TEMPLDB use a common practice of calling the load balancer periodically. The period between calls is specified by the user. Other strategies, such as METATEMPCONTROLLER, decide automatically when to perform load balancing, based on information from the runtime system.

Our energy-aware adopts a similar strategy to the first group, where the load balancing calls are periodic. However, at each call, it decides whether to run the actual load balancer or to perform DVFS adjustments. This decision is based upon the minimum and maximum weighted loads computed for each core. The ratio between them is compared to the threshold to decide whether to call the load balancer or to perform DVFS on each processor according to its load. Therefore, the amount of the energy saving depends on the irregularity of the application's load and on the threshold parameter. This threshold can be *aggressive*, calling the load balancer more often or *moderate*, performing less task migrations and more DVFS, as will be discussed in Subsection 5.2.2.2.

For a system with c cores, FG-ENERGYLB presents a complexity of $O(c)$ in the worst case, where *threshold* is never achieved. Otherwise, when weighted loads are more than the *threshold*, FG-ENERGYLB presents a complexity of $O(max(c, lb))$, where $O(lb)$ is the complexity of the selected load balancer.

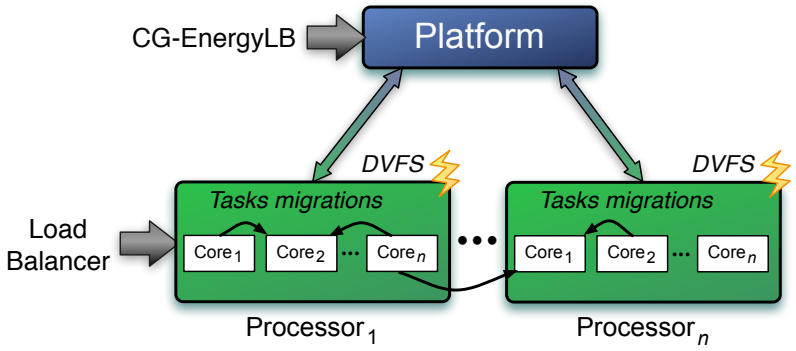
4.1.2 CG-ENERGYLB: Coarse-Grained EnergyLB

CG-ENERGYLB is a hierarchical algorithm of our energy-aware load balancer, being so suitable for HPC platforms composed of several multicore processors that feature per-chip DVFS. The main difference of this algorithm is that it considers a hierarchical view of the underlying platform, which is represented as a two-level tree, as shown in Figure 4.2. In this scheme, the platform (*root*) is composed of several multicore processors (*leaves*). CG-ENERGYLB performs load balancing to distribute the load among the processor's cores inside the leaves and between the leaves. In the root level, on the other hand, it adjusts the frequency of each multicore processor according to its relative load to save energy by exploiting residual imbalances.

4.1.2.1 CG-ENERGYLB *Algorithm*

Algorithm 2 describes the overall steps performed by CG-ENERGYLB. Basically, it takes the same input parameters of FG-ENERGYLB. However, weighted loads

Figure 4.2: Overview of the coarse-grained algorithm.



Source: The author

of cores are now aggregated by processor (lines 1–7). The weighted load of a processor p (*Wload[p]*) is determined by the sum of the weighted loads of its cores c. We divide the load of the core by the current frequency of the processor to compute the weighted load of each core of the same processor, since all cores of the same processor share the same clock frequency.

Algorithm 2: CG-EnergyLB's algorithm.

input : A load balancer *lb*, the maximum frequency *max_freq* and a threshold *thrld*.
output: A new task distribution or the frequency of each processor p adjusted.

```
1  foreach processor p do
2  |  freq ← getFrequency(p)
3  |  foreach core c in processor p do
4  |  |  load ← getLoad(c)
5  |  |  Wload[p] ← Wload[p] + load / freq
6  |  end
7  end
8  min_Wload ← min(Wload)
9  max_Wload ← max(Wload)
10 if (max_Wload / min_Wload ≥ thrld) then
11 |  call(lb)
12 else
13 |  foreach processor p do
14 |  |  freq ← round (Wload[p] / max_Wload ∗ max_freq)
15 |  |  setFrequency(p, freq)
16 |  end
17 end
```

The rest of the algorithm is very similar to Algorithm 1. The minimum and maximum weighted loads are computed and the ratio between them is compared to the threshold *thrld* to decide whether to call load balancer or to perform DVFS on each processor according to its load.

4.1.2.2 CG-ENERGYLB *Algorithm Properties*

CG-ENERGYLB weights the loads of processors p, adding the load of all its cores c. Thus, CG-ENERGYLB presents a complexity of $O(p * c)$ in the worst case where *threshold* is not achieved. Otherwise, when weighted loads are more than a *threshold*, FG-ENERGYLB presents a complexity of $O(max((p * c), lb))$, where $O(lb)$ is the complexity of the selected load balancer.

4.2 Implementation Details

Currently, several parallel programming environments provide support for the development of strategies for load balancing. Some of these environments also provide load balancing information that can be obtained and used at application runtime. However, few tools are able to evaluate power demand and consequently the energy consumption of each processor at runtime of a parallel program.

In this context, for the implementation of new load balancing strategies it is necessary to choose an environment/framework for parallel programming. Currently, there are tools developed with specific characteristics for each type of environment that aims to facilitate the development and productivity. However, to achieve the best performance possible, it is necessary that the scheduler knows all the details of the architecture available, as well as the specificities of the application, to then make its decisions.

Most load balancing approaches focus on mapping the tasks and their data, seeking a best load distribution between the processors in order to avoid imbalance and reduce the latency. The implementation of these approaches are usually linked to the characteristics of the parallel programming environment. Among the options available, we can highlight the following parallel programming environment: Anahy-3 (CAVALHEIRO et al., 2007), CHARM++ (KALÉ; KRISHNAN, 1993; KALÉ et al., 2008; CHARM++, 2014; MEI et al., 2010), sofa (FAURE et al., 2007), XKAAPI (TCHIBOUKDJIAN et al., 2011; GAUTIER; BESSERON; PIGEON, 2007) and UPC (HOFMEYR et al., 2011).

In the context of this thesis, we demonstrate our proposed load balancing approach within an environment. We provide an implementation of our proposed strategies of ENERGYLB in CHARM++ runtime system, which is discussed in the next subsection.

4.2.1 CHARM++ Parallel Programming

CHARM++ is a parallel programming model based on the C++ language. It was developed with the goal of improving the productivity of parallel programming through a high level abstraction of parallel computing while providing good performance over platforms based on shared and distributed memory (KALÉ; KRISHNAN, 1993; CHARM++, 2014). CHARM++ programs are not implemented to physical cores. Programs are decomposed into interconnected parallel objects called *chares*, *tasks* or *work units* that are initially mapped onto processors using a default mapping and can be migrated from one processor to another during program execution by CHARM++ runtime system (RTS).

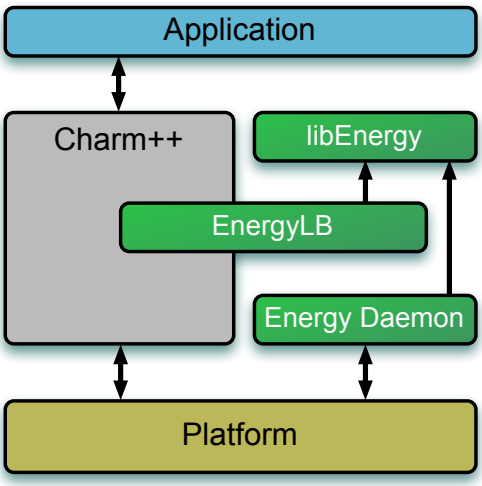
Parallel CHARM++ applications use an interface description language to describe their objects. Using such objects, the programmer describes computation and communication interaction among them. All messages generated from these interactions are managed by CHARM++, using a remote method invocation in a message-driven model.

The CHARM++ platform has an interface that captures the statistics of the tasks, including their computation and communication during the execution of the application. These measurements are stored in a database and can be used to make decisions and to improve the load balancing (BHATELÉ; KALÉ; KUMAR, 2009; CHARM++, 2014). This platform also provides a simple API that can be used to implement new load balancers without changing the source code of the applications. In this context, load balancing in CHARM++ is measurement-based and depends on instrumented data from previous time steps to balance the application workload for future time steps. The instrumented data can be dynamically obtained during the application execution through the use of the load balancing API.

The main reason that led us to choose CHARM++ was that it features a mature load balancing framework and has presented good efficiency for a large class applications, such as NAMD, ChaNGa, *Lulesh*, *Lassen* and others. A second reason is because CHARM++ provides a set of load balancing strategies that make decisions and can produce a new tasks mapping through of task migration between processors at runtime (KALÉ; KRISHNAN, 1993). So, programs developed in CHARM++ can use their own load balancing strategy or chose one of the available in CHARM++ that is best for their execution. However, our proposed energy-aware load balancing can be implemented in any other parallel programming framework that provides task migration.

We have implemented two algorithms of ENERGYLB as CHARM++ load bal-

Figure 4.3: ENERGYLB implementation.



Source: The author

ancers which are discussed in the (Subsections 4.1.1 and 4.1.2).

4.2.2 Energy Modules

In addition, we also have implemented two other modules (Figure 4.3). The first is an *Energy Daemon* (EMonDaemon) that is responsible for gathering power and energy measurements from the underlying platform with a periodicity defined by the user.

This module was developed since correlating performance with power and energy consumption during runtime can be difficult, since current platforms have different interfaces to collect information from all their components. Moreover, some of the existing tools provide data to be analyzed only after execution. Researchers have been using the energy consumption of the whole system for their evaluations due to the difficulty of isolating the power demand of the processors. Although new processors feature sensors that enable precise measurements, they provide different interfaces to collect data, making it difficult to correlate runtime with power demand and energy consumption.

To overcome this issue, in the context of this thesis, we developed a platform-independent tool that collects power and energy data from homogeneous and heterogeneous systems (PADOIN et al., 2015). Using the data provided by our proposed tool, we are able to plot the instantaneous power demand in each processor during the execution of each test, through the use of model-specific registers (MSR) available on Intel Sandy Bridge processors, or specific registers of AMD and ARM processors. Beyond of collect the instantaneous power demand of each processor, it also compute the energy spend and

the execution time of each application computed.

The second is *LibEnergy*, a library that offers an interface to get system information such as the maximum frequency that can be set to a core or processor, to set the clock frequency of individual processors/cores or to set a specific Linux governor. It relies on the Linux kernel infrastructure called *cpufreq* to detect and set the frequency level of individual processors at runtime. The *cpufreq* module provides a common interface to the various low-level, CPU-specific frequency-control technologies and high-level CPU frequency-controlling policies. This makes our energy-aware load balancer portable across different processors and Linux distributions.

4.2.3 ENERGYLB Phases

The energy-aware strategy adopted by ENERGYLB consists of two main phases:

I - Initialization phase - This first phase is performed only once when ENERGYLB is initialized and involves capturing information from the system configuration. This includes information about the processors' model and manufacturer, which is gathered because the energy consumption measured by the EMonDaemon comes from performance counters. According to processor model, it uses different approaches to collect power and energy.

ENERGYLB gets from kernel the following information:

- processor model - check the manufacturer processor (Intel/AMD/ARM) of the systems nodes. Only recent processor models have registers with energy information. e.g. Intel Sandy, Ivy and Haswell microarchitecture (ROTEM et al., 2012), AMD Bulldozer of family 15h microarchitecture and (AMD, 2013) ARM Cortex A15, so, this feature allows the use of ENERGYLB in systems with heterogeneous processors;
- clock frequencies - get the available clock frequencies of each processor to define the frequency adjustment limits available in the platform;

The information of tasks on each processor are acquired through the LBDatabase that CHARM++ provides. The ENERGYLB gets from CHARM++ the following information:

- total number of application tasks;

- initial current task mapping;

II - Work phase - The second phase is periodic and it happens at each load balancing call done by the application. In this phase, ENERGYLB collects load information for each task on every core/processor from CHARM++ LBDatabase. This information includes:

- current load;
- current clock frequency;

After that, ENERGYLB weighs the core loads according to their current load and clock frequencies. When load imbalance is detected, the difference between the most and least loaded cores is compared to the available clock frequency limits. If the load difference is greater, then we benefit from the execution time-focused load balancers available with CHARM++ (ZHENG et al., 2011). Otherwise, ENERGYLB corrects the clock frequencies of the different cores through DVFS according to their current loads by increasing the frequency of overloaded cores, and reducing the frequency of underloaded ones.

In this way, using our approach, the average power demanded by the system of an electric company is reduced, saving energy, without impact on the total execution time of applications.

4.3 Discussion

Parallel applications have been optimized for several years to achieve better performance. However, achieving energy savings on these parallel applications without degrading their performance is today an increased challenge. In this context, several load balancing strategies provide a set of information about application and allow to migrate task between processors to avoid load imbalance.

However, despite the existence of several load balancers available, most of them are designed for performance optimization and aim to reduce the execution time only, not taking into account the energy consumption characteristics or power demand of system. At the same time, when a system runs load imbalanced applications, cores with shorter tasks some cores can finish their tasks before others and remain consuming or wasting energy without doing any actual work for the application.

In this chapter, we presented our research that aims to fill this gap, the residual

imbalance after load balancing. We propose and implement a new load balancing system aiming to reduce the total energy consumption and avoiding migration of tasks through the use of DVFS techniques.

As opposed to other related works, our approach performs load balancing along with DVFS to improve the performance and to reduce the energy consumption by exploiting residual imbalances of parallel applications. Different from other works, our ENERGYLB approach uses DVFS to change the clock frequency of the processors according with their relative load to avoid imbalance instead to migrate tasks among the processors. In this way, this approach of ENERGYLB tries to reduce also the energy consumption with data and task migration.

ENERGYLB approach also differs from other strategies that make decision only at the end of one run. These approaches require a previous execution to analyze the imbalance to then make decisions. From this premise, to save energy without losing performance our periodic load balancer can be used with load imbalanced applications, addressing the computational load and making decisions during the runtime and not when execution ends like some load balancers.

Some related approaches have been used to avoid core overheating on HPC systems. Our approach differs from these works, as presented by Sarrod and Menon. The ENERGYLB algorithms employ DVFS as a way to decrease energy consumption after balancing the load while cited works use DVFS to regulate temperature of processors aim to reduce the energy spent in cooling.

Another difference from ENERGYLB to METATEMPCONTROLLER and TEMPLDB is that both approaches check the temperature and then if it exceeds the specified temperature, the load balancer uses DVFS to decrease the frequency by one level only. Our ENERGYLB approach weights the core loads according to their current clock frequencies, and when load imbalance is detected, the difference between the most and least loaded cores is compared to the available clock frequency limits, changing then the current clock frequency in several levels according to levels available on processor used.

ENERGYLB is also different from approach proposed by Aupy *et al.*. They developed a strategy that consider the failures on HPC system aiming fault-tolerance. This way, the approach work with re-execution of tasks while reducing the energy consumption, considering an expected energy consumption for a second execution. Our proposed energy-aware load balancers, FG-ENERGYLB and CG-ENERGYLB, consider both load (irregularity and dynamicity) of the application and current power demand (clock fre-

Table 4.2: Comparison of the proposed algorithms with the state of the art in terms of decision criteria.

Algorithm	Category*	Metrics				
		Load	Communication	Topology	Temperature	Energy
FG-EnergyLB	C	•				•
CG-EnergyLB	H	•				•
GreedyLB	C	•				
RefineLB	C	•				
GrapeVineLB	D	•				
GrapePlusLB	D	•				
Seyedmehdi *et al.*	C	•				
GreedyCommLB	C	•	•			
RefineCommLB	C	•	•			
TopoAwareLDB	C			•		
NucoLB	C	•	•	•		
HwTopoLB	C	•	•	•		
HierarchicalLB	H	•	•	•		
HybridLB	H	•	•			
TempLDB	C				•	
MetaTempController	C				•	
Merkel *et al.*	C				•	
Kodama *et al.*	C				•	
Goel *et al.*	C				•	
Hartog *et al.*	C				•	
Aupy *et al.*	C					•

*Category: (**C**)entralized (**D**)istributed (**H**)ierarchical

Source: The author

quency) of the platform to make its decisions, as shown in the Table 4.2. The centralized FG-EnergyLB and the hierarchical CG-EnergyLB can rely on currently available load balancers to better distribute the tasks among the cores when the load imbalance is high or adjust the clock frequency of underloaded cores individually when the load imbalance is low, defined by the threshold parameter. Both approaches used by load balancers save energy. In the first, reducing the execution time, and in the second, reducing the power demand of platform, which directly impacting on total energy consumption.

The main difference from EnergyLB proposal with Freeh *et al.* work is due the dynamicity. While Freeh *et al.* propose a static framework to execute an application and in a second execution choose the best frequency for each phase aiming saving energy, out approaches combines dynamic load balancing with DVFS techniques on iterative applications to improve the energy efficiency.

Our proposed algorithms try to reduce the total energy consumption by exploiting residual imbalances of applications without considerably impacting the overall system performance. To implement this, we combine two approaches, Dynamic Load Balancing and DVFS. We evaluate our proposed energy-aware load balancers with a set of benchmarks and real world applications in the next chapter.

5 LOAD BALANCERS EVALUATION

In this chapter, we validate our proposed energy-aware load balancer presented in the previous chapter. We present a runtime evaluation, power demand reduction and the energy savings when the ENERGYLB load balancer is used on an experimental platform.

First, in Section 5.1 we present the methodology used to evaluate our strategy. The experimental results are organized into the two sections after that. In Section 5.2, we show the results measured using the FG-ENERGYLB, our centralized approach over benchmarks and real world applications, and lastly, in Section 5.3 we evaluate our hierarchical CG-ENERGYLB with real world applications.

5.1 Evaluation Methodology

Our methodology is organized as follows. First, we describe the execution environment used in tests. In following, we detail the CHARM++ benchmarks, real world applications and CHARM++ load balancers selected to evaluate our proposed energy-aware load balancer. At the end, we present a discussion about experimental results.

5.1.1 Experimental Environment

Our experiments are conducted on an Altix UV 2000 platform designed by SGI. The platform is composed of 24 NUMA nodes. Each node has an Intel Xeon E5-4640 Sandy Bridge-EP x86-64 processor with 8 physical cores. There are 14 clock frequency levels available in these processors. This range of levels allow us to vary the clock frequency of the processor from 1.2 GHz, the lowest frequency and more power save, up to 2.4 GHz, the highest frequency and more power hungry. Considering the power demand measured to each processor of our parallel platform, when only one core is being used, reducing the clock frequency from the maximum to the minimum level we can reduce power demand from 36.3 W to 20.4 W on average, which represents a reduction of 43%.

Each core of the Intel Xeon E5-4640 has 32 KB instruction and 32 KB data L1 caches and 256 KB of L2 cache. All the 8 cores share a 20 MB L3 cache. Each node has 32 GB of DDR3 memory, which is shared with other nodes in a cc-NUMA fashion through SGI's proprietary NUMAlink6. Overall, this platform has 192 physical cores and

768 GB DDR3 memory.

The platform runs an unmodified SUSE Linux Enterprise Server operating system with kernel 3.0.101-0.29. All applications, as well as the CHARM++ programming model were compiled with GCC 4.8.2. The CHARM++ version used in our experiments was multicore-linux $64 - 6.5.1$. The results presented in this chapter are the average of at least 20 runs. The relative error was less than 5% using a 95% statistical confidence by Student's t-distribution.

5.1.2 Benchmarks and Irregular Applications

For the runtime evaluation, power demand and energy consumption of our proposed ENERGYLB, we have used the CHARM++ environment. As described in Subsection 4.2.1, CHARM++ provides a set of benchmarks, which we used to compare our energy-aware load balancer. We selected four benchmarks and three real world applications. These benchmarks and real world applications were chosen due to their varied range of communication patterns and workload characteristics, which are describe as follows.

We have used from CHARM++ the *lb_test*, *kNeighbor*, *stencil 4D* and *ComprehensiveBench* benchmarks. The description of the benchmarks is given in following:

- ***lb_test*** is a synthetic benchmark that provides a scenario where it is possible to create an amount of distributed process, to control load imbalance, and also to use different communication patterns among the processors. To perform the tests, we have used a random communication graph (KALÉ; KRISHNAN, 1993; CHARM++, 2014);
- ***kNeighbor*** is an iterative micro-benchmark with an intensive and a near-neighbor communication pattern. During the execution, each task communicates with k other neighbor tasks at each iteration, exchanging 16 KB sized messages. In our tests, we used the communication pattern ring (KALÉ; KRISHNAN, 1993);
- ***stencil 4D*** is a balanced four-dimensional stencil computation. It uses a four-dimensional mesh to represent its communication pattern (KALÉ; KRISHNAN, 1993; CHARM++, 2014); and
- ***ComprehensiveBench*** is a highly configurable synthetic benchmark that can be used to create iterative applications according to different parameters such as the

number of tasks, iterations, communication graph, task loads and message sizes (PILLA et al., 2015).

We also evaluate the benefits of our energy-aware load balancers with real world applications. The description of the applications is given in following:

- ***Ondes3D*** is a seismic wave propagation simulator employed to estimate the damage in future earthquake scenarios (DUPROS et al., 2008), as presented in Subsection 2.1.2. In *Ondes3D*, seismic waves are modeled as a set of elastodynamics equations. These equations are then solved by applying a finite difference method. In our experiments, we used a version recently adapted to Adaptive MPI (HUANG; LAWLOR; KALÉ, 2004; KALÉ et al., 2008) that profits from CHARM++'s load balancing framework (TESSER et al., 2014a). In this version, the application is over decomposed into multiple virtual MPI processes per core. *Ondes3D* presents load irregularity due to the boundary conditions producing additional work, and load dynamicity from the simulation of waves spreading through space;
- ***Lulesh*** simulates a variety of science and engineering problems require modeling hydrodynamics, which describes the motion of materials relative to each other when subject to forces. The Livermore Unstructured Lagrange Explicit Shock Hydrodynamics (LULESH) application was originally developed as one of the five challenge problems in the DARPA Ubiquitous High Performance Computing (UHPC) program. *Lulesh* solves one octant of the spherical Sedov problem using Lagrange hydrodynamics (KARLIN et al., 2013; KARLIN et al., 2012; DOSANJH et al., 2014); and
- ***Lassen*** is a mini-application that explores applications with varying load balance. It models and simulates the propagation of a wave through as it travels around an unstructured mesh and it works on a 2D and 3D version. The computational load of the mesh is sub-divided into domains, which are assigned to processors. Therefore, this becomes a challenging problem to effectively parallelize since the workload is constantly changing (MCCANDLESS, 2015).

While some benchmarks and applications start its execution in an imbalanced state, others have a balanced state at the start with a dynamic behavior during its execution, as discussed in Section 2.1.1. Table 5.1 summarizes the imbalance type and initial state of the benchmarks and real world applications selected for the load balancers evaluation.

Table 5.1: Benchmarks and real world applications characteristics.

Application	Imbalance Type	Initial State
lb_test	load	imbalanced
kNeighbor	communication	balanced
stencil 4D	load	imbalanced
ComprehensiveBench	load/communication	imbalanced
Ondes3D	load	imbalanced
Lulesh	load/communication	imbalanced
Lassen	load	imbalanced

Source: The author

5.1.3 Load Balancers

As presented in Subsection 2.2.1, today there are several load balancers available to avoid load imbalance in parallel applications. In this context, the CHARM++ platform provides a set of load balancing techniques that can be used to migrate tasks among processors and to reduce the load imbalance. Thus, we validated our energy-aware load balancers comparing the achieved results with different standard load balancers available in CHARM++. To compare the execution time, power demand and energy consumption results, we have selected the load balancers GREEDYLB and GREEDYCOMMLB that have a greedy algorithm, REFINELB and REFINECOMMLB that use a limit of task migrations, RANDCENTLB that randomly assigns objects to processors and SCOTCHLB that map based on the recursive bi partitioning using both the source process graph and the target architecture graph.

5.1.4 Power and Energy Monitoring

To perform our tests, we use our EMonDaemon on the experimental platform. This module allows us collect runtime information and also store in a file with a periodicity defined by the user (PADOIN et al., 2015). In this case, to the execution of experiments, we have configured a measure window of 1 second. This value is used since lower frequency values would incur in a high overhead to the application.

5.1.5 Test Execution Details

Different load balancing frequencies have been chosen and tested for different applications in order to achieve a balance between the benefits of remapping tasks and the overhead of moving tasks and computing a new task mapping. As discussed in Section 2.2, the interval to call the load balancer is decisive to reduce the load imbalance, however if the strategy is performed very frequently, it may incur a high overhead. We have validated our ENERGYLB approach using different configurations, however to decide the optimal moment to call a load balancer is a challenging problem (MENON et al., 2012) and is out of the scope of our research. Table 5.2 summarizes the configurations of the applications and parameters used in our experiments in this thesis. These input parameters are similar to the used in ones tests of the Parallel Programming Laboratory.

Table 5.2: Summary of the input parameters of benchmarks and real applications.

Application	**Tasks**	**Iterations**	**LB Frequency (Iterations)**
lb_test	200	150	10
kNeighbor	1600	50	10
stencil 4D	4096	50	10
ComprehensiveBench	55	50	5
Ondes3D	128	500	20
Lulesh	729	1000	50
Lassen	256	550	50

Source: The author

Our energy-aware load balancers use three input parameters, as described in Algorithms 1 and 2 of Chapter 4. We have conducted the validation tests setting the first one, *lb*, with GREEDYLB, GREEDYCOMMLB, REFINELB, REFINECOMMLB, RANDCENTLB and SCOTCHLB load balancers. The second one, *max_freq*, with the maximum frequency available on selected processor. We have tested different values for the third one, *thrld*. Based on the clock frequency range available on our experimental platform, we have used a threshold value of 2. This value was used once that processors allow us to vary the clock frequency of the processor from 1.2 GHz up to 2.4 GHz. Tests using different value for threshold also were evaluated, which are discussed in Subsections 5.2.2.2 and 5.3.1.2

The results are organized to compare the total energy consumption, followed by total execution time and the power demand of each benchmark, when executed in four scenarios: (i) without load balancer, which is defined as baseline, since no task is mi-

grated; (ii) using only the FG-ENERGYLB to adjust the frequency of the cores according to their loads; (iii) using only the standard load balancers available in CHARM++; and (iv) using FG-ENERGYLB along with the standard load balancers.

Table 5.3 presents the list of load balancers and their abbreviations used in the presentation of results.

Table 5.3: List of load balancers abbreviations used in the presentation of results.

Load Balancer	**Abbreviations**
without load balancer	NOLB
CG-ENERGYLB	CG
FG-ENERGYLB	FG
GREEDYLB	G
GREEDYCOMMLB	GC
REFINELB	R
REFINECOMMLB	RC
RANDCENTLB	Ra
SCOTCHLB	S

Source: The author

5.2 FG-ENERGYLB Evaluation

The first part of our evaluation presents the benefits of FG-ENERGYLB, our centralized approach over benchmarks and real world applications presented in the Section 5.1. In the specific case of our experimental platform, although it is possible to set the frequency of each core individually, no power gains are achieved due to a shared voltage rail and clock source present in their processors. Because of that, we used a single core of each processor to simulate a scenario in which *per-core* DVFS is allowed. This was done by configuring the CHARM++ environment to use 24 processors and we performed a mapping of the processes such that only the first core of each processor is used.

We organize the FG-ENERGYLB evaluation in two parts. Firstly, we present the saving energy over benchmarks. This is followed by the energy improvements over real world applications. For test with real applications, we present also the percentage of energy spent on load balancing and the results achieved using different threshold values.

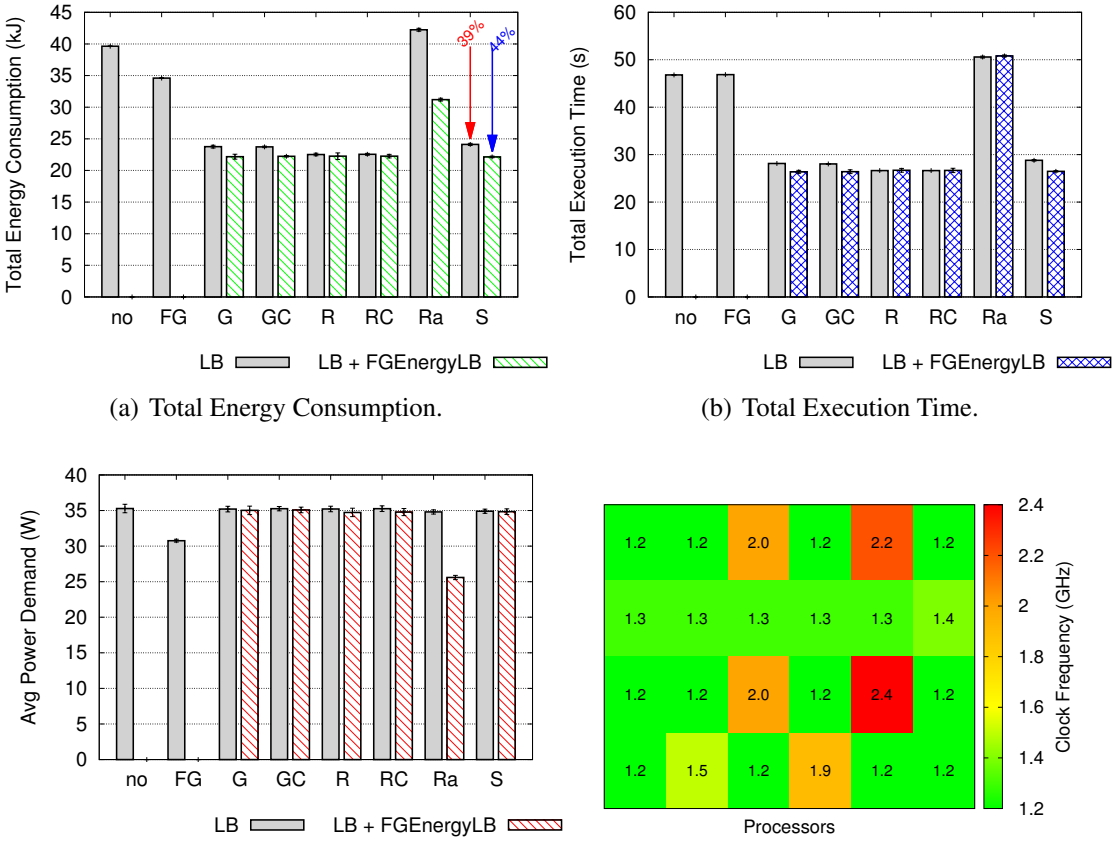
5.2.1 Evaluation on Benchmarks

This section presents the results achieved using the FG-EnergyLB over benchmarks. We provide a comparison between FG-EnergyLB that mitigates the residual imbalance and other selected load balancers. We start discussing the evaluation measured during the test executions of *lb_test* and *ComprehensiveBench* benchmarks. Due to reasons of space, the results of *kNeighbor* and *stencil 4D* benchmarks are presented in Appendix A.1.

First, we discuss the gains when FG-EnergyLB was used alone, and after, when FG-EnergyLB was employed over the residual imbalance of other load balancing algorithms, as discussed in following:

- Benchmark: *lb_test*

Figure 5.1: Evaluation of FG-EnergyLB with *lb_test* benchmark.



(a) Total Energy Consumption.

(b) Total Execution Time.

(c) Average Power Demand.

(d) Final Clock Frequency to FG-EnergyLB alone.

Source: The author

i) FG-EnergyLB Evaluation

As discussed in Section 5.1, the *lb_test* starts its execution with a large imbal-

ance. For this reason, FG-ENERGYLB is able to reduce by 13% (Figure 5.1(a)) the total energy consumption from 39.64 kJ to 34.60 kJ when employed alone on *lb_test*. As FG-ENERGYLB did not influence the execution time of the benchmark (Figure 5.1(b)), the energy saving is a result of the power reduction in the cores (from 35.28 W to 30.76 W on average - Figure 5.1(c)). This power reduction of 12.82% happens because the frequency of the cores was reduced during the runtime according to the load of each core in relation to the most overloaded core. In this way, the average frequency during the runtime was reduced from 2.4 GHz to 1.429 GHz.

When FG-ENERGYLB was used alone, the execution finished with 17 cores using a frequency reduced to 1.3 GHz or lower (Figure 5.1(d)), and only one core, the most loaded, remains with its maximum clock frequency.

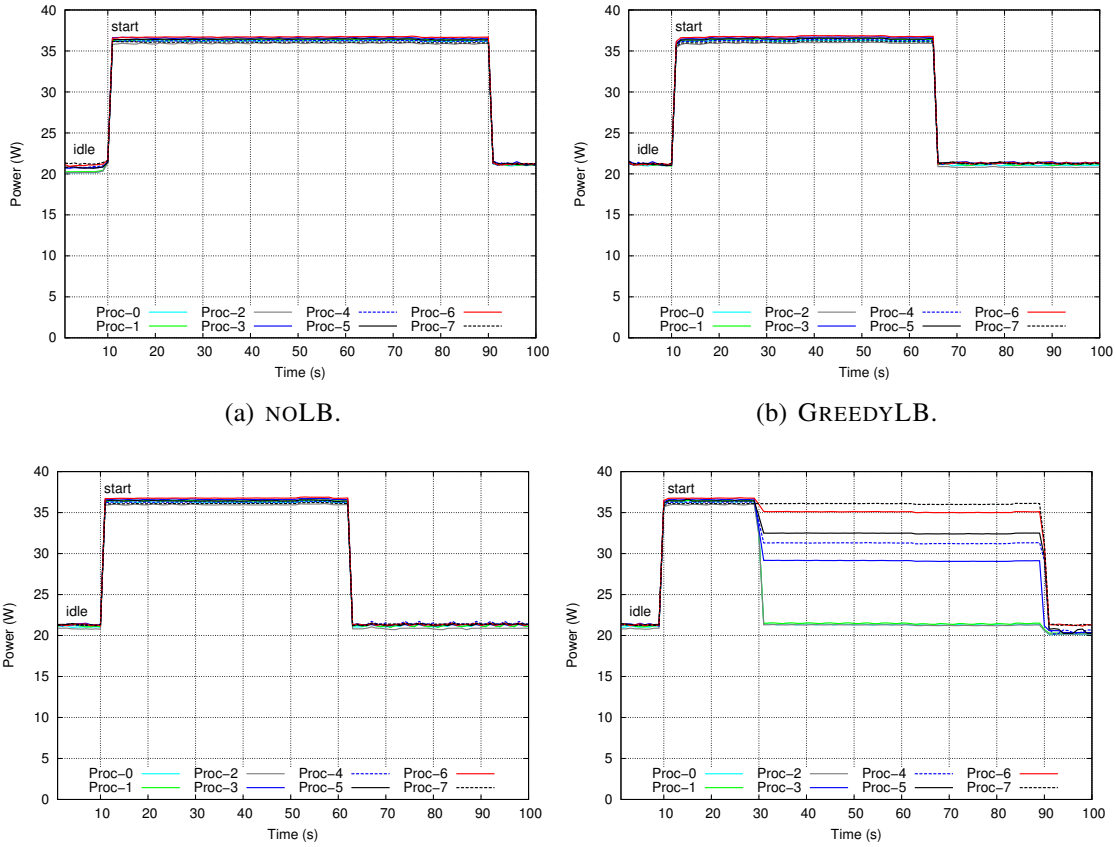
In this test, applying FG-ENERGYLB alone, we are evaluating only its DVFS functionality. To better analyze this functionality, we conducted a new test. *lb_test* was executed with 500 tasks, each task with 250 iterations on only 8 cores. The load balancing frequency was set to 10 iterations.

Figure 5.2 illustrates the instantaneous power demand measured during the test executions, without load balancer (NOLB) that represents a baseline execution, using different CHARM++ load balancers, GREEDYLB, REFINELB and our FG-ENERGYLB load balancer.

Figure 5.2(a) shows the power measured during the execution of benchmark without any load balancer strategy. As discussed in Section 3.3, when parallel load imbalanced applications are executed, the kernel does not realize the imbalance between processor used by applications. So, it runs the application using the maximum frequency available that maintains the power constant. In all processors, the power measured during the execution was at all times close to 36 W.

When the benchmark is executed with the GREEDYLB and REFINELB load balancers, these realize the imbalance load in the application, make their decisions and migrate task, reducing the execution time. However, with GREEDYLB and REFINELB, the application is also executed using the highest frequency available in each processor. In this form, the power demand measured in the GREEDYLB execution (Figures 5.2(b)) and REFINELB execution (Figures 5.2(c)) is constant and always above to 36 W, similarly to power demand of NOLB (Figure 5.2(a)). In this way, as all cores present a similar power demand during the execution and finish their tasks in a similar time, all cores have a similar energy consumption. On the other hand, some cores run tasks with less computational

Figure 5.2: Instantaneous power measured during execution of the imbalanced *lb_test* benchmark using different load balancers.



(a) NOLB. (b) GREEDYLB.

(c) REFINELB. (d) FG-ENERGYLB alone.

Source: The author

load, so the system is wasting energy.

We realize a different behavior on power demand of system when the *lb_test* benchmark is run with the FG-ENERGYLB load balancers (Figures 5.2(d)). Initially, all cores used have a power demand similar to NOLB, GREEDYLB and REFINELB since they are running their task using the maximum frequency available. However, when FG-ENERGYLB is called, only one core, which has the highest computational load, remains at 2.4 GHz and keeps the same power demand. All other cores have their clock frequencies reduced to intermediate or to the minimum frequency. Consequently, the instantaneous power demand of each processor is reduced according their computational load, reducing the total energy spent.

We can observe that, when the FG-ENERGYLB was executed the first time (at 20^{th} second), it changed the clock frequency of all cores according to their relative load. However, as this benchmark does not present variations in its load, in the other times that FG-ENERGYLB was called, no change in the clock frequency of cores was necessary.

Table 5.4 presents the execution time, energy consumption and average power de-

mand measured without load balancer (NOLB), with GREEDYLB, REFINELB and FG-ENERGYLB applied alone. GREEDYLB, REFINELB load balancers reduce the energy consumption of the *lb_test* benchmark by reducing its execution time. On the other hand, FG-ENERGYLB, with a small overhead (0.2%), is able to reduce the total energy consumption reducing the clock frequency of processors through of the reduction the average power demand of processors. The average power measured during NOLB, GREEDYLB, REFINELB executions was above to 36 W on average, while FG-ENERGYLB kept it at 30.53 W on average. Additionally, FG-ENERGYLB is able to reduce energy consumption by up to 16% (from 2883 J to 2430 J) when compared to NOLB. However, the use of FG-ENERGYLB alone spends more energy than GREEDYLB and REFINELB, since they load balancers were able to reduce their total execution time. In these tests, the difference between the total executions times achieved is less than 3% for all cases, which is considered to be inside the error margin.

Table 5.4: Total execution time, energy consumption and average power demand measured to *lb_test* benchmark.

	NOLB	**GREEDYLB**	**REFINELB**	**FG-ENERGYLB alone**	$\frac{\text{FG-ENERGYLB}}{\text{NOLB}}$
Time (s)	79.4	55.8	51.3	79.6	1.002
Energy (J)	2883.0	2028.3	1865.3	2430.5	0.84
Power avg (W)	36.31	36.35	36.36	30.53	0.84

Source: The author

The greatest reduction in execution time was achieved when REFINELB was applied. GREEDYLB and REFINELB load balancers obtain speedup of 1.42 and 1.54 respectively when compared to the baseline (NOLB). This reduction in the execution time results in an equivalent reduction in the amount of energy spent. However, if we compare FG-ENERGYLB results with baseline, we observe that the execution time is very similarly, while total energy consumption is reduced up to 15.7%. This behavior is justified due to load imbalance present between cores, this that achieve up to 9.2 times at the end of the execution. During the runtime, FG-ENERGYLB load balancer can adjust the clock frequency in only 2 times, once that the range of clock frequency of these processors is from 2.4 to 1.2 GHz.

In this way, when the application load imbalance is greater than range of clock frequencies of the processors used, the use of the FG-ENERGYLB alone is not able to avoid the total imbalance and improve the execution time. Thus, in these cases, FG-ENERGYLB is able only to reduce the power demand. Therefore, to avoid this problem

FG-ENERGYLB can be performing together with other load balancers. So, when detected this behavior it automatically calls another load balancer to migrate tasks between the processor cores and after perform DVFS. The results of the FG-ENERGYLB employed together with other load balancers are discussed in following.

ii) LB + FG-ENERGYLB Evaluation

When FG-ENERGYLB was employed over the residual imbalance of other load balancing algorithms, the energy consumption of *lb_test* was reduced by up to 1% for REFINELB and REFINECOMMLB, 6.6% for GREEDYLB and GREEDYCOMMLB, 8.2% for SCOTCHLB, and 26.1% for RANDCENTLB. This improvement comes from a combination of two factors: the power demand of the cores was reduced to less than 35 W on average; and the reduction of the execution time.

RANDCENTLB used alone increase significantly the total execution time. This occurs, because this load balancer randomly assigns tasks between the cores resulting in a large imbalance load, once that this load balancer algorithm does not take into account any information about the application or platform to make its decisions.

Having a large imbalance load in the application, when FG-ENERGYLB is applied with RANDCENTLB, FG-ENERGYLB has a big range to vary the clock frequency of underloaded cores, which have some residual imbalance. In this context, the power demand is reduced significantly, from 34.79 to 25.19 W. So, in this test, during the runtime only one core, the most overloaded remains with the maximum available frequency, while all other cores have their clock frequencies reduced to minimum value. For this execution, the average clock frequency is reduced to 1.624 GHz.

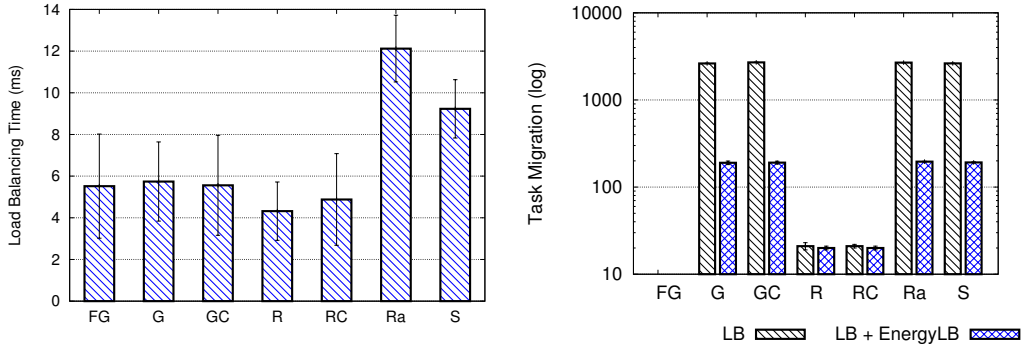
Differently from RANDCENTLB, when FG-ENERGYLB is used together with REFINELB and REFINECOMMLB, the residual imbalance is small. In this case, the average clock frequency of runtime is only reduced to 2.357 GHz.

Since FG-ENERGYLB performs DVFS instead of migrate tasks, the total number of task migrations is reduced. In this context, we evaluate the impact of load balancing step on the total execution time of the application, comparing the load balancing times (overhead) of FG-ENERGYLB with other load balancers. For this, we consider the average load balancing step during all execution.

Figure 5.3(a) presents a comparison of time overhead incurred on different strategies for load balancing step with our proposed FG-ENERGYLB. To compare this, we run the application using each one of the selected load balancers and measure the runtime that each one spends to collect load balancing data, make its decisions, find a new mapping to

each task and migrate the tasks or yet, change frequency if necessary.

Figure 5.3: Load balancing overhead and total task migration to *lb_test* with different load balancers.



(a) Load balancing overhead. (b) Total task migration.

Source: The author

Since the number of task migrations can increase the application execution time, we also selected the number task migrations for each load balancer to analyze together with load balancing time overhead, as is shown in the Figure 5.3(b).

For *lb_test*, the load balancing times varies from 4.3 up to 12.1 ms per load balancing call. This difference between the time is mainly justified by difference in the number total of tasks migrated. GREEDYLB, GREEDYCOMMLB, RANDCENTLB and SCOTCHLB migrate an average of 189.9 tasks in every call, which result in $2,659$ migrations during the benchmark execution, as shown in the Figure 5.3(b). On the other hand, REFINELB and REFINECOMMLB migrate only 20 tasks on first their call, being that in other calls any task is migrated.

RANDCENTLB migrates almost all the tasks whenever it was called. In this way, RANDCENTLB has an overhead higher than other load balancer algorithms. While that overhead of the other load balancers is 6 ms on average, for RANDCENTLB, the load balancing times is 12.12 ms. Furthermore, the tasks are randomly mapped to cores, which result in an increase on total execution time.

When FG-ENERGYLB was employed together with GREEDYLB, GREEDYCOMMLB, SCOTCHLB and RANDCENTLB, the total amount of task migrations is reduced from $2,659$ to 192 on average, which represent reduction of up to 72% (Figure 5.3(b)). As a result of a more balanced application after these migrations, a small residual imbalance was left to FG-ENERGYLB, leaving between 11 and 21 cores at their maximum clock frequencies. This number of migrations results in 1.89 MB of data transferred every call these load balancers. On the other hand, when it is used with REFINELB and REFINECOMMLB the amount of data migrated is 0.21 MB.

This reduction in the number total of task migrations result also in a reduction of the total execution time. For the tests with GREEDYLB, GREEDYCOMMLB and SCOTCHLB, the total energy consumption is reduced mainly due the reduction of total execution time, which represent energy saving of 6.2%, 5.8% and 8% respectively. Differently from the REFINELB, REFINECOMMLB and RANDCENTLB load balancers, the total execution time remain constant, similar to observed when FG-ENERGYLB was employed alone.

Using load balancing over *lb_test*, we can reduce the total energy consumption by up to 39% for SCOTCHLB, 40% for GREEDYLB and GREEDYCOMMLB, and 43% for REFINELB and REFINECOMMLB if compared to baseline. When applying FG-ENERGYLB together with these load balancers, we achieve even greater energy savings since the average power demand is reduced in up to 0.5% to GREEDYLB, GREEDYCOMMLB and SCOTCHLB; and up to 1.3% for REFINELB and REFINECOMMLB. Thus, the greatest reduction in total energy consumption is achieved using SCOTCHLB with FG-ENERGYLB, which represent 44.2% of energy savings, as shown in the Figure 5.1(a).

- Benchmark: *ComprehensiveBench*

i) FG-ENERGYLB Evaluation

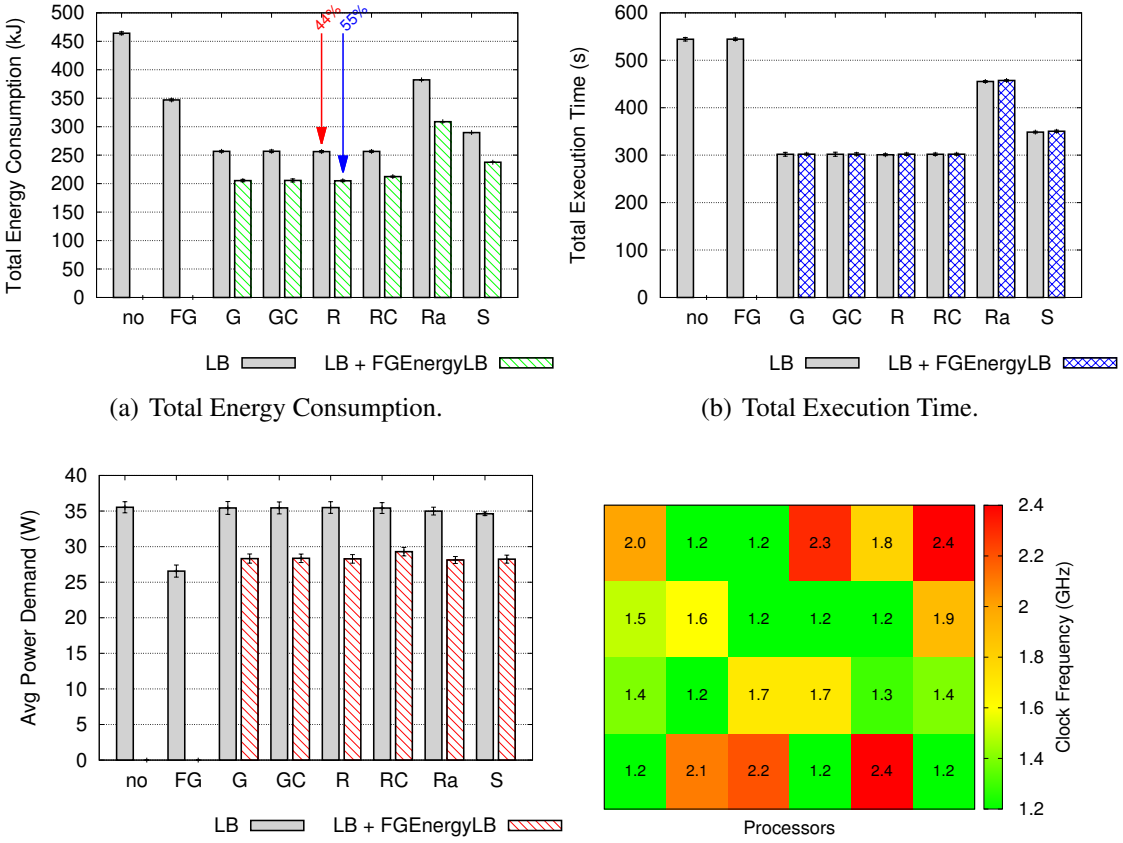
Similar to *lb_test*, this benchmark has a large imbalance when starting its execution. So, when FG-ENERGYLB was employed alone on *ComprehensiveBench*, it is able to reduce up to 25.2%, from 464.21 kJ to 347.09 kJ the total energy consumption. This energy saving occurs due a 25.1% reduction on average in power demand, from 35.5 W to 26.56 W as shown in Figure 5.4(c).

Using FG-ENERGYLB alone over *ComprehensiveBench*, the execution finishes having only 2 cores with maximum clock frequency and 9 core with their clock frequency reduced to minimum value available (Figure 5.4(d)). In this execution, the average frequency of the runtime was reduced to 1.604 GHz and at end of execution 18 cores were clock frequency registered below of 2.0 GHz.

We selected the relative imbalance load (ratio between *max_Wload* and *min_Wload*) present at the first load balancing call (initial imbalance) and at the last load balancing call (final imbalance) of this execution. Figure 5.5 illustrates the initial and final imbalance of *ComprehensiveBench* order by imbalance load.

ComprehensiveBench has a large variation in its task load, starting its execution with a large imbalance. In its initial state (gray bar on Figure 5.5), the system ran the

Figure 5.4: Evaluation of FG-ENERGYLB with *ComprehensiveBench* benchmark.



(a) Total Energy Consumption.

(b) Total Execution Time.

(c) Average Power Demand.

(d) Final Clock Frequency to FG-ENERGYLB alone.

Source: The author

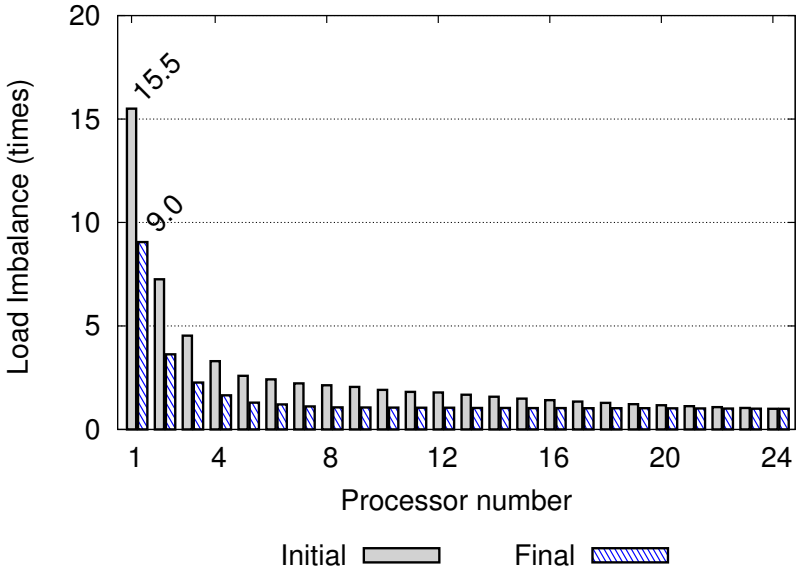
benchmark with all cores using the maximum clock frequency available (2.4 GHz). In this execution point, the load imbalance increases up to 15.5 times, i.e., the overloaded processor has 15.5 times more computational load than the underloaded processor. During the benchmark execution always that the FG-ENERGYLB load balancer is called, it does DVFS to adjust the clock frequency of cores according to their relative computational load. Only using DVFS, FG-ENERGYLB is able to reduce the imbalance load from 15.5 times to 9 times (blue bar on Figure 5.5), so resulting in a 25.2% of energy saving. However, it can not avoid all the load imbalance present in the execution.

For applications with a large imbalance, only adjusting the clock is not sufficient to avoid the total load imbalance. In these cases, FG-ENERGYLB can be employed together with other load balancers are discussed in following.

ii) LB + FG-ENERGYLB Evaluation

The improvements achieved employing over the residual imbalance of the selected load balancer are very similar. FG-ENERGYLB is able to reduce the energy consump-

Figure 5.5: Load imbalance relation between processors at initial and final execution.



Source: The author

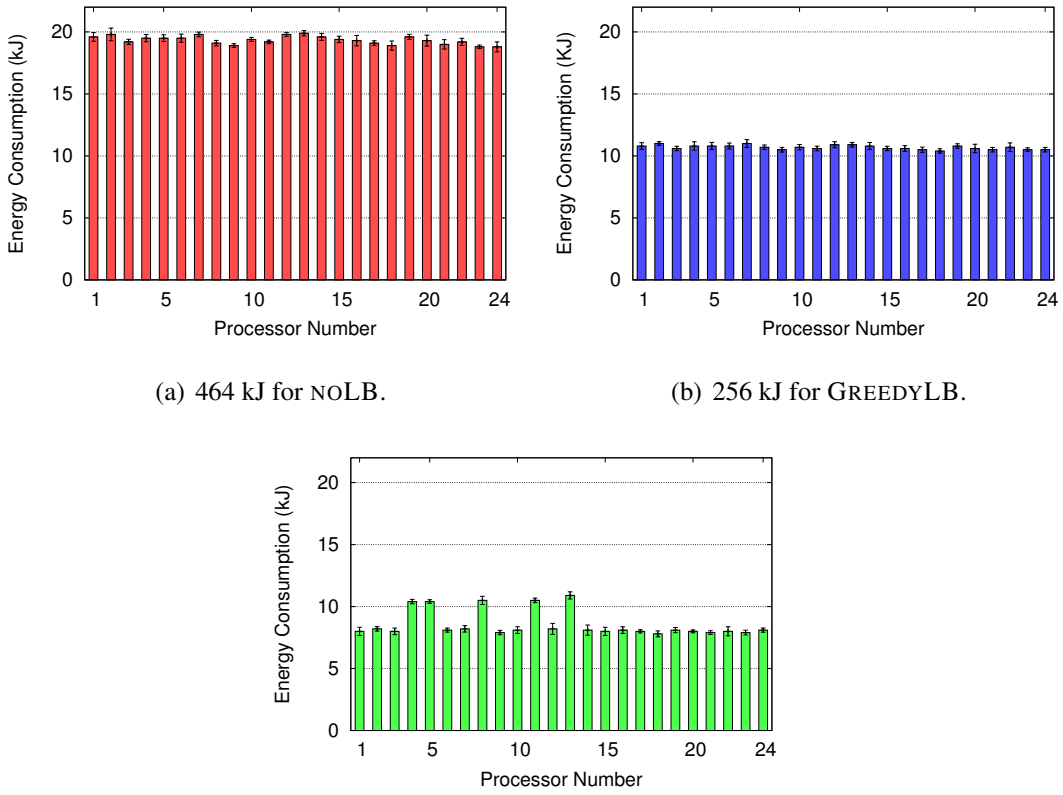
tion in 19.08% on average in this benchmark. It reduces by 20%, 19%, 20%, 17%, 19% and 18% for GREEDYLB, GREEDYCOMMLB, REFINELB, REFINECOMMLB, RANDCENTLB and SCOTCHLB respectively.

Since the energy saving is very similar between these load balancers, to better analyze the improvements achieved with this benchmark, we select the measured consumption of 3 executions (NOLB, GREEDYLB and GREEDYLB together with FG-ENERGYLB), as shown in the Figure 5.6.

Figure 5.6(a) shows the energy spent by each processor during the execution of benchmark without any load balancer strategy. Since all processors present a similar power demand during the execution, their energy consumption is also very similar, 19.33 kJ on average. In this execution, the total energy spent is 464.21 kJ.

GREEDYLB, illustrated in Figure 5.6(b), reduces the overall energy consumption to 256.09 kJ by improving load distribution of *ComprehensiveBench* and, consequently, reducing its total execution time. This resulted in a 44% energy consumption reduction when compared to the baseline (NOLB). Nevertheless, some residual imbalance was still present after tasks were remapped. In this case, when FG-ENERGYLB was employed together with GREEDYLB (Figure 5.6(c)). During the runtime, the less loaded processors had their clock frequencies reduced, which resulted in an improvement in energy consumption of 20% over GREEDYLB and 56% over the baseline (NOLB). Similar improvements were obtained over all other load balancers by keeping the average power of the cores at 29 W.

Figure 5.6: Total energy consumption of *ComprehensiveBench* in each processor.



(a) 464 kJ for NOLB.

(b) 256 kJ for GREEDYLB.

(c) 205 kJ for GREEDYLB + FG-ENERGYLB.

Source: The author

Different from the *lb_test* benchmark, on *ComprehensiveBench* the total execution time wasn't reduced in function of the reduction of amount of task migrations. In this benchmark, the load balancing overhead is similar and has values lower compared to other benchmarks. The overhead varies from 3.5 ms up to 7.5 ms per load balancing call. Since the overhead is almost zero, the reduction in the energy result of the power demand reductions during the benchmark execution.

Thus, for *ComprehensiveBench*, the adjust of the clock frequencies according to the residual imbalanced is responsible for the reduction of power demand and consequently reduction of total energy spent. The reductions of the power demand were similar the reductions in the total energy consumption, which represent 19.28% on average.

The greater energy saving for this benchmark is achieved using the GREEDYLB and REFINELB. Using REFINELB alone it is able to reduce in up to 44.7% the total energy consumption in relation to baseline. Applying FG-ENERGYLB together with REFINELB the average the average clock frequency is reduced to 1.857 GHz, reducing so the power demand from 35.5 W to 28.3 W. In this execution, using FG-ENERGYLB together with REFINELB, an amount of energy saving achieve 55.8%, as shown in the

Figure 5.4(a).

5.2.2 Evaluation on Real Applications

This section provides an evaluation of FG-ENERGYLB load balancer with the real applications *Ondes3D* and *Lulesh*, since each of them have different initial load imbalances and different dynamicities. The results of *Lassen* are presented in Appendix A.2. The results are organized by application, highlighting the total energy consumption, total execution time, average power demand and end final clock frequency when FG-ENERGYLB was used alone. For better analysis and evaluation of these applications, we also present the instantaneous load relative and instantaneous clock frequency of each real application.

Similar to observed in the tests with the benchmarks, each real applications has a different residual load imbalance. However, our load balancing algorithm has information of the application and platform, which allows it to make better decision, which are presented and discussed separately for each application in following.

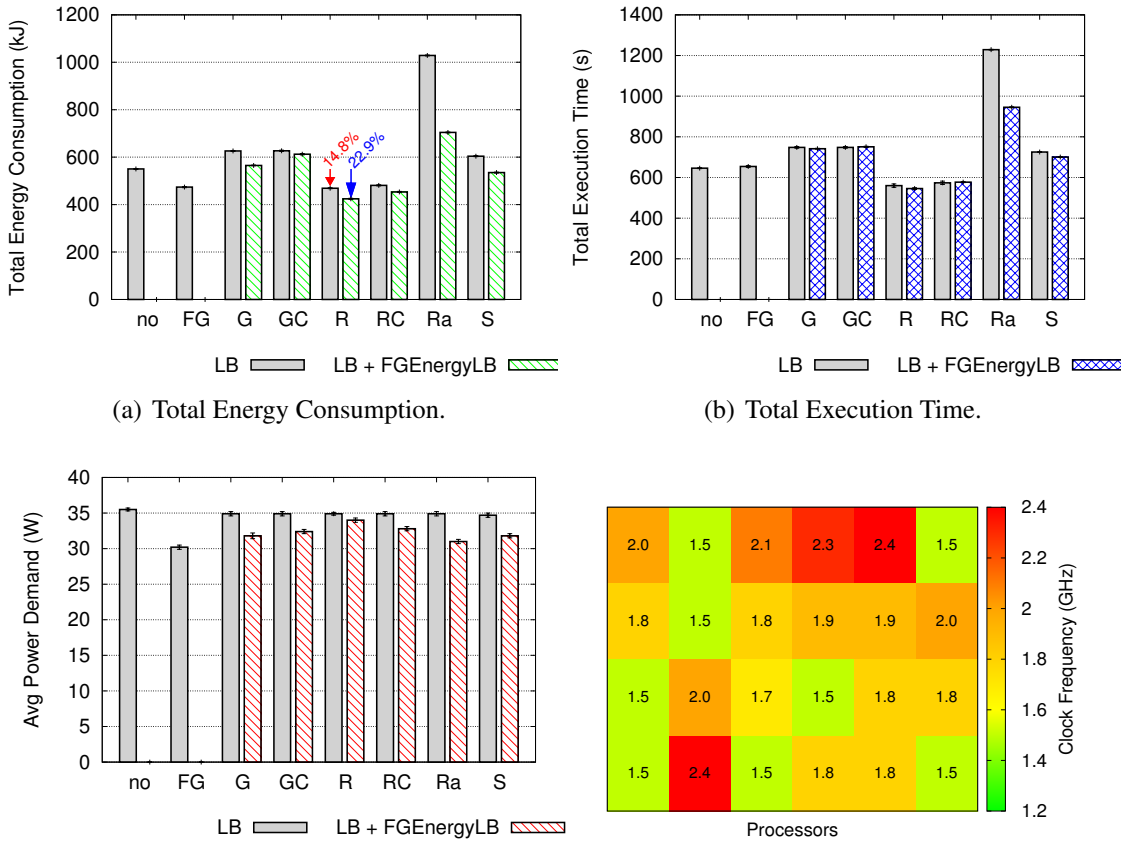
- Application: *Ondes3D*

i) FG-ENERGYLB Evaluation

This application starts with an almost static computational load. However, the propagation of the wave changes this behavior, introducing dynamism to the load. In this way, *Ondes3D* has both irregularity and dynamicity in its load. Applying FG-ENERGYLB alone over *Ondes3D* the total execution time increase from 645.8 to 654.2 seconds, which represent an overhead of 1.2% (Figure 5.7(b)). For instance, taking into account the original mapping of tasks, the start imbalance is of 1.5 times between the overloaded and underloaded processor, as shown in the Figure 5.8(a). This imbalance increases to 1.7 times in the second call and arrive 2.1 times in the 8^{th} call of the load balancer. From the 10^{th} call, the imbalance starts to reduce stabilizing in the 14^{th} call. From this point, little dynamicity is observed until the end of the execution. These imbalances during the execution leave room for energy consumption improvement.

Considering and aiming to correct this load imbalance, FG-ENERGYLB reduces the clock frequency of cores used during the runtime, as shown in the Figure 5.7(c). In this execution, the average frequency of cores was reduced to 1.804 GHz, which reduces

Figure 5.7: Evaluation of FG-ENERGYLB with *Ondes3D* application.



(a) Total Energy Consumption.

(b) Total Execution Time.

(c) Average Power Demand.

(d) Final Clock Frequency to FG-ENERGYLB alone.

Source: The author

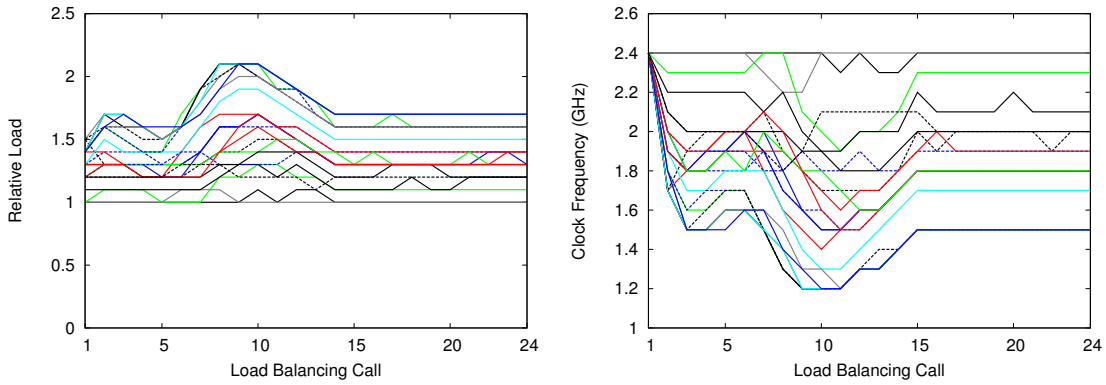
the power demand in 15% on average, from 35.5 W to 30.2 W. By reducing the processors power according their computational load, FG-ENERGYLB achieve saving energy of 13.9%, reducing the total energy consumption from 550.4 kJ to 473.8 kJ.

Using FG-ENERGYLB alone, the execution finishes having 8 cores with clock reduced to 1.5 GHz, 6 cores 1.8 GHz and only two cores remain with the maximum frequency (Figure 5.7(d)). That way, the system clock frequency at end of runtime was 1.812 GHz on average.

ii) LB + FG-ENERGYLB Evaluation

GREEDYLB, GREEDYCOMMLB, SCOTCHLB and mainly RANDCENTLB increase the total execution time of *Ondes3D*. The main reason for this increase is the time spent with load balancing. These algorithms spend 4.01, 4.16, 3.0 and 6.7 seconds on average per load balancing call, while FG-ENERGYLB, REFINELB and REFINECOMMLB only spend 0.3, 1.42 and 1.43 seconds, as shown in the Figure 5.9(a).

Figure 5.8: Comparison of Instantaneous Relative Load with Instantaneous Clock Frequency for *Ondes3D* when used FG-ENERGYLB alone.

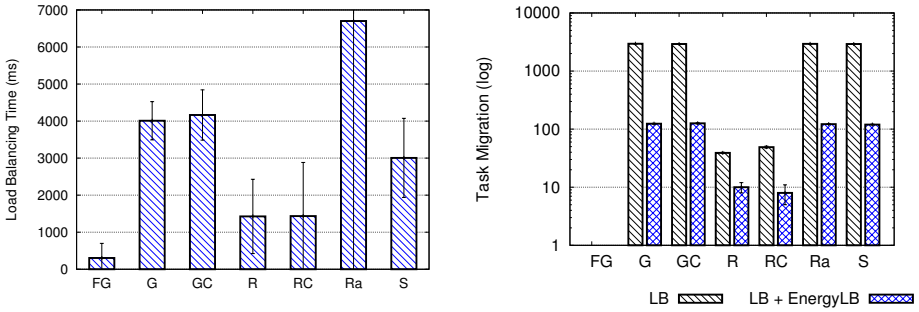


(a) Instantaneous Relative Load. (b) Instantaneous Clock Frequency.

Source: The author

This large number of migrated tasks result in a significant overhead. GREEDYLB, GREEDYCOMMLB, SCOTCHLB and RANDCENTLB migrate almost all their tasks (on average 96%) every call of the load balancer resulting in 1.23 MB of data transferred. On the other hand, REFINELB and REFINECOMMLB migrated only 3.1% and 17% of its tasks respectively, as shown in the Figure 5.9(b).

Figure 5.9: Load balancing overhead and Total task migration to *Ondes3D* with different load balancers.



(a) Load balancing overhead. (b) Total task migration.

Source: The author

When FG-ENERGYLB was applied over the residual imbalance of other load balancing algorithms, the total energy consumption was reduced by up to 31.5% for RANDCENTLB, 11.45% for SCOTCHLB and 8.67% for GREEDYLB. In these tests, FG-ENERGYLB has a small overhead and it is able to minimize task migrations that increasing overhead through of communication costs. For example, for GREEDYCOMMLB the total execution time increases by 0.54%. On the other hand, the average power demand is reduced up to 5.29%. With SCOTCHLB and RANDCENTLB, the FG-ENERGYLB achieve reductions of up to 8.3% and 11.3% respectively, compared to execution with

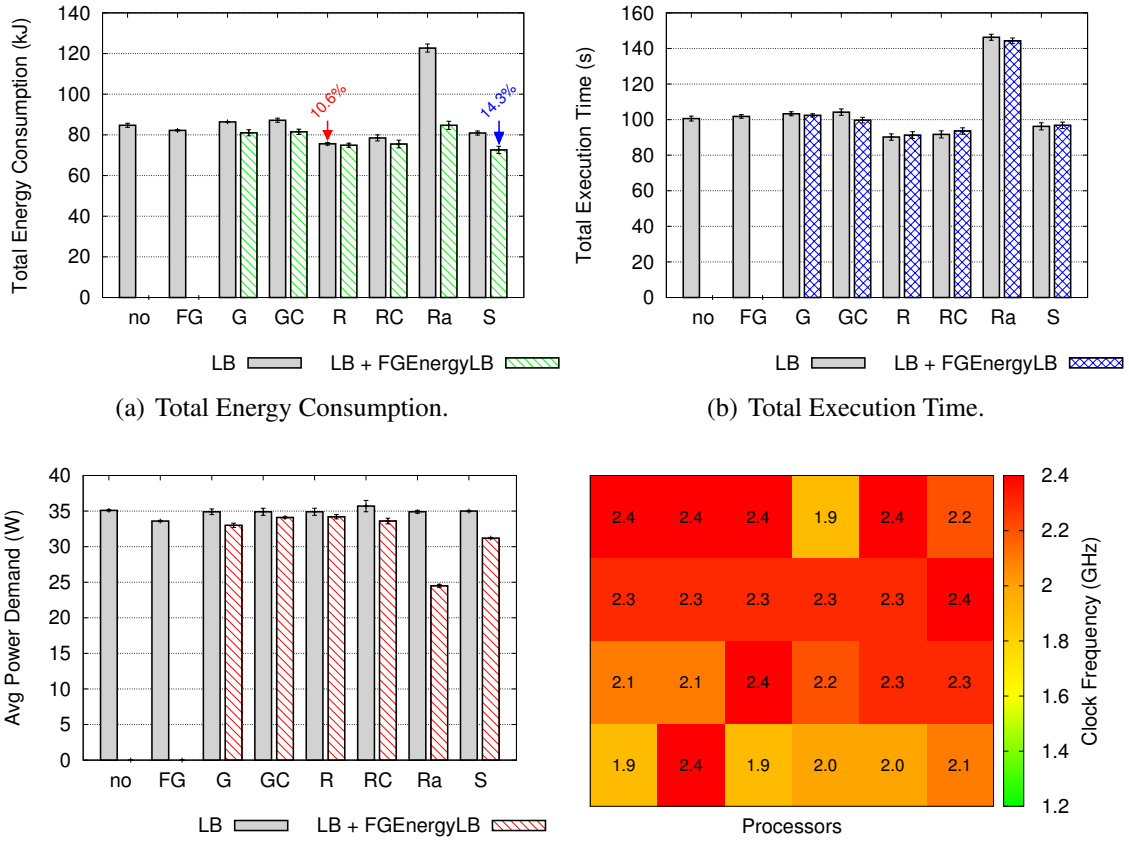
load balancer alone.

For this application the energy saving was mainly due the reduction of average power during the runtime. Only for RANDCENTLB, the improvements come from a combination of the power demand reduction and the reduction of the total execution time.

The greatest amount of energy saving for this real application is achieved with the REFINELB load balancer. Using REFINELB alone the total energy is reduced in up to 14.78% compared to baseline. However, using FG-ENERGYLB together REFINELB, the average power is reduced in 7.1% and the total execution time in 2.6%. With these reductions, the total of energy saving achieved 22.9%.

- Application: *Lulesh*

Figure 5.10: Evaluation of FG-ENERGYLB with *Lulesh* application.



(a) Total Energy Consumption.

(b) Total Execution Time.

(c) Average Power Demand.

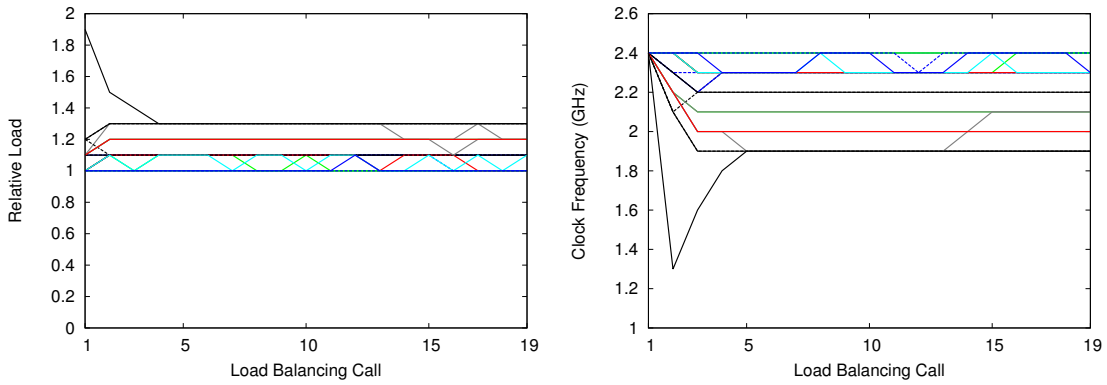
(d) Final Clock Frequency to FG-ENERGYLB alone.

Source: The author

i) FG-ENERGYLB Evaluation

Similar to *Ondes3D*, this application also has both load irregularity and dynamicity. Its irregularity comes from the motion of materials relative to other when subject to forces. It starts with imbalance of 1.9 times in the original mapping of its 729 task on 24

Figure 5.11: Comparison of Instantaneous Relative Load with Instantaneous Clock Frequency for *Lulesh* when used FG-ENERGYLB alone.



(a) Instantaneous Relative Load. (b) Instantaneous Clock Frequency.

Source: The author

cores. However, this irregularity reduces by 1.3 times in fourth call of load balancer, no more increasing until the end of the execution (Figure 5.11(a)).

Considering this low imbalance load throughout the execution, between 1.1 and 1.3 times, the FG-ENERGYLB applied alone has a small margin to make DVFS in cores used and reduce its power. After the fourth call of the load balancer, the frequency is not reduced to less than 1.9 GHz (Figure 5.11(b)). This way, considering this load imbalance, average clock frequency of the entire run is only reduced for 2.225 GHz, which reduces the average power of 35.06 W to 33.64 W, a reduction of only 4.05%.

At the end of the execution with FG-ENERGYLB applied alone, 3 cores have their clock reduced to 1.9 GHz, 7 cores to 2.3 GHz and 7 run with maximum frequency (Figure 5.10(d)).

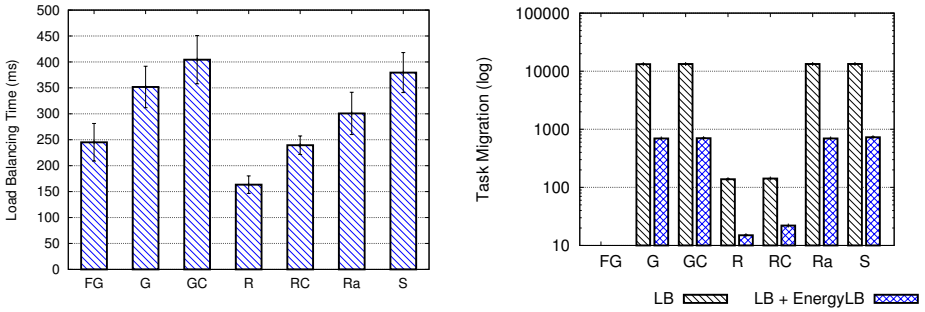
FG-ENERGYLB has 1.2% overhead in the tests with this application. This maintains the total execution time in approximately 101 seconds. However, the power reduction achieved is greater than the its load balancing overhead, which results in a reduction of the total energy consumption from 84.7 to 82.2 kJ (2.86%), as shown in Figure 5.11(b).

ii) LB + FG-ENERGYLB Evaluation

An increase in the runtime of this application can be seen for load balancers GREEDYLB, GREEDYCOMMLB and RANDCENTLB. These load balancers do not take into account the current task mapping, which result in large amount of task migrations at each load balancing call. The load balancing time are 351, 404 and 300 ms on average, per load balancing call. These load balancers migrate 698 tasks on average, which result in 27.92 MB of data transferred in each call (Figure 5.12(b)). On the other hand, REFINELB and REFINECOMMLB have times of 200 ms on average, once that the amount

of migrated tasks is very small, which is only 6 tasks on average. Figure 5.12 summarizes the load balancing costs and total task migration of *Lulesh*.

Figure 5.12: Load balancing overhead and Total task migration to *Lulesh* with different load balancers.



(a) Load balancing overhead. (b) Total task migration.

Source: The author

Residual imbalances remain after the task migrations of the load balancers. Aiming mitigate this residual imbalance, FG-ENERGYLB is able to reduce up to 6.6% when used together with GREEDYLB, 10.3% with SCOTCHLB and 30.9% with RANDCENTLB in the total energy consumption. These reductions comes mainly from of the power demand reduction, which is reduced 8.12% on average, once that the execution time is almost not increased.

In this application, the lowest energy consumption was achieved with SCOTCHLB and FG-ENERGYLB together. SCOTCHLB used alone reduces consumption by up to 4.44%. When FG-ENERGYLB is executed over the SCOTCHLB residual imbalance, the average power is reduced 10.9%, the runtime is also reduced in 3.76%, which represent an energy saving of up to 14.28% compared the baseline.

5.2.2.1 Percentage of Energy Spent on Load Balancing

Running scientific applications that represent a real world scenarios demand high processing power and more memory space. Real applications as *Ondes3D*, *Lulesh* and *Lassen* have different initial imbalance and different dynamicity in their computational load, which leaving space for the use of load balancing algorithms aiming to mitigate load imbalance. In this context, some common energy improvements were achieved.

In the evaluation, we realize that FG-ENERGYLB has load balancing times lower than all load balancers when tested with *Ondes3D*, and lower than some load balancers in execution with *Lulesh*.

Table 5.5 depicts the comparison of load balancing times incurring on FG-ENERGYLB

and other load balancer strategies.

Table 5.5: Average load balancing duration in seconds for each real application.

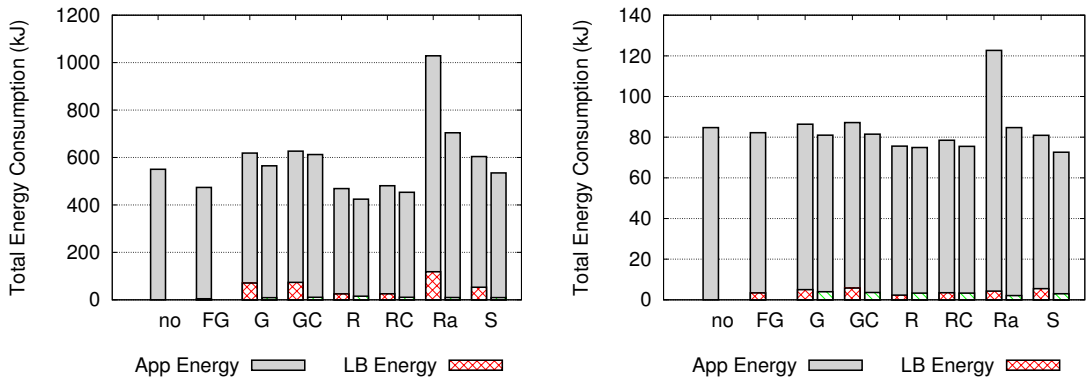
Load Balancer	- - *Ondes3D* - -		- - *Lulesh* - -	
	LB	**LB+FG**	**LB**	**LB+FG**
FG-ENERGYLB	0.30	-	0.25	-
GREEDYLB	4.01	0.57	0.35	0.29
GREEDYCOMMLB	4.16	0.65	0.40	0.26
REFINELB	1.43	0.88	0.16	0.23
REFINECOMMLB	1.43	0.64	0.24	0.24
RANDCENTLB	6.70	0.62	0.30	0.20
SCOTCHLB	3.01	0.58	0.38	0.23

Source: The author

Most load balancing algorithms try to mitigate load imbalance by moving tasks between processors. However, this load balancing incurs in overheads to data collection, decision making and mainly tasks migration, which may affect the application's runtime and consequently the total of energy spent.

Generally, the use of load balancing approaches results in reducing the total execution time and represent also saving energy, as evaluated for benchmarks on Section 5.2.1. In this context, we analyzed the total energy spent with load balancing (LB Energy) of every test performed and we relate it with the total energy consumption (App Energy), as depicted in the Figure 5.13.

Figure 5.13: Energy spend with load balancing over total energy consumption for each real application.



(a) *Ondes3D*. (b) *Lulesh*.

Source: The author

The employ of load balancer algorithm is important to improve the scalability of large parallel systems once they manage tasks distribution, aims to mitigate the effects of load imbalance and some costs of communication. On the other hand, they introducing

overhead in the total execution time of the application, which increases the total energy consumption.

In this way, the energy spent with load balancing comes from of the time overhead of every load balancer call. In the test with *Ondes3D*, the load balancers are responsible in average for spending of 7.87% of the total energy consumed during execution of the application (Figure 5.13(a)). When FG-EnergyLB was used together with load balancers is able to saving energy of own load balancing. The energy spent with load balancing was reduced from 7.87% to 2.07%, on average. This occurs due the reduction of the total execution time and mainly by reductions on average power demand.

Over *Lulesh*, the load balancing is responsible for 4.91% of the total energy consumed during execution of the application. For this application, using FG-EnergyLB together with other load balancers, the total energy spent with load balancing is reduced from 4.91% to 3.52%, on average, as shown in the Figure 5.13(b).

5.2.2.2 Threshold Evaluation

As discussed in Section 2.2.1, defining the interval between calls to the load balancer is decisive to reduce the load balance overhead. If the load balancer is invoked in long time periods, the load imbalance may increase too much and result in loss of performance, which consequently increases the total energy consumption. On the other hand, if the strategy is performed very frequently, it also may incur in a reduction of performance, since the load balancing overhead may exceed its benefits. In this context, aiming to decrease the load balancing overhead, recent strategies have adopted a threshold value to determine if load balancing must be performed or not.

Several load balancers are able to reduce the total energy consumption reducing the application execution time. Our proposed algorithms try to reduce the total energy consumption by exploiting residual imbalance left by load balancing algorithm. They identify the possibility of reducing the processors clock to achieve better gains over these algorithms. In this form, energy improvements are achieved due the reduction of average power during the runtime and also through of reducing the application execution time by reducing the amount of tasks migrated.

Aiming to reduce the effects of load imbalance and load balancing overhead to save energy, this section provides a FG-EnergyLB evaluation over real applications using different threshold values. As mentioned in Section 2.2, the application runtime depends on several issues, among them, the number of parallel tasks and their load, the

duration of each timestep, and the selected load balancing strategy. The impact of load balancing is directly related to the load balancing frequency once load balancing overhead can overcome the gains achieved with load balancing.

In this way, to FG-ENERGYLB, in each call of the load balancer, is verified if the weighted load of each processor exceeds or not the *threshold*, make decisions to adjust the frequencies (determining so that the frequency will be decreased or increased) or invoke other load balancer to migrate tasks. However, the load balancer generates an overhead and when this cost exceeds its benefits, the total execution time is increased, i.e., calling load balancing strategies incurs timing penalty to applications.

Our proposed load balancers take three input parameters in their execution. The first one, is the load balancer that is used to migrate tasks when the imbalance is high. The second one, is the maximum frequency available by processors that can be set to a core, and the last one, is a threshold value, used to decide whether call the load balancer or perform DVFS strategy.

In the previous sections the FG-ENERGYLB was analyzed with threshold value equal to 2.0, since the clock frequency range available on our experimental platform allows us to vary the clock frequency of the processor from 1.2 GHz up to 2.4 GHz.

In order to make a trade-off between runtime, power demand and total energy consumption, we vary the threshold parameter of the algorithm from 0 up to 5 and execute the applications again. Due to reasons of space, the results of *Lassen* are presented in Appendix A.2.

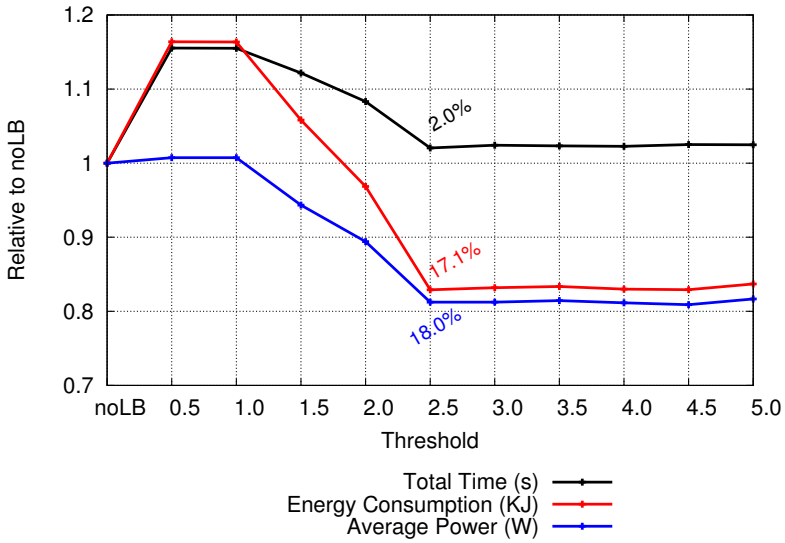
Running the applications with FG-ENERGYLB configured with different threshold values, we obtain different amounts of DVFS performed or load balancing called what determines different frequency settings of cores or migration tasks. In this way, we can analyze which is the best threshold value for each application under the total energy consumption, total execution time or the average power demand focus.

- *Ondes3D* Application

Experiments with *Ondes3D* were performed using 128 tasks, which run 500 iterations each. Total energy spent to run this application without load balancer is 550.4 kJoules and its total execution time is 645.8 seconds. During this execution, the average power demand is 35.5 W. These values are taken as reference in the analysis and represent the NOLB value in the Figure 5.14.

In the tests with FG-ENERGYLB load balancer, it is called at each 20 iterations, resulting in a total of 24 calls. Using threshold value equal to 0.5 and 1.0, FG-ENERGYLB

Figure 5.14: FG-ENERGYLB comparison with different threshold value on *Ondes3D*.



Source: The author

no performs any time DVFS, in all calls tasks are migrated calling GREEDYLB load balancer. In this form, the average power remains constant around of 35.5 W. However, performing migrations in this application is very costly, which incurs in an increase of 15.5% in runtime and total energy consumption. This increase is the result of overhead migrations undertaken by load balancer.

Using a threshold equal to 1.5, FG-ENERGYLB load balancer adjust 18 times the clock frequency through DVFS and only 6 times call other load balancer to migrate tasks. In this way, it is able to reduce the average power in 5.7%. Reducing the amount of migration, the runtime suffers a small reduction to 724.38 seconds, but still 12.2% larger and spending 5.82% more energy than the execution without load balancer.

To threshold value equal to 2.0 are achieved gains in both execution time, power demand and consequently energy consumption. Were performed 18 times DVFS during the execution, which reduce the average power in 10.6%, but the runtime still is 8.32% larger than the baseline. Nevertheless, using this threshold the total energy consumption is reduced in only 3.15%.

For threshold from 1.0 up to 2.5, the increase of threshold value also increases the number of calls of DVFS. For these values, the runtime has a reduction near to linear. The total execution times is reduced from 15% to 2% larger than the baseline. Similarly, for this threshold range, the average power demand of the parallel platform is reduced in up to 19% less than baseline. In this way, both runtime and power demand reductions contribute to reduce the total energy spent.

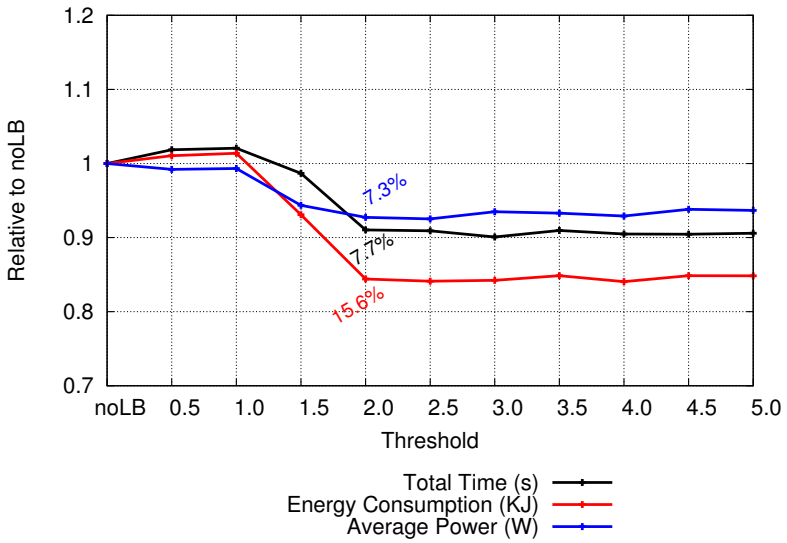
The greater energy saving for this real application is achieved using the threshold

value equal or greater than 2.5. Using these values FG-ENERGYLB is able to reduce in up to 17.1% the total energy consumption in relation to baseline NOLB. These gains are achieved through reduction in the overhead, which is only 2% and reduction of the average power is reduced in 18%, once that in all calls were performed DVFS, which result in greater amount of energy saving, as shown in the Figure 5.14.

- *Lulesh* Application

Lulesh was executed with 1000 iterations in each one of its 729 processes mapped in 24 cores. This application spent 100.6 kJoules of energy and takes 84.7 seconds when executed without load balancer. In this execution the average power demand is 35.1 W. These values are taken as reference (baseline) and shown in column NOLB of the Figure 5.15) to examine the threshold variation of the *Lulesh* application.

Figure 5.15: FG-ENERGYLB comparison with different threshold value on *Lulesh*.



Source: The author

Load balancing call is configured with a frequency call of 50 iterations, so executed 19 times during the execution. When used threshold equal to 0.5 and 1.0 the FG-ENERGYLB not perform any time adjust in clock frequency, it only migrates tasks calling the other load balancer. So, using these thresholds the average power is not changed. In addition, in these tests the load balancing overhead increases the runtime and consequently, the total energy consumption in up to 2%.

A greater amount of DVFS is performed when the value of threshold is increased. To threshold equal to 1.5, FG-ENERGYLB calls 17 times DVFS and only 2 times task migrations. Thus, reduces the runtime in up to 1.33%, which also contributes to reduce the total energy spent. FG-ENERGYLB achieves a reduction of up to 5.6% in average

power demand. In this way, reducing both, the power demand and runtime, the total of energy spent is reduced in up to 6.91%.

The energy saving further increases when using thresholds equal to 2.0 or greater. For these values, the runtime reduction is greater than the reduction of the average power demand. In every call of the load balancer were performed DVFS, which result in reductions of 7.3% in average power demand and an average performance improvement of 7.74% compared to NOLB. In this way, FG-ENERGYLB is able to save energy in up to 15.6% to *Lulesh*.

For this application, the threshold variation from 1.0 up to 2.0 present the reduction more significant in the execution time. When used these values, the runtime is reduced in up to 11%, while that the average power demand is reduced in up to 6%. Similarly to *Ondes3D*, both runtime and power demand reductions contribute to reduction of the total energy consumption.

FG-ENERGYLB load balancer achieved significant energy savings on all benchmarks evaluated in the previous section. For those experiments, were used a single core of each one of the 24 processors available, since our experimental platform does not feature processors capable of performing *per-core* DVFS. However, is undesirable limiting the number of cores when running scientific applications, since it demands high processing power in real world scenarios. In this case, our CG-ENERGYLB can be applied to overcome such limitation, allowing us to use all the 192 cores available on platform. In the next section, we show present the energy saving of our proposed hierarchical load balancer using all hardware resources available in the parallel system.

5.3 CG-ENERGYLB Evaluation

The second part of our evaluation presents the benefits of CG-ENERGYLB, our hierarchical approach over real world applications presented in the Section 5.1. As discussed in Section 4.1.2, CG-ENERGYLB considers a hierarchical view of the platform, which is represented as a two-level tree. In this way, ENERGYLB is employed at the root level, while other load balancers are used at the leaf level.

We organize the CG-ENERGYLB evaluation in two parts. First, we present the runtime, power and energy improvements and after, we analyze the percentage of energy spent on load balancing. Finally, we compare the results achieved using different threshold values in our hierarchical energy-aware approach.

These scientific applications present a dynamic behavior, as the load of its tasks change through the iterations, which provides a more challenging scenario for energy aware load balancing. Since all the 192 cores will be used, different parameters are used in the evaluation of the CG-ENERGYLB, as shown in the Table 5.6.

Table 5.6: Summary of the input parameters of real applications.

Application	**Tasks**	**Iterations**	**LB Frequency (Iterations)**
Ondes3D	1024	500	20
Lulesh	5832	1000	50
Lassen	512	550	50

Source: The author

Similar to FG-ENERGYLB evaluation, the results are also present combining the executions of: without load balancer (NOLB), with our CG-ENERGYLB (CG) and other CHARM++ load balancers alone (GREEDYLB (G), GREEDYCOMMLB (GC), REFINELB (R), REFINECOMMLB (RC), RANDCENTLB (Ra) and SCOTCHLB (S)).

The results are organized by gains achieved when CG-ENERGYLB is used alone, following of gains when CG-ENERGYLB was employed over the residual imbalance of other load balancing algorithms, as following:

5.3.1 Evaluation on Real Applications

This section present the results achieved using the CG-ENERGYLB over real applications. Similar to previous evaluations, the results of *Lassen* are presented in Appendix B.1. The results are organized by application, highlighting the total energy consumption, total execution time, average power demand and end final clock frequency when CG-ENERGYLB was used alone.

- Application: *Ondes3D*

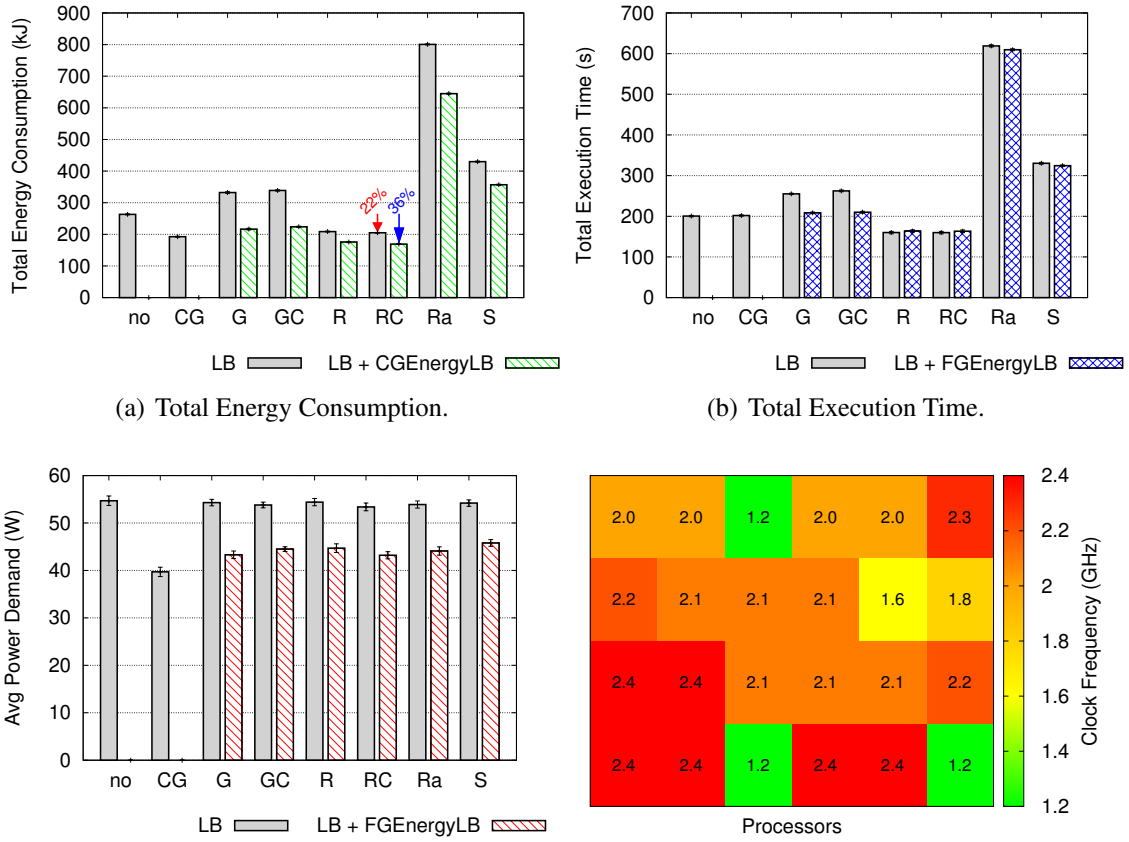
As discussed in Section 5.2.2, *Ondes3D* has irregularity in the creation of its tasks and load dynamicity during execution. This load irregularity comes from boundary conditions that produce additional work, and load dynamicity from the simulation of waves spreading through space.

Using all 192 cores available in the system, the application has at the start of running an unbalance up to 1.16 times. This imbalance exceeds 4 times in the fourth, fifth

and eighth calls of load balancer. In other calls, the imbalance remains below 4 times.

Taking advantage of this load imbalance, each load balancer has different results regarding the total execution time, power demand and average power demand during runtime. Figure 5.16 presents a comparison of total energy consumption, total execution time and average power demand for different load balancing strategies with our proposed CG-EnergyLB. It also presents the clock frequency of each processor when application execution finished using CG-EnergyLB.

Figure 5.16: Evaluation of CG-EnergyLB with *Ondes3D* application.



(a) Total Energy Consumption.

(b) Total Execution Time.

(c) Average Power Demand.

(d) Final Clock Frequency to CG-EnergyLB alone.

Source: The author

i) CG-EnergyLB Evaluation

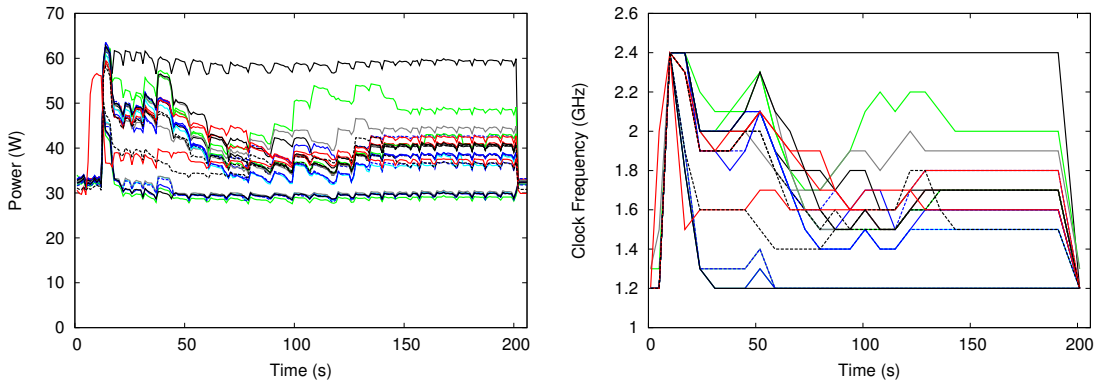
When CG-EnergyLB is employed alone, it is able to reduce the total energy consumption by 27% (from 263.1 kJ to 192.5 kJ). This reduction represents 26.86% of total energy spent during application runtime (Figure 5.16(a)). As CG-EnergyLB has a small overhead (0.7%), which almost does not influence the execution time of the application (Figure 5.16(b)), thus, the energy saving is mainly result of the power reduction in

the processors as shown in the Figure 5.16(c).

In order to observe how CG-ENERGYLB reacts to load imbalances during the execution of *Ondes3D*, we profiled its execution with EMonDaemon. Figure 5.17 shows the instantaneous power demand and clock frequency of the 24 processors measured during its execution. As it can be observed, CG-ENERGYLB sets the frequency of the overloaded processor to the maximum value available (2.4 GHz), resulting in a higher power demand of this processor. On the other hand, the frequency of underloaded processors is set to the minimum (1.2 GHz) or to intermediate level according to its relative load.

Through these frequency adjustments, the average clock while running was reduced to 1.631 GHz. The execution with CG-ENERGYLB alone finishes having 8 cores with clock frequency of 2.4 GHz (Figure 5.16(d)), resulting in an average clock frequency of the system at the end of execution of 2.029 GHz.

Figure 5.17: Instantaneous power demand and clock frequency measured during the execution of *Ondes3D* with CG-ENERGYLB.



(a) Power. (b) Clock Frequency.

Source: The author

We can observe some drops in the power demand on all processors throughout the execution in Figure 5.17(a). This happens at each load balancer step. When CHARM++ performs load balancing, it temporarily stops the execution of the application to execute the load balancer. In our case, during this period CG-ENERGYLB is either performing DVFS or executing the load balancer to migrate tasks.

By updating the frequency of the processors according to their weighted loads, CG-ENERGYLB reduces the average power demand of the system while running the application. This reduction, shown in Figure 5.16(c), consequently reduce the total energy consumption, presented in Figure 5.16(a). In this execution, a reduction in power demand of up to 27.4% is achieved, reducing it from 54.71 W to 39.72 W.

ii) LB + CG-ENERGYLB Evaluation

Once applied load balancers, residual imbalances can still be present. In this way, when CG-ENERGYLB is employed to mitigate this residual imbalance left by the load balancing algorithm, it identifies the possibility of reducing the processors clock to achieve better gains over these algorithms. For *Ondes3D* application, the energy consumption is reduced between 15% (REFINELB) and 34% (GREEDYCOMMLB).

Table 5.7 summarizes the total energy consumption and improvement for *Ondes3D* on parallel platform with different load balancers. Energy consumption for NOLB represents a baseline execution when any load balancer was used. The energy reduction over NOLB represents an improvement of up to 1.37 times, which is higher than the improvement obtained for REFINELB and REFINECOMMLB, of 1.26 and 1.28 times, respectively. On the other hand, we can notice that GREEDYLB, GREEDYCOMMLB, RANDCENTLB and SCOTCHLB when used alone slowed down the application by approximately 1.27, 1.30, 3.08 and 1.64 times, respectively, increasing the total energy consumption.

However, GREEDYLB, GREEDYCOMMLB, REFINELB and REFINECOMMLB used with CG-ENERGYLB have reductions, which represent improvements between 1.18 and 1.56 over NOLB, as shown in Table 5.7. In best case, to REFINECOMMLB, the energy consumption of *Ondes3D* was reduced to 169.1 kJ, which represent a reduction of 35.7% in total energy consumption. This energy saving comes from a combination of two factors: i) reduction of the processors power demand of 21% (Figure 5.16(c)); and ii) reduction of the application execution time of 18.5% (Figure 5.16(b)). CG-ENERGYLB employed together other load balancers, represent improvements from 1.19 up to 1.53 times over own load balancer execution time.

Table 5.7: Total energy consumption and improvements for CG-ENERGYLB and other load balancers.

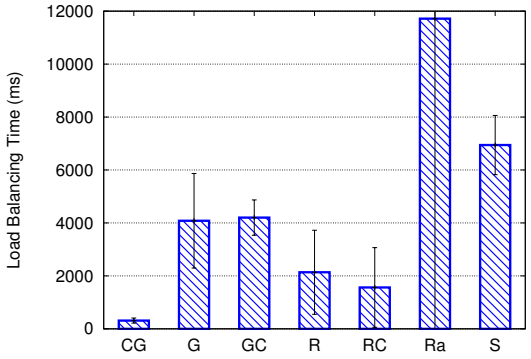
	----- LB alone -----		**----- LB + CG-ENERGYLB -----**		
	Energy Consumption (kJ)	**Improvement over NOLB**	**Energy Consumption (kJ)**	**Improvement over NOLB**	**Improvement over own LB**
NOLB	263.1	-	-	-	
CG-ENERGYLB (CG)	192.5	1.37	-	-	
GREEDYLB (G)	332.3	0.79	216.6	1.21	1.53
GREEDYCOMMLB (GC)	338.7	0.78	223.9	1.18	1.51
REFINELB (R)	208.8	1.26	175.7	1.50	1.19
REFINECOMMLB (RC)	205.1	1.28	169.1	1.56	1.21
RANDCENTLB (Ra)	800.7	0.33	645.0	0.41	1.24
SCOTCHLB (S)	429.9	0.61	356.8	0.74	1.20

Source: The author

The execution times measured for the *Ondes3D* are presented in Figure 5.16(b). Similar to FG-ENERGYLB evaluation, was noticed that load balancers slowed down the

application. This is a result of the overhead coming from task migrations. When the load balancers are applied in the *Ondes3D*, they present a wide variation in load balancing times. GREEDYLB, GREEDYCOMMLB RANDCENTLB and SCOTCHLB have overhead of 4.08, 4.20, 11.71 and 6.94 seconds on average, respectively. On the other hand, the load balancers CG-ENERGYLB, REFINELB and REFINECOMMLB spend only 0.3, 2.13 and 1.56 seconds in each step (Figure 5.18).

Figure 5.18: Average load balancing overhead to *Ondes3D* application.



Source: The author

This different overhead comes from of the different times to make their decisions and the amount of migrated tasks by these load balancers. While GREEDYLB, GREEDYCOMMLB RANDCENTLB and SCOTCHLB migrate an average of 1021, 1024, 1020 and 921 tasks every call. This high number of task migration, resulting in 59.79 MB of data transferred between the cores, which incurs in an increase in the total execution time.

On the other hand, REFINELB and REFINECOMMLB migrate only 460 and 164 tasks respectively, which have only 0.01 MB of data, resulting in 4.6 and 1.64 MB of data transferred between the cores

When CG-ENERGYLB was used along with GREEDYLB and GREEDYCOMMLB, it is able to reduce the total execution time compared to these load balancers alone. However, the execution time of *Ondes3D* was slightly increased by approximately 4% on average, in comparison to the execution without load balancer. This occurs since to CG-ENERGYLB does not migrate task between processors, only performs task migrations between cores of the same processor.

This different percent of increase in the runtime is due to the fact that CG-ENERGYLB only relies on other load balancers when the load imbalance is considerably high, i.e., greater than threshold value. When imbalance is lower than threshold, only doing DVFS performing so much less task migrations.

In contrast with that, REFINELB and REFINECOMMLB were able to improve the execution time of *Ondes3D* by up to 20% on average. These results are also reflected on the energy savings obtained through load balancing in comparison to GREEDYLB and GREEDYCOMMLB, as the average power measured for all these load balancers was approximately the same (54 W). When CG-ENERGYLB was applied with REFINELB and REFINECOMMLB, we observed a slightly increase of 1.7% in the execution time of *Ondes3D* in comparison to the execution with only GREEDYLB and GREEDYCOMMLB. However, CG-ENERGYLB is able to reduce even more the energy consumption of REFINELB and REFINECOMMLB by 16.7% on average in these cases by exploiting residual imbalances.

Overall, the results obtained with CG-ENERGYLB showed energy savings of up to 36% with an average of 23%. These energy improvements come from the exploitation of residual imbalances and from the reduction of the average power demand on all processors. With CG-ENERGYLB, the average power demand was reduced from 54.7 W to 44.3 W in comparison to the standard load balancers alone.

- Application: *Lulesh*

i) CG-ENERGYLB Evaluation

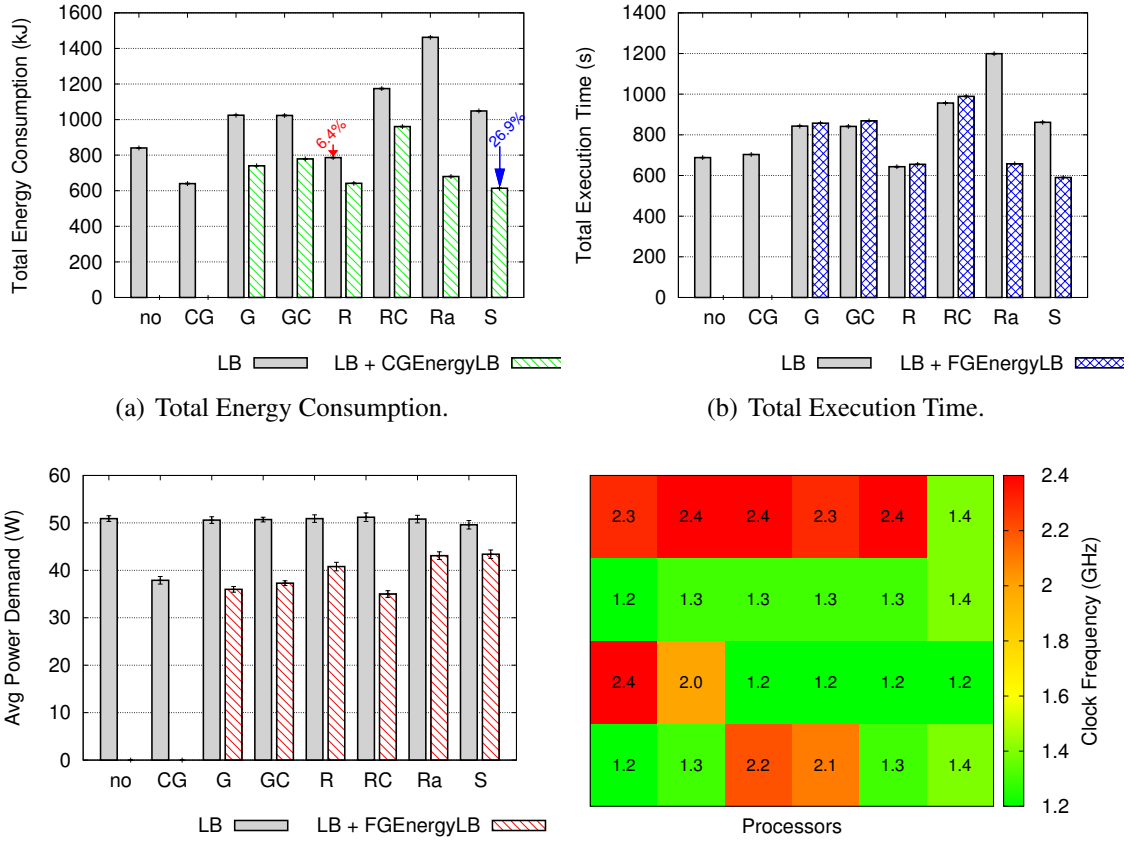
Lulesh has irregularity in the creation of its 5832 tasks over 192 cores and load dynamicity in its execution. Due the task distribution and irregular computational load, this application start with an imbalance load up to 1.45 times. However, during the execution due its dynamicity, imbalance increases up to 5.21 times.

The average power demand obtained on the tests with *Lulesh* are shown in Figure 5.10(c). Due its initial imbalance generated by original task mapping, when CG-ENERGYLB is applied alone, it has a good range for conducting DVFS and save energy. It is able to reduce the average power from 50.91 W to 37.93 W, a reduction of 25.5%, result of the reduction of the clock frequency during the entire run for 1.97 GHz.

At the end of the run this application with CG-ENERGYLB alone, the average frequency is 1.68 GHz. 14 cores have their clock reduced to 1.4 GHz or less, and only 4 cores performed with the maximum frequency.

The CG-ENERGYLB has an overhead of 2.23% in the tests with this application. It increases the total time of execution from 688.0 to 703.3 seconds. However, the gains

Figure 5.19: Evaluation of CG-ENERGYLB with *Lulesh* application.



(a) Total Energy Consumption.

(b) Total Execution Time.

(c) Average Power Demand.

(d) Final Clock Frequency to CG-ENERGYLB alone.

Source: The author

achieved with the power reduction is higher than its overhead. Thus, it achieves a reduction in the total consumption from 840.6 to 640.3 kJ (23.8%), as shown in Figure 5.19(a).

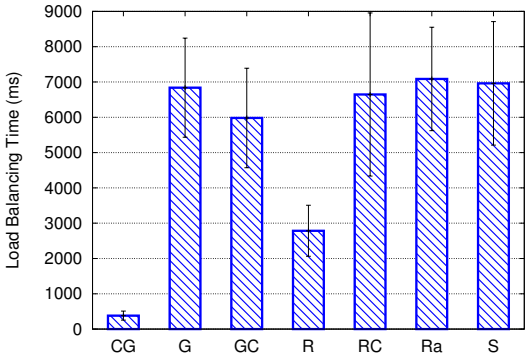
ii) LB + CG-ENERGYLB Evaluation

Load balancers are used to mitigate imbalances between processors. However, according to application and system characteristics, some residual imbalances can be present after making decision and the tasks migration. In way, employing CG-ENERGYLB over the residual imbalance of the *Lulesh*, the energy consumption was reduced between 18.2% (REFINECOMMLB) and 41.1% (SCOTCHLB).

Except REFINELB, all other load balancers increased the runtime of this real application. One cause that explain these increases is the amount of task migrations. GREEDYLB, GREEDYCOMMLB, RANDCENTLB and SCOTCHLB migrate 5802 tasks on average. These migrations move 2.088 MB of data between cores every call. This way, the load balancing times these balancers are 6.8, 5.9, 7.1 and 6.964 seconds on average. On the other hand, REFINELB and REFINECOMMLB have smaller number of

migration, only 357 tasks, that represents 7.14 MB of data migrated. So, their overhead is also lower, 2.7 and 6.6 seconds on average, as shown in the Figure 5.20.

Figure 5.20: Average load balancing overhead to *Lulesh* application.



Source: The author

When CG-ENERGYLB is applied together other load balancers over *Lulesh*, the total energy consumption is reduced in 33.9% on average. Performing DVFS to adjust clock during the execution of this application, CG-ENERGYLB is able to reduce the average power demand in 20.6% on average.

For this application, the lowest energy consumption was achieved with REFINELB. It reduces in up to 6.4% compared to baseline. However, using CG-ENERGYLB together with load balancers, the greatest reduction in energy consumption was achieved with SCOTCHLB and CG-ENERGYLB. The total energy consumption was reduced to 614.1 kJ, which represents 26.9% compared to baseline.

5.3.1.1 Percentage of Energy Spent on Load Balancing

Run scientific applications that represent simulation of real world scenarios demands even bigger processing power and more memory space. *Ondes3D*, *Lulesh* and *Lassen* application are example these scenarios in regards to load imbalance, dynamicity load, processing demand and memory used.

In this context, aiming to mitigate load imbalance, load balancing algorithms are applied, which achieving different energy improvements with different times overhead. In summary, we realize that our CG-ENERGYLB has load balancing times lower than other selected CHARM++ load balancers when tested with real applications.

Table 5.8 presents a comparison of load balancing times incurring on different strategies for the load balancing step with our proposed CG-ENERGYLB algorithm.

Since CG-ENERGYLB has load balancing times lower than other CHARM++ load

Table 5.8: Average load balancing duration in seconds for each real application.

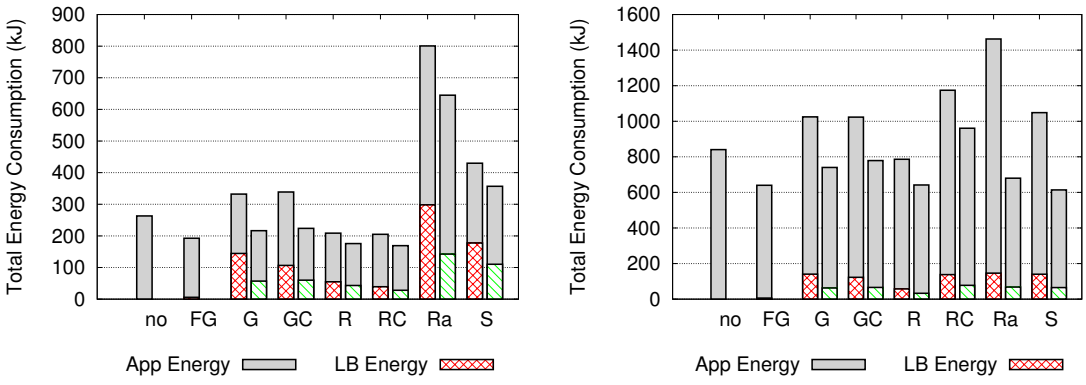
Load Balancer	- - *Ondes3D* - -		- - *Lulesh* - -	
	LB	**LB+CG**	**LB**	**LB+CG**
CG-ENERGYLB	0.31	-	0.38	-
GREEDYLB	4.08	3.85	6.84	4.72
GREEDYCOMMLB	4.20	3.95	5.98	4.78
REFINELB	2.14	2.83	2.78	2.17
REFINECOMMLB	1.56	1.90	6.65	5.98
RANDCENTLB	11.72	9.46	7.09	4.26
SCOTCHLB	6.94	7.05	6.96	4.05

Source: The author

balancers, when the application call CG-ENERGYLB, it spend less time to adjust the clock frequency, which results in a reduction in total execution time and an equivalent saving energy. In this context, we analyzed the total energy spent with load balancing to each test performed and relate it with the total energy consumption on execution.

Figure 5.21 depicts the energy spend with load balancing (LB Energy) over total energy consumption for each selected real application (App Energy).

Figure 5.21: Energy spend with load balancing over total energy consumption for each real application.



(a) *Ondes3D*. (b) *Lulesh*.

Source: The author

In the tests with the *Ondes3D* and *Lulesh* applications, the load balancers are responsible in average by spent of 28.87% and 9.87% of total energy consumed during execution of the applications. When CG-ENERGYLB was used together with load balancers, it also reduces the load balancing energy. Thus, for *Ondes3D* and *Lulesh*, the energy spent with load balancing was reduced on average to 24.55% and 8.46% respectively.

5.3.1.2 Threshold Evaluation

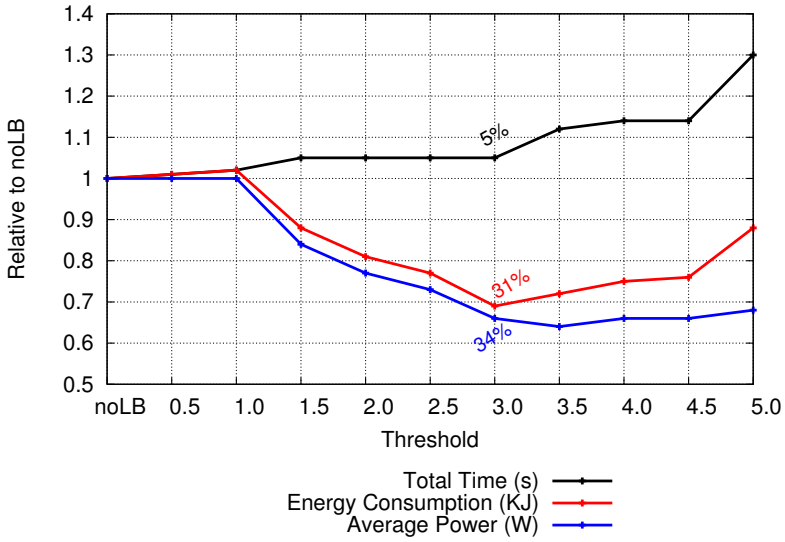
This section provides a CG-ENERGYLB evaluation over real world applications using different threshold values. In the previous section, the CG-ENERGYLB load balancer was analyzed with threshold value equal to 2.0. As mentioned in 5.2.2.2, this threshold value was adopted due that clock frequency range available on our experimental platform, allow us to vary the clock frequency of the processor from 1.2 GHz up to 2.4 GHz.

The use of different threshold values, incurs in different amounts of DVFS performed or load balancing called, which determines different frequency settings of cores or migration tasks. In this way, to analyze which is the best threshold value for each application under the total energy consumption, total execution time or the average power demand, we vary the threshold parameter of the algorithm from 0 up to 5.

- *Ondes3D* Application

Experiments with *Ondes3D* were performed using 1024 tasks mapped on 192 cores, which run 500 iterations each. Total energy spent to run this application without load balancer is 263.1 kJoules and its total execution time is 200.41 seconds. During this execution, the average power demand is 54.71 W. These values are taken as reference (NOLB in Figure 5.22) in our analysis.

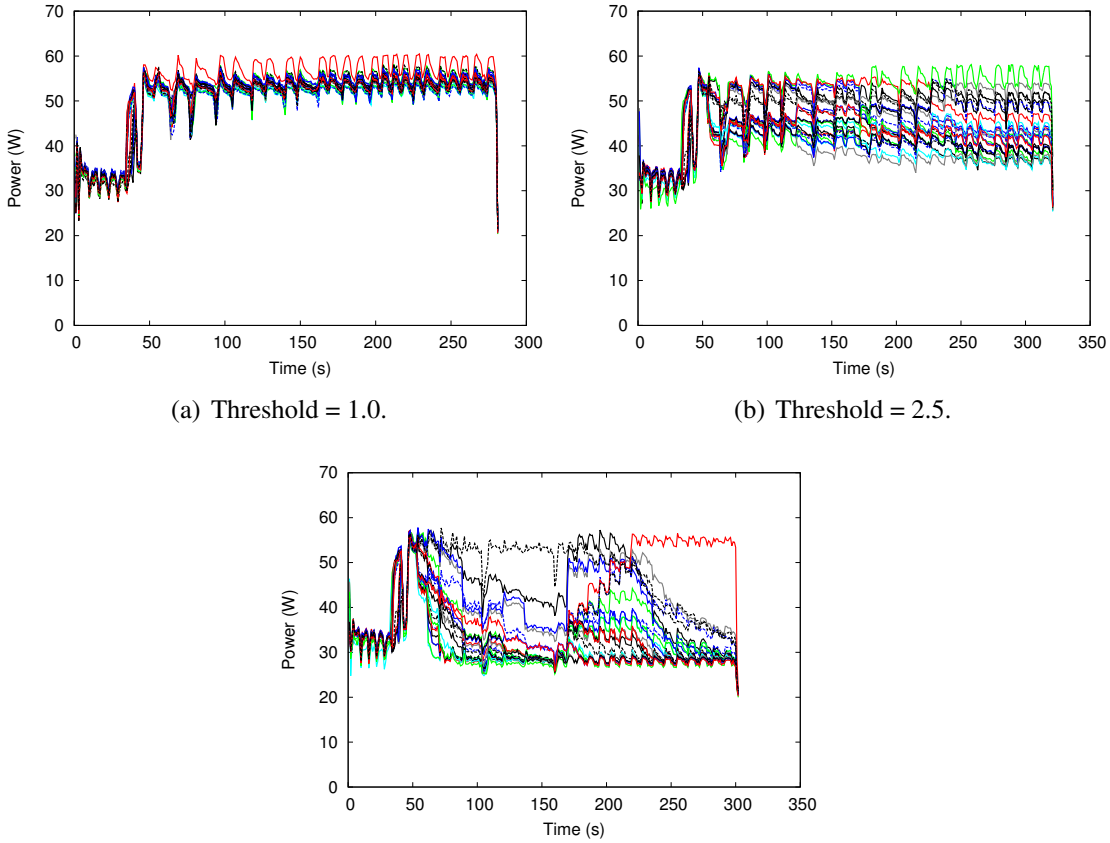
Figure 5.22: CG-ENERGYLB comparison with different threshold value on *Ondes3D*.



Source: The author

Figure 5.23(a) depicts the instantaneous power of the execution when CG-ENERGYLB uses threshold value equal to 1.0. In this execution any times DVFS is performed, in all the 24 load balancer calls tasks were migrated by REFINELB. In this way, during execu-

Figure 5.23: Power evaluation to different threshold value on *Ondes3D*



(a) Threshold = 1.0.

(b) Threshold = 2.5.

(c) Threshold = 5.0.

Source: The author

tion the power of all processors is always high, resulting in an average power of 54.7 W.

Using threshold equal to 2.5 the processors power is differently reduced as shown in the Figure 5.23(b). For this threshold, 17 times DVFS is performed and only 7 calls migrate tasks by REFINELB. This form, the power is reduced in great majority of the processors, which result in a total reduction of 16.08%, leaving the average power in 43.4 W.

A different amount of energy is saved when using threshold equal to 5.0 (Figure 5.23(c)). In this execution in all call (24) adjusts in clock were performed, leaving only one processor using its maximum power. For this threshold value, the power demand follows the increase of application needs, once the increases from the second 160 and reduces again from the second 212. This form, in this test the average power is reduced in 32.39%, resulting in an average of 35.0 W.

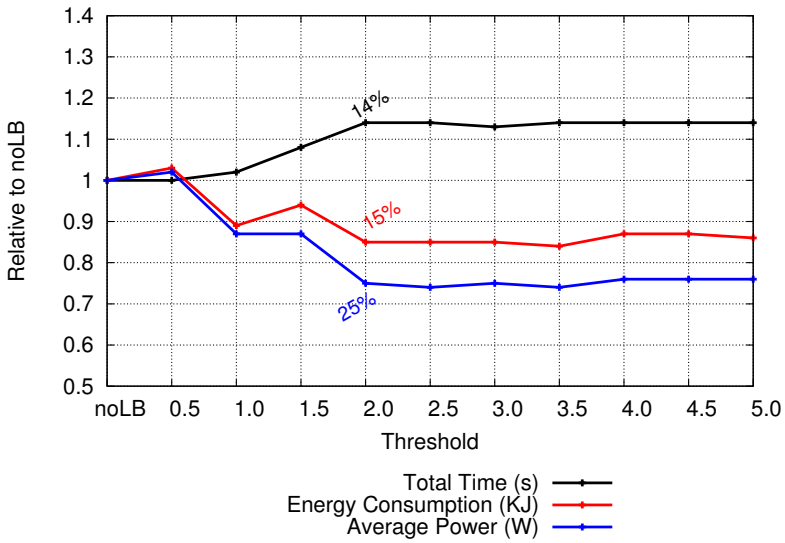
For *Ondes3D*, the least amount of energy spent is achieved using threshold value equal to 3.0. Using this value CG-ENERGYLB is able to reduce in up to 31% the total energy consumption in relation to baseline NOLB. This reduction is achieved through of

the reduction of the average power demand in 34%, which overcome the time overhead of 5%, as shown in the Figure 5.22.

- *Lulesh* Application

Lulesh was executed with 1000 iterations in each one of its 5832 processes mapped in 192 cores. This application spent 840.6 kJoules of energy and takes 688 seconds when executed without load balancer. Thus, in this execution the average power demand is 50.9 W. These values are taken as reference (baseline) and shown in column NOLB of the Figure 5.24) to examine the threshold variation of the *Lulesh* application.

Figure 5.24: CG-ENERGYLB comparison with different threshold value on *Lulesh*.



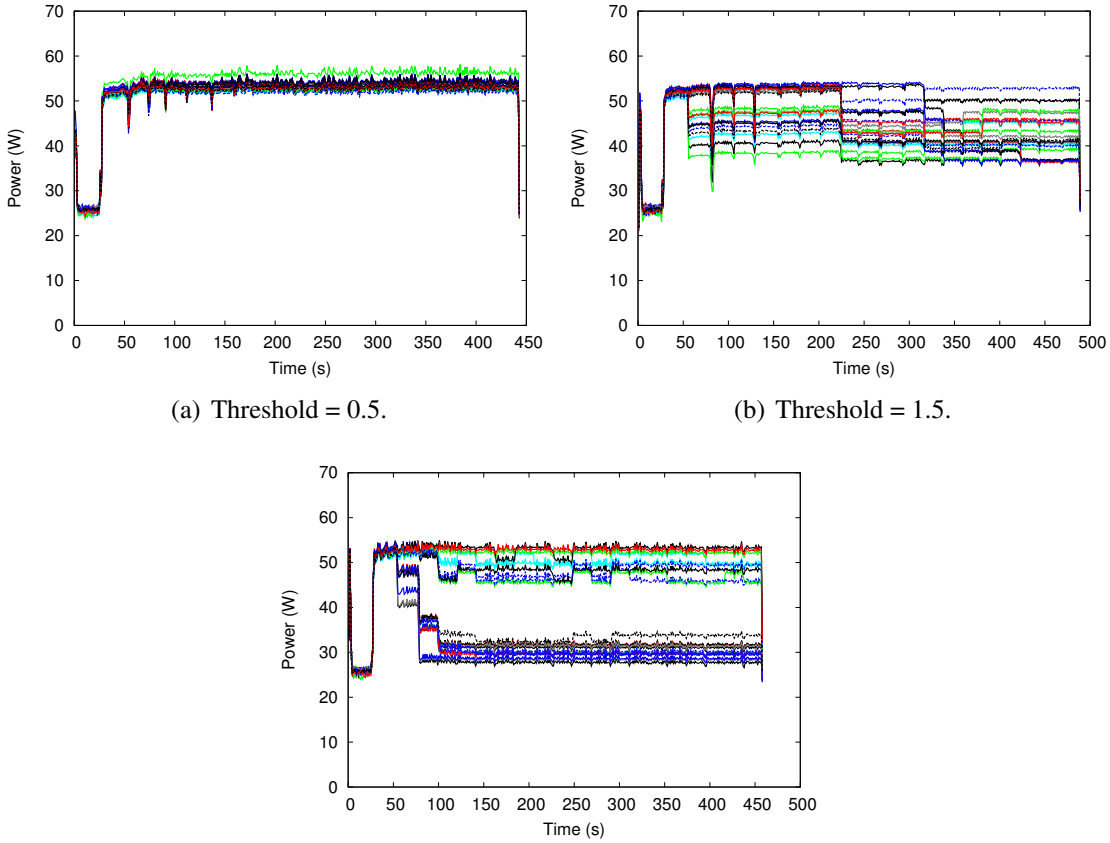
Source: The author

Instantaneous power measured when the application is executed with threshold equal to 0.5 is depicted in the Figure 5.25(a). Using this value, in all load balancing calls (19 times) were migrated tasks through do REFINELB. Similar to *Ondes3D* execution, for this threshold all processors running using a high power during all execution, which result in an average of power of 50.9 W.

On the other hand, using a threshold equal to 2.5, CG-ENERGYLB is able to reduce the power demand to intermediate levels as shown in the Figure 5.25(b). With this threshold, in this execution are performed 4 times DVFS, which reduced the power of the processors in 12.7%, to 43.4 W.

For threshold equal to 5.0 (Figure 5.25(c)) were adjusted 19 times the clock frequency of cores. This way, the power of most of the processors is reduced to minimum levels saving more energy. The total reduction was of 24%, which reduced the average power demand to 38.7 W.

Figure 5.25: Power evaluation to different threshold value on *Lulesh*



(a) Threshold = 0.5.

(b) Threshold = 1.5.

(c) Threshold = 5.0.

Source: The author

The threshold variation from 0.5 up to 2.0 present the reduction more significant in energy consumption for this application. Differently from *Ondes3D*, when used these threshold values in CG-ENERGYLB load balancer, the runtime increases in up to 14%, while that the average power demand reduces in up to 25%. In this way, the least amount of energy spent for *Lulesh*, is achieved using threshold value equal to 2.0. For this value, CG-ENERGYLB reduces the total energy consumption in up to 15% if compared to baseline NOLB. This reduction is achieved through of the reduction of the average power in 25%, which overcome an overhead of 14%, as shown in the Figure 5.24.

The results achieved show that our algorithms are able to mitigate residual imbalance left by load balancing algorithm. Our load balancing algorithm have improvements in both goals, runtime, power demand and energy consumption on experiments with benchmarks and real world applications. In the next section, we discuss more about the gains of our load balancers.

5.4 Overall Results

In this chapter, we presented an experimental evaluation of our energy-aware load balancing approaches. Firstly, in the methodology, we discuss the experimental environment, the benchmarks and real world applications that simulate natural phenomena, and state-of-the-art load balancers used in the evaluation. Finally, our two algorithms for our proposed ENERGYLB were tested using different characteristics of load imbalance and were able to mitigate load imbalance coming from load irregularity and dynamicity.

They usually have a reduction in power demand and in the amount of the energy spent while incurring in a low overhead. Our approaches can be applied alone or attached to other load balancing to save energy. Overall, the energy savings with ENERGYLB were up to 55.8% with our fine-grained algorithm that performs per-core DVFS when it is used over benchmarks and up to 23% when it is used over real applications.

Overall, the results obtained with FG-ENERGYLB alone over the benchmarks showed energy savings of up to 25% with an average of 10.7%. The obtained results pointed that FG-ENERGYLB load balancer used alone improve the energy consumption by 13%, 2.4%, 2.1% and 25.2% on average for *lb_test*, *kNeighbor*, *stencil 4D* and *ComprehensiveBench* respectively. Furthermore, these reductions are results of the use of DVFS strategy during the execution, which reduces the power demand in 12.8%, 3.4%, 3.8% and 25.1% respectively.

When used FG-ENERGYLB together with other load balancers over the residual imbalance the energy saving achieved were different for each test. The greater reduction on energy is achieved using different load balancers to each benchmark. For *lb_test* and *kNeighbor* when SCOTCHLB is applied alone, it achieves a reduction of 39% and 7.28%, respectively in the total energy consumption compared to the baseline. However, an even greater reduction is achieved when FG-ENERGYLB is used together with SCOTCHLB. In this test, the average power demand is also reduced, resulting so in a total energy saving of 44.2% and 25.48% respectively.

Different from other benchmarks, on *stencil 4D* the greatest amount of energy saving is achieved using FG-ENERGYLB alone. This load balancer has an overhead of only 1.73% and reduces the average power in 3.82% on average if compared to the baseline. This way, the total energy consumption is reduced in up to 2.16% for this benchmark. Due to large memory footprint present in this benchmark, its migrations are costly. This mode, all the load balancers tested increase the total execution time of this

benchmark.

A similar amount of energy is saving for the *ComprehensiveBench* benchmark when using GREEDYLB and REFINELB alone. These load balancers are able to reduce in up to 44.7% the total energy consumption in relation to baseline. Applying FG-ENERGYLB together with REFINELB the amount of energy saving achieved 55.8%.

In summary, the results obtained with FG-ENERGYLB together other load balancers over benchmark showed energy savings of up to 55.8% with an average of 41.83%.

On a final point on FG-ENERGYLB evaluation, we compare the results achieved with FG-ENERGYLB algorithm over real applications. The results obtained when FG-ENERGYLB load balancer is used alone over real application point improve of the energy consumption by 13.9%, 2.86% and 10.9% on average for *Ondes3D*, *Lulesh*, and *Lassen* respectively. These reductions are results of the use of DVFS during the execution, which reduces the power demand in 15%, 4.05%, and 11.48% respectively.

For *Ondes3D*, the greatest amount of energy saving is achieved with the REFINELB load balancer. Using REFINELB alone, the total energy is reduced in up to 14.78% compared to baseline. However, using FG-ENERGYLB together with REFINELB, the average power is reduced in 7.1% and the total execution time in 2.6% compared to REFINELB used alone. This way, the total of energy saving was up to 22.9%.

For *Lulesh*, the lowest energy consumption is achieved with SCOTCHLB and FG-ENERGYLB together. SCOTCHLB used alone reduces consumption by up to 4.44%. When FG-ENERGYLB is performed over the SCOTCHLB residual imbalance, the average power is reduced more 10.9%. Thus, the total energy consumption was reduced by up to 14.28% compared the baseline.

For *Lassen*, in all test the execution time, the power demand and the total energy consumption is reduced. The lower energy consumption is achieved with GREEDYLB, GREEDYCOMMLB, REFINELB, REFINECOMMLB. Using these load balancers alone 19.8% of energy was saving. In addition to the clock adjustment using the FG-ENERGYLB has a total energy reduction of 23% for GREEDYCOMMLB.

Finally, we summarize the runtime, power demand and energy consumption gains for each application considered using our hierarchical strategy. Using our coarse-grained algorithm that performs per-chip DVFS were achieved the energy savings up to 35.7%.

When CG-ENERGYLB load balancer is used alone over the real application the energy consumption is improved by 27%, 23.8% and 8.85% on average for *Ondes3D*, *Lulesh*, and *Lassen* respectively. These reductions come from of the use of DVFS during

the execution, which reduces the power demand in 27.4%, 25.5% and 9.17% respectively.

For *Ondes3D*, using REFINELB alone is achieved the greatest amount of energy saving, which represent 21.6% compared to baseline. However, using CG-ENERGYLB together with REFINELB and REFINECOMMLB, the average power is reduced up to 18.3%, which result in a reduction in total energy spent of 33.2% and 35.7% respectively.

For *Lulesh*, the lowest energy consumption was achieved with REFINELB. It reduces by up to 6.4% compared to baseline. However, using CG-ENERGYLB together with load balancers, the greatest reduction in energy consumption was achieved with SCOTCHLB and CG-ENERGYLB. The total energy consumption was reduced in 26.9% compared to baseline.

For *Lassen*, using REFINELB alone the energy saving is 11.5%. However, the lower energy consumption is achieved using GREEDYLB and CG-ENERGYLB together. In this test, the total energy consumption reduces 17.7% compared to baseline.

The processors of our experimental environment have 14 clock frequency levels available. This range of levels allow us to vary the clock frequency of the processor from 1.2 GHz up to 2.4 GHz, defining more power save or more power hungry according of the computational load. We realize that this processor feature is very important to our energy-aware approach, since that in test realized all available levels were used by our ENERGYLB, allow us a higher precision. For example, in the *Ondes3D* execution, Figure 5.8(b) of Section 5.2.2, the intermediate level of 2.2 GHz was used only in 3 timesteps, while that the level 1.5 GHz was used in 109 timesteps. So, having a higher precision information of computational load and using processors with a range of clock frequency levels, we can make more accurate decisions and save more energy.

6 CONCLUSIONS AND FUTURE WORK

HPC systems have increased in performance, number of processors and complexity. This growth also increased the power demand of these parallel platforms, with power becoming a critical aspect in the design of future Exascale systems. On the other hand, large and complex scientific applications increasingly demand for more performance. Scientific applications generally present dynamic or irregular characteristics that difficult the efficient use of HPC systems and contribute to reduce energy efficiency. In this context, to use all resources available in parallel machines and achieve a better performance with a high energy efficiency is important to analyze characteristics of both platform and application, and use this information to make load balancing decision.

Several load balancing strategies have been developed, aiming to improve the load distribution across parallel processors and increase application performance. Moreover, many load balancers take into account only computation load to make their decisions due the hard solutions to achieve a good balance, beyond introducing overhead to the total execution time of the application. These algorithms have left aside the power demand and total energy consumption of each task of the application. In this context, since processors are responsible for a large percentage of the total energy consumption of a parallel machine, managing processors power is crucial to save energy.

In this context, the main objective of this thesis focus on load balancers to achieve an efficient use of all available resources of parallel machines and to reduce the average power demand of systems, thus saving energy in the execution of imbalanced applications.

6.1 Contributions

The contributions of this thesis were based on the hypothesis that most load balancing strategies are mainly based on application characteristics and architectural aspects of the machine. Current load balancing approaches generally take into account several pieces of information, such as computational load, processor speed, communication cost and hardware topology. A few recent strategies started including decisions based on power and energy constrains.

In this context, we proposed ENERGYLB, a new energy-aware load balancer that exploits the existence of residual imbalances on iterative applications to adjust the clock frequency of underloaded cores/processors combining dynamic load balancing and DVFS

technique during the application execution. ENERGYLB considers characteristics of the platform, load irregularity and dynamicity of applications. Its strategy takes into account current computational load to make decision of calling other load balancing strategies that reduce load imbalance by migrating tasks, or it adjusts the clock frequencies of the cores according to their weighted loads.

In response to different processors architecture, that can feature two levels of granularity, per-chip DVFS or per-core DVFS, we introduced two different algorithms of our energy-aware load balancer. FG-ENERGYLB allows a fine control of the clock frequency of cores in systems that have few tens of cores and feature per-core DVFS control. On the other hand, CG-ENERGYLB is suitable for HPC platforms composed of several multi-core processors that do not allow such a fine-grain control, i.e., that only feature per-chip DVFS. Both algorithms were implemented in CHARM++, allowing it to be used together with existing load balancer available in CHARM++ without any application changes. We showed that applications implemented in CHARM++ can benefit from ENERGYLB to improve energy efficiency, reducing to power demand during the execution of imbalanced applications, without considerably degrading their overall performance.

We evaluated our energy-aware load balancers and demonstrate the improvements in energy savings. For this purpose, we ran benchmarks and real world applications and compared their energy spent, runtime and power demand with state-of-the-art load balancers.

Results obtained with the use of the *Fine-Grained EnergyLB (*FG-ENERGYLB*)* algorithm alone over benchmark present energy savings of up to 25.2% with an average of 10.7%. Nevertheless, some residual imbalance was still present after tasks were remapped. In this case, when FG-ENERGYLB was employed together with these load balancers, energy savings were achieved up to 55.8% with an average of 12.0%.

Our results also demonstrated that FG-ENERGYLB used alone is able to achieve energy savings of up to 13.9% with an average of 9.2% over real applications. However, greater improvements in energy are achieved using FG-ENERGYLB to mitigate residual imbalance over applications. In this way, the total energy save achieve is up to 23% with an average of 20.06%.

Considered the use of the *Coarse-Grained EnergyLB (*CG-ENERGYLB*)* algorithm alone over the real application, it improves the energy consumption up to 27% with an average of 19.88%. When CG-ENERGYLB is applied together other load balancers to mitigate the residual imbalance, it can achieve energy savings up to 35.7% with

an average of 26.76%.

The achieved results have shown that the adjustment of the clock frequency according to workload of each processing unit during the runtime has an impact on the average power demand and consequently on the total energy consumption. The results presented in this thesis show that by combining dynamic load balancing and DVFS considering the characteristics of the application and of the system to make decisions, we are able to reduce the total energy consumption on iterative applications.

6.2 Perspectives of Future Work

Our research has shown that the application of the two ENERGYLB algorithms proposed in this thesis are able to reduce the average power demand of system during the execution and also to reduce the total energy consumption. We have identified some opportunities to extend the contributions of this thesis. The main opportunities are highlighted in following:

1. **Inclusion of new heuristics in the decisions.** Most current strategies have used heuristic together with threshold to define the load imbalance. We evaluated the current versions of our energy-aware load balancers using clock frequency-workload product to define the residual imbalance after load balancer call, and threshold to determine if load balancer or DVFS must be used. However, as power demand constraints are an increased challenge for building of HPC systems, evaluation of the energy savings could to exploit with different metrics;
2. **Evaluate the use of predictability.** The current versions of the ENERGYLB load balancer consider both current characteristics of the platform and the current load of applications to make their decisions. However, to applications with irregularity and dynamicity making decisions based on past information can avoid redundant changes in clock frequency. In this way, evaluating the use of task load predictability during the execution could identify possible performance gains while reduce the DVFS use and also the overhead;

   In the same context, we realize that the impact of load balancing is directly related to the load balancing frequency and that load balancing overhead can overcome the gains achieved with load balancing. The use of ENERGYLB strategies result in a less load imbalance, once first load balancing strategies migrate task and after

ENERGYLB approach adjusts the clock considering the residual imbalance. However, when applications have a high dynamicity and the load balancing frequency is also high, the reduction in clock can incurs in performance degradation. In this way, combining predictability, less aggressive threshold and return the processor frequency to the original position could be evaluated to avoid this behavior; and

3. **Make decision improvements.** The current versions of the ENERGYLB load balancer perform task migrations between cores of the same processor and can migrate tasks between processors when needed, taking as base the computational load. In this scheme, decisions can also take into account the *cost of task migrations (time and energy)* between cores/processors that operate on different clock frequencies, thus reducing overhead of performing DVFS or call other load balancing on all processors at each load balancing step.

6.3 Published Papers

The following list shows the relevant papers regarding this thesis that were published since entering the PhD program, ordered by relevance:

- *E. L. Padoin*, L. L. Pilla, M. Castro, F. Z. Boito, P. O. A. Navaux, and J.-F. Méhaut, **Performance/energy trade-off in scientific computing: The case of ARM big.LITTLE and Intel Sandy Bridge**, IET Computers & Digital Techniques, vol. 9, no. 1, pp. 27–35, 2015.
- *E. L. Padoin*, L. L. Pilla, F. Z. B. Boito, R. V. Kassick, P. Velho, and P. O. A. Navaux, **Evaluating application performance and energy consumption on hybrid CPU+GPU architecture**, Cluster Computing, vol. 16, no. 3, pp. 511–525, 2013, 10.1007/s10586-012-0219-6.
- *E. L. Padoin*, M. Castro, L. L. Pilla, P. O. A. Navaux, and J.-F. Méhaut, **Saving energy by exploiting residual imbalances on iterative applications**, in 21st International Conference on High Performance Computing (HiPC), Goa, India, 2014, pp. 1–10.
- *E. L. Padoin*, P. Velho, D. A. G. de Oliveira, P. O. A. Navaux, and J.-F. Méhaut, **Tuning performance and energy consumption of HPC applications on ARM MPSoCs**, in 5th Workshop on Applications for Multi-Core Architectures (WAMCA), Paris, France, 2014, pp. 1–6.

- *E. L. Padoin*, D. A. G. de Oliveira, P. Velho, P. O. A. Navaux, and J.-F. Méhaut, **ARM-based cluster: Performance, scalability and energy efficiency**, in 4th Workshop on Applications for Multi-Core Architectures (WAMCA), Porto de Galinhas, PB, Brasil, 2013, pp. 1–6.
- *E. L. Padoin*, D. A. G. de Oliveira, P. Velho, and P. O. A. Navaux, **Time-to-solution and energy-to-solution: A comparison between ARM and Xeon**, in Third Workshop on Applications for Multi-Core Architectures (WAMCA), New York, USA, 2012, pp. 48–53.
- *E. L. Padoin*, D. A. G. Oliveira, P. Velho, and P. O. A. Navaux, **Evaluating the performance and energy of ARM processors for high performance computing**, in 41st International Conference on Parallel Processing Workshop (ICPPW - UCAA) 2012, pp. 165-172.
- *E. L. Padoin*, L. L. Pilla, M. Castro, P. O. A. Navaux, and J.-F. Méhaut, **Energy Consumption Reduction through Load Balancing and DVFS on Residually Imbalanced Cores**. In: 2nd Workshop of the Joint-Laboratory on Extreme Scale Computing (JLESC), Chicago, USA. 2014. p.1.
- *E. L. Padoin*, L. L. Pilla, M. Castro, P. O. A. Navaux, and J.-F. Méhaut, **Saving energy by exploiting residual imbalances on iterative applications**, in 11th Workshop Joint-Laboratory on Petascale Computing (JLPC) 1st Workshop of the Joint-Laboratory on Extreme Scale Computing (JLESC), Sophia Antipolis, France, 2014, pp. 1.
- *E. L. Padoin*, L. L. Pilla, M. Castro, P. O. A. Navaux, and J.-F. Méhaut, **Saving energy by exploiting residual imbalances on iterative applications**, in 12th Workshop on Charm++ and its Applications. University of Illinois at Urbana Champaign (UIUC), Urbana, USA, 2014, pp. 1–3.
- *E. L. Padoin*, E. H. M. Cruz, F. Z. Boito, F. B. MOREIRA, L. L. Pilla, and P. O. A. Navaux, **Avaliação de um modelo de consumo energético aplicado ao mapeamento de processos**, in XXXI Congresso da Sociedade Brasileira de Computação (CSBC) - X Workshop em Desempenho de Sistemas Computacionais e de Comunicação (WPERFORMANCE). SBC, 2011, pp. 2075–2085.
- E. H. de Oliveira, J. X. S. Jr, *E. L. Padoin*, and P. O. A. Navaux, **Utilização de aceleradores embarcados de baixo consumo na implementação de sistemas de HPC**, in XVI Simpósio de Sistemas Computacionais (WSCAD-SSC), Florianópo-

lis, SC, 2015, pp. 1–8.

- G. Arruda, *E. L. Padoin*, L. L. Pilla, P. O. A. Navaux, and J.-F. Méhaut, **Proposta de balanceamento de carga para redução de migração de processos em ambientes multiprogramados**, in XVI Simpósio de Sistemas Computacionais (WSCAD-WIC), Florianópolis, RJ, 2015, pp. 1–8.
- L. dos Santos Eitelvein, *E. L. Padoin*, and P. O. A. Navaux, **Avaliando a eficiência energética do grid5000 com o benchmark HPL**, in XIII Simpósio de Sistemas Computacionais (WSCAD-SSC), Petrópolis, RJ, 2012, pp. 1–4.
- *E. L. Padoin*, P. Velho, D. A. G. de Oliveira, P. O. A. Navaux, B. Videau, A. Degomme, and J.-F. Méhaut, **Análise de desempenho, escalabilidade e eficiência energética de mpsocs com processadores ARM**, in Conferencia Latino Americana de Computación de Alto Rendimiento (CLCAR), San José, Costa Rica, 2013, pp. 1–10.
- P. Velho, D. A. G. de Oliveira, *E. L. Padoin*, M. Castro, and P. O. A. Navaux, **Fast and accurate models for GPU applications**, in Conferencia Latino Americana de Computación de Alto Rendimiento (CLCAR), San José, Costa Rica, 2013, pp. 1–9.
- *E. L. Padoin*, D. A. G. de Oliveira, P. Velho, E. R. Rodrigues, R. F. Pires, and P. O. A. Navaux, **Análise de processadores ARM em cluster visando a eficiência energética para computação de alto desempenho**, in Conferencia Latino Americana de Computación de Alto Rendimiento (CLCAR), Ciudad de Panamá, Panamá, 2012, pp. 1–7.
- *E. L. Padoin*, L. L. Pilla, F. Z. Boito, R. V. Kassick, and P. O. A. Navaux, **Análise do consumo energético do algoritmo de irrigação de solos em arquiteturas heterogêneas**, in Conferencia Latino Americana de Computación de Alto Rendimiento (CLCAR), 2011, pp. 1–7.

REFERENCES

ALVES, M. A. Z. et al. Enhancing energy efficiency using efficient parallel programming techniques. In: CONFERENCIA LATINO AMERICANA DE COMPUTACION DE ALTO RENDIMENTO (CLCAR 2010), Gramado, RS. **Proceedings...** [S.l.]: UFRGS, PUCRS, UFPel, 2010. p. 117–124.

AMD. **AMD Family 15h Processor BIOS and Kernel Developer Guide, rev 3.14 edition**. [S.l.], 2013. v. 3.

AUPY, G.; BENOIT, A.; ROBERT, Y. Energy-aware scheduling under reliability and makespan constraints. In: INTERNATIONAL CONFERENCE ON HIGH PERFORMANCE COMPUTING (HIPC), Goa, India. **Proceedings...** [S.l.]: IEEE Computer Society, 2012. p. 1–10.

BAHI, J. M.; CONTASSOT-VIVIER, S.; COUTURIER, R. Dynamic load balancing and efficient load estimators for asynchronous iterative algorithms. **Parallel and Distributed Systems, IEEE Transactions on**, IEEE, v. 16, n. 4, p. 289–299, April 2005. ISSN 1045-9219.

BARKER, K. et al. Using performance modeling to design large-scale systems. **Computer**, v. 42, n. 11, p. 42–49, 2009. ISSN 0018-9162.

BECKMAN, P. et al. On the Road to Exascale. **Scientific Computing World**, Europa Science Ltd., n. 116, p. 26–28, 2011.

BHATELÉ, A. et al. Architectural constraints to attain 1 exaflop/s for three scientific application classes. In: INTERNATIONAL PARALLEL & DISTRIBUTED PROCESSING SYMPOSIUM, Washington, DC, USA. **Proceedings...** [S.l.]: IEEE Computer Society, 2011. (IPDPS '11), p. 80–91. ISBN 978-0-7695-4385-7.

BHATELÉ, A.; KALÉ, L. V.; KUMAR, S. Dynamic topology aware load balancing algorithms for molecular dynamics applications. In: INTERNATIONAL CONFERENCE ON SUPERCOMPUTING (ICS), Yorktown Heights, NY, USA. **Proceedings...** [S.l.]: ACM, 2009. p. 110–116. ISBN 978-1-60558-498-0.

BHATELÉ, A. et al. Overcoming scaling challenges in biomolecular simulations across multiple platforms. In: INTERNATIONAL SYMPOSIUM ON PARALLEL AND DISTRIBUTED PROCESSING (IPDPS), Miami, Florida, USA. **Proceedings...** [S.l.]: IEEE, 2008. p. 1–12. ISSN 1530-2075.

BRAMS. **Brazilian developments on the Regional Atmospheric Modelling System**. 2011. Available from Internet: <http://brams.cptec.inpe.br/>. Accessed: 2011.01.15.

BRODOWSKI, D. **CPU frequency and voltage scaling code in the Linux(TM) kernel**. [S.l.], 2014. Accessed: 2014.05.18. Available from Internet: <www.kernel.org>.

BRUCKER, P.; BRUCKER, P. **Scheduling algorithms**. 5th. ed. [S.l.]: Springer-Verlag New York, Inc., 2007. 371 p. ISBN 978-3-540-69516-5.

CAMERON, K. A Tale of Two Green Lists. **Computer**, v. 43, n. 9, p. 86 –88, sept. 2010. ISSN 0018-9162.

CARIÑO, R. L.; BANICESCU, I. Dynamic load balancing with adaptive factoring methods in scientific applications. **The Journal of Supercomputing**, Springer, v. 44, n. 1, p. 41–63, 2008.

CAVALHEIRO, G. et al. Anahy: A programming environment for cluster computing. In: **High Performance Computing for Computational Science - VECPAR 2006**. [S.l.]: Springer Berlin Heidelberg, 2007, (Lecture Notes in Computer Science, v. 4395). p. 198–211. ISBN 978-3-540-71350-0.

CHANDRAKASAN, A.; SHENG, S.; BRODERSEN, R. Low-power cmos digital design. **Solid-State Circuits, IEEE Journal of**, IEEE, v. 27, n. 4, p. 473–484, 1992. ISSN 0018-9200.

CHANG, J.-M.; PEDRAM, M. Energy minimization using multiple supply voltages. In: LOW POWER ELECTRONICS AND DESIGN, Monterey, CA. **Proceedings...** [S.l.], 1996. p. 157–162.

CHARM++. **Parallel Programming Laboratory (PPL)**. 2014. Available from Internet: <http://charm.cs.illinois.edu/>. Accessed: 2014.21.10.

CYBENKO, G. Dynamic load balancing for distributed memory multiprocessors. **Journal of parallel and distributed computing**, Elsevier, v. 7, n. 2, p. 279–301, 1989.

DIKAIAKOS, M. D.; STADEL, J. A performance study of cosmological simulations on message-passing and shared-memory multiprocessors. In: INTERNATIONAL CONFERENCE ON SUPERCOMPUTING (ISC), Philadelphia, PE, USA. **Proceedings...** [S.l.]: ACM, 1996. p. 94–101.

DONG, Y.; CHEN, J.; TANG, T. Power measurements and analyses of massive object storage system. In: INTERNATIONAL CONFERENCE ON COMPUTER AND INFORMATION TECHNOLOGY (CIT), Bradford, UK. **Proceedings...** [S.l.]: IEEE Computer Society, 2010. p. 1317–1322.

DONGARRA, J.; MEUER, H.; STROHMAIER, E. **TOP500 Supercomputer Sites**. 2014. Available from Internet: <http://www.top500.org>. Accessed: 2014.10.10.

DORSEY, J. et al. An integrated quad-core opteron processor. In: INTERNATIONAL SOLID-STATE CIRCUITS CONFERENCE (ISSCC). **Proceedings...** [S.l.]: IEEE, 2007. p. 102–103. ISSN 0193-6530.

DOSANJH, S. et al. Exascale design space exploration and co-design. **Future Generation Computer Systems**, Elsevier, v. 30, p. 46–58, 2014.

DUPROS, F. et al. Exploiting intensive multithreading for the efficient simulation of 3d seismic wave propagation. In: INTERNATIONAL CONFERENCE ON COMPUTATIONAL SCIENCE AND ENGINEERING, Perugia, Italy. **Proceedings...** [S.l.]: IEEE, 2008. p. 253–260.

EIA. **U.S. Energy Information**. 2012. Available from Internet: <http://www.eia.doe.gov/emeu/iea/wecbtu.html>. Accessed: 2012.04.08.

EL-REWINI, H.; ABD-EL-BARR, M. **Advanced computer architecture and parallel processing**. [S.l.]: Wiley-Interscience, 2005. 288 p. ISBN 0471467405.

ENERGIA, M. de Minas e. **Projeção da demanda de energia elétrica para os próximos 10 anos (2011-2020)**. [S.l.], 2011. 1–1055 p. Accessed: 2011.07.12. Available from Internet: <www.epe.gov.br>.

FAURE, F. et al. Sofa: A modular yet efficient simulation framework. In: SURGETICA 2007 COMPUTER-AIDED MEDICAL INTERVENTIONS: TOOLS AND APPLICATIONS, California, USA. **Proceedings...** 2007. p. 101–108. Available from Internet: <http://www-evasion.imag.fr/Publications/2007/FACNBDDG07>.

FENG, W.; CAMERON, K. The green500 list: Encouraging sustainable supercomputing. **Computer**, IEEE, v. 40, n. 12, p. 50–55, 2007.

FENG, W.; LIN, H. The Green500 List Year Two. In: INTERNATIONAL PARALLEL AND DISTRIBUTED PROCESSING WORKSHOPS (IPDPSW), Atlanta, Georgia, USA. **Proceedings...** [S.l.]: IEEE, 2010. p. 1–8.

FENG, X.; GE, R.; CAMERON, K. W. Power and energy profiling of scientific applications on distributed systems. In: INTERNATIONAL CONFERENCE ON PERFORMANCE ENGINEERING, Denver, Colorado. **International Parallel and Distributed Processing Symposium (IPDPS)**. [S.l.]: IEEE, 2005. p. 34–34.

FOWLER, R. **Eletricidade: princípios e aplicações**. [S.l.]: Makron Books do Brasil, 1992. 537 p.

FREEH, V. W.; LOWENTHAL, D. K. Using multiple energy gears in mpi programs on a power-scalable cluster. In: SYMPOSIUM ON PRINCIPLES AND PRACTICE OF PARALLEL PROGRAMMING (SIGPLAN), Chicago, IL, USA. **Proceedings...** [S.l.]: ACM, 2005. p. 164–173. ISBN 1-59593-080-9.

GALINDO, I.; ALMEIDA, F.; BADÍA-CONTELLES, J. M. Dynamic load balancing on dedicated heterogeneous systems. In: RECENT ADVANCES IN PARALLEL VIRTUAL MACHINE AND MESSAGE PASSING INTERFACE, Dublin, Ireland. **Proceedings...** [S.l.]: Springer-Verlag, 2008. p. 64–74. ISBN 978-3-540-87474-4.

GAUTIER, T.; BESSERON, X.; PIGEON, L. KAAPI: A thread scheduling runtime system for data flow computations on cluster of multi-processors. In: INTERNATIONAL WORKSHOP ON PARALLEL SYMBOLIC COMPUTATION, London, Ontario, Canada. **Proceedings...** [S.l.]: ACM, 2007. p. 15–23. ISBN 978-1-59593-741-4.

GE, R. et al. Power Measurement Tutorial for the Green500 List. **The Green500 List: Environmentally Responsible Supercomputing.**, p. 1–6, 2007.

GERARDS, M. E. et al. Analytic clock frequency selection for global DVFS. In: EUROMICRO INTERNATIONAL CONFERENCE ON PARALLEL, DISTRIBUTED, AND NETWORK-BASED PROCESSING (PDP), Torino, It. **Proceedings...** [S.l.], 2014. p. 512–519. ISSN 1066-6192.

GOEL, B. et al. Portable, scalable, per-core power estimation for intelligent resource management. In: INTERNATIONAL GREEN COMPUTING CONFERENCE (IGCC), Chicago, IL. **Proceedings...** [S.l.]: IEEE Computer Society, 2010. p. 135–146.

HARTOG, J.; DEDE, E.; GOVINDARAJU, M. Mapreduce framework energy adaptation via temperature awareness. **Cluster Computing**, Springer, v. 17, n. 1, p. 111–127, 2013. Available from Internet: <http://dx.doi.org/10.1007/s10586-013-0270-y>.

HERNANDEZ, C.; SILLA, F.; DUATO, J. Energy and performance efficient thread mapping in noc-based cmps under process variations. In: PARALLEL PROCESSING (ICPP), 2011 INTERNATIONAL CONFERENCE ON, Taipei, China. **Proceedings...** [S.l.], 2011. p. 41–50. ISSN 0190-3918.

HEWITT, P. **Fisica Conceitual**. Porto Alegre: Bookman, 2002. 685 p. ISBN 9788536300405. Available from Internet: <https://books.google.com.br/books?id=Znu-BsJO-agC>.

HIRSCH, C. **Numerical Computation of Internal and External Flows: The Fundamentals of Computational Fluid Dynamics**. [S.l.]: Butterworth-Heinemann, 2007. 680 p. ISBN 9780080550022.

HOFMEYR, S. et al. Juggle: proactive load balancing on multicore computers. In: INTERNATIONAL SYMPOSIUM ON HIGH PERFORMANC DISTRIBUTED COMPUTING (HPDC), San Jose, California, USA. **Proceedings...** ACM, 2011. p. 3–14. ISBN 978-1-4503-0552-5. Available from Internet: <http://doi.acm.org/10.1145/1996130.1996134>.

HOSSEINIMOTLAGH, S.; KHUNJUSH, F.; HOSSEINIMOTLAGH, S. A cooperative two-tier energy-aware scheduling for real-time tasks in computing clouds. In: EUROMICRO INTERNATIONAL CONFERENCE ON PARALLEL, DISTRIBUTED, AND NETWORK-BASED PROCESSING (PDP), Torino, It. **Proceedings...** [S.l.]: IEEE, 2014. p. 178–182. ISSN 1066-6192.

HSU, C.; FENG, W. A power-aware run-time system for high-performance computing. In: SUPERCOMPUTING CONFERENCE (SC), Seattle, WA. **Proceedings...** [S.l.]: IEEE Computer Society, 2005. p. 1–9.

HSU, C.-H.; FENG, W. chun; ARCHULETA, J. S. Towards efficient supercomputing: A quest for the right metric. In: INTERNATIONAL PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM (IPDPS), Denver, Colorado. **Proceedings...** [S.l.]: IEEE, 2005. p. 8–pp.

HUANG, C.; LAWLOR, O.; KALÉ, L. V. Adaptive mpi. In: RAUCHWERGER, L. (Ed.). **Languages and Compilers for Parallel Computing**. Springer Berlin Heidelberg, 2004, (Lecture Notes in Computer Science, v. 2958). p. 306–322. ISBN 978-3-540-21199-0. Available from Internet: <http://dx.doi.org/10.1007/978-3-540-24644-2_20>.

HUMMEL, S. F. et al. Load-sharing in heterogeneous systems via weighted factoring. In: SYMPOSIUM ON PARALLEL ALGORITHMS AND ARCHITECTURES, Padua, Italy. **Proceedings...** ACM, 1996. p. 318–328. ISBN 0-89791-809-6. Available from Internet: <http://doi.acm.org/10.1145/237502.237576>.

ICHIKAWA, S.; YAMASHITA, S. Static load balancing of parallel pde solver for distributed computing environment. In: INTERNATIONAL CONF. PARALLEL AND DISTRIBUTED COMPUTING SYSTEMS (PDCS), Las Vegas, Nevada, USA. **Proceedings...** [S.l.]: ISCA Press, 2000. p. 399–405. ISBN 1-880843-34-X.

INPE. **Instituto Nacional de Pesquisas Espaciais**. 2010. Available from Internet: <http://www.plasma.inpe.br>. Accessed: 2010.03.17.

INTEL. **Intel Architecture Software Developer's Manual, System Programming Guide**. [S.l.], 2009. v. 3.

IPMI. **Intelligent Platform Management Interface**. 2013. Available from Internet: <http://www.gnu.org/software/freeipmi>. Accessed: 2013.07.05.

ISCI, C. et al. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In: INTERNATIONAL SYMPOSIUM ON MICROARCHITECTURE (MICRO), Orlando, FL. **Proceedings...** [S.l.]: IEEE Computer Society, 2006. p. 347–358. ISSN 1072-4451.

JETLEY, P. et al. Massively parallel cosmological simulations with ChaNGa. In: INTERNATIONAL SYMPOSIUM ON PARALLEL AND DISTRIBUTED PROCESSING (IPDPS), Miami, FL. **Proceedings...** [S.l.]: IEEE, 2008. p. 1–12. ISSN 1530-2075.

JOHNSON, D.; HILBURN, J.; JOHNSON, J. **Fundamentos de análise de circuitos elétricos**. [S.l.]: LTC Editora, 2000. 537 p. ISSN 231881892.

KALÉ, L. The virtualization approach to parallel programming: Runtime optimizations and the state of the art. In: LOS ALAMOS COMPUTER SCIENCE INSTITUTE SYMPOSIUM (LACSI), Los Alamos, USA. **Proceedings...** [S.l.], 2002.

KALÉ, L.; BHANDARKAR, M.; BRUNNER, R. Load balancing in parallel molecular dynamics. In: FERREIRA, A. et al. (Ed.). **Solving Irregularly Structured Problems in Parallel**. Springer Berlin Heidelberg, 1998, (Lecture Notes in Computer Science, v. 1457). p. 251–261. ISBN 978-3-540-64809-3. Available from Internet: <http://dx.doi.org/10.1007/BFb0018544>.

KALÉ, L. V. et al. Programming Petascale Applications with Charm++ and AMPI. In: BADER, D. (Ed.). [S.l.]: Chapman & Hall / CRC Press, 2008. p. 421–441.

KALÉ, L. V.; KRISHNAN, S. CHARM++: A portable concurrent object oriented system based on C++. In: ANNUAL CONFERENCE ON OBJECT-ORIENTED PROGRAMMING SYSTEMS, LANGUAGES, AND APPLICATIONS (OOPSLA). **Proceedings...** [S.l.]: ACM, 1993. p. 91–108. ISBN 0-89791-587-9.

KARLIN, I. et al. **LULESH Programming Model and Performance Ports Overview**. [S.l.], 2012. 1-17 p. Available from Internet: <http://www.osti.gov/scitech/servlets/purl/1059462>.

KARLIN, I. et al. Exploring traditional and emerging parallel programming models using a proxy application. In: 27TH IEEE INTERNATIONAL PARALLEL & DISTRIBUTED PROCESSING SYMPOSIUM (IEEE IPDPS 2013), Boston, USA. **Proceedings...** [S.l.], 2013.

KAXIRAS, S.; MARTONOSI, M. Computer architecture techniques for power-efficiency. **Synthesis Lectures on Computer Architecture**, Morgan & Claypool Publishers, v. 3, n. 1, p. 1–207, 2008.

KIM, S.-g. et al. Energy-centric DVFS controlling method for multi-core platforms. In: HIGH PERFORMANCE COMPUTING, NETWORKING, STORAGE AND ANALYSIS (SCC), Salt Lake City, UT. **Proceedings...** [S.l.]: IEEE Computer Society, 2012. p. 685–690.

KODAMA, Y. et al. Imbalance of cpu temperatures in a blade system and its impact for power consumption of fans. **Cluster Computing**, Springer, v. 16, n. 1, p. 27–37, 2013. ISSN 1573-7543. Available from Internet: <http://dx.doi.org/10.1007/s10586-011-0174-7>.

KOGGE, P. et al. **Exascale Computing Study: Technology Challenges in achieving Exascale Systems**. 2008. 1–297 p. Available from Internet: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.165.6676>.

KRISHNA, C.; LEE, Y.-H. Voltage-clock-scaling adaptive scheduling techniques for low power in hard real-time systems. In: REAL-TIME TECHNOLOGY AND APPLICATIONS SYMPOSIUM (RTAS), Washington, DC. **Proceedings...** [S.l.], 2000. p. 156–165. ISSN 1080-1812.

LAROS, J. et al. Topics on measuring real power usage on high performance computing platforms. In: INTERNATIONAL CONFERENCE ON CLUSTER COMPUTING AND WORKSHOPS (ICCC), New Orleans, LA. **Proceedings...** [S.l.], 2009. p. 1–8. ISSN 1552-5244.

LEE, W. Y. Energy-saving dvfs scheduling of multiple periodic real-time tasks on multi-core processors. In: INTERNATIONAL SYMPOSIUM ON DISTRIBUTED SIMULATION AND REAL TIME APPLICATIONS, Singapore. **Proceedings...** [S.l.]: IEEE Computer Society, 2009. p. 216–223.

LEGRAND, A. et al. Mapping and load-balancing iterative computations. **Parallel and Distributed Systems, IEEE Transactions on**, IEEE, v. 15, n. 6, p. 546–558, 2004.

LEUNG, J. Y. T. **Handbook of scheduling: algorithms, models, and performance analysis**. [S.l.]: Chapman & Hall/CRC, 2004. 1120 p. ISBN 978-1584883975.

LIFFLANDER, J.; KRISHNAMOORTHY, S.; KALÉ, L. V. Work stealing and persistence-based load balancers for iterative overdecomposed applications. In: INTERNATIONAL SYMPOSIUM ON HIGH-PERFORMANCE PARALLEL AND DISTRIBUTED COMPUTING (HPDC), Delft, The Netherlands. **Proceedings...** ACM, 2012. p. 137–148. ISBN 978-1-4503-0805-2. Available from Internet: <http://doi.acm.org/10.1145/2287076.2287103>.

LIM, M. Y.; FREEH, V. W.; LOWENTHAL, D. K. Adaptive, transparent frequency and voltage scaling of communication phases in MPI programs. In: SUPERCOMPUTERS CONFERENCE (SC), Tampa, FL. **Proceedings...** [S.l.]: ACM/IEEE, 2006. p. 14–14.

LIM, M. Y.; FREEH, V. W.; LOWENTHAL, D. K. Adaptive, transparent CPU scaling algorithms leveraging inter-node MPI communication regions. **Parallel Computing**, Elsevier, v. 37, n. 10, p. 667–683, 2011.

MARTÍNEZ, J. A. et al. Automatic tuning of iterative computation on heterogeneous multiprocessors with ADITHE. **The Journal of Supercomputing**, Springer, v. 58, n. 2, p. 151–159, 2011.

MCCANDLESS, B. **Lassen mini-app**. 2015. Available from Internet: <https://codesign.llnl.gov/lassen.php>. Accessed: 2015.30.09.

MEI, C. et al. Enabling and scaling biomolecular simulations of 100 million atoms on petascale machines with a multicore-optimized message-driven runtime. In: PROCEEDINGS OF 2011 INTERNATIONAL CONFERENCE FOR HIGH PERFORMANCE COMPUTING, NETWORKING, STORAGE AND ANALYSIS, New York, NY, USA. **Proceedings...** ACM, 2011. (SC '11), p. 61:1–61:11. ISBN 978-1-4503-0771-0. Available from Internet: <http://doi.acm.org/10.1145/2063384.2063466>.

MEI, C. et al. Optimizing a parallel runtime system for multicore clusters: a case study. In: TERAGRID CONFERENCE (TG), Pittsburgh, Pennsylvania. **Proceedings...** [S.l.]: ACM, 2010. p. 12.

MENON, H. et al. Thermal aware automated load balancing for HPC applications. In: INTERNATIONAL CONFERENCE ON CLUSTER COMPUTING (CLUSTER), Indianapolis, IN. **Proceedings...** [S.l.]: IEEE Computer Society, 2013. p. 1–8.

MENON, H. et al. Automated load balancing invocation based on application characteristics. In: IEEE INTERNATIONAL CONFERENCE ON CLUSTER COMPUTING (CLUSTER), Beijing, China. **Proceedings...** [S.l.]: IEEE Computer Society, 2012. p. 373–381.

MENON, H.; KALÉ, L. A distributed dynamic load balancer for iterative applications. In: INTERNATIONAL CONFERENCE FOR HIGH PERFORMANCE COMPUTING, NETWORKING, STORAGE AND ANALYSIS (SC), Denver, CO. **Proceedings...** [S.l.]: ACM, 2013. p. 15.

MERKEL, A.; BELLOSA, F. Balancing power consumption in multiprocessor systems. **Operating Systems Review (SIGOPS)**, ACM, v. 40, n. 4, p. 403–414, 2006.

MICHALAKES, J.; VACHHARAJANI, M. Gpu Acceleration of Numerical Weather Prediction. **Parallel Processing Letters**, v. 18, n. 04, p. 1–8, 2008. ISSN 0129-6264. Available from Internet: <http://www.worldscinet.com/ppl/18/1804/S0129626408003557.html>.

NELSON, M. T. et al. NAMD: a parallel, object-oriented molecular dynamics program. **International Journal of High Performance Computing Applications**, SAGE Publications, v. 10, n. 4, p. 251–268, 1996.

PADOIN, E. et al. Saving energy by exploiting residual imbalances on iterative applications. In: HIGH PERFORMANCE COMPUTING (HIPC), 21ST INTERNATIONAL CONFERENCE ON, Goa, India. **Proceedings...** [S.l.], 2014. p. 1–10.

PADOIN, E. et al. Performance/energy trade-off in scientific computing: the case of arm big.little and intel sandy bridge. **Computers Digital Techniques, IET**, v. 9, n. 1, p. 27–35, 2015. ISSN 1751-8601.

PADOIN, E. L. et al. ARM-based cluster: Performance, scalability and energy efficiency. In: 4TH WORKSHOP ON APPLICATIONS FOR MULTI-CORE ARCHITECTURES (WAMCA), Porto de Galinhas, PB, Brasil. **Proceedings...** [S.l.], 2013. p. 1–6.

PADOIN, E. L. et al. Evaluating application performance and energy consumption on hybrid CPU+GPU architecture. **Cluster Computing**, Springer US, v. 16, n. 3, p. 511–525, 2013. ISSN 1386-7857. 10.1007/s10586-012-0219-6.

PADOIN, E. L. et al. Energy consumption reduction through load balancing and dvfs on residually imbalanced cores. In: 2ND WORKSHOP OF THE JOINT-LABORATORY ON EXTREME SCALE COMPUTING (JLESC), Chicago, USA. **Proceedings...** [S.l.], 2014. p. 1.

PADOIN, E. L. et al. Saving energy by exploiting residual imbalances on iterative applications. In: 11TH WORKSHOP JOINT-LABORATORY ON PETASCALE COMPUTING (JLPC) 1ST WORKSHOP OF THE JOINT-LABORATORY ON EXTREME SCALE COMPUTING (JLESC), Sophia Antipolis, France. **Proceedings...** [S.l.], 2014. p. 1–3.

PADOIN, E. L. et al. Saving energy by exploiting residual imbalances on iterative applications. In: 12TH WORKSHOP ON CHARM++ AND ITS APPLICATIONS, University of Illinois at Urbana Champaign, US. **Proceedings...** [S.l.]: UIUC, 2014. p. 1–3.

PALLIPADI, V.; LI, S.; BELAY, A. cpuidle: Do nothing, efficiently. In: LINUX SYMPOSIUM, Ottawa, Canada. **Proceedings...** [S.l.]: Citeseer, 2007. p. 1–10.

PANETTA, J. et al. Accelerating Kirchhoff Migration by CPU and GPU Cooperation. In: INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE AND HIGH PERFORMANCE COMPUTING (SBAC-PAD), Sao Paulo, BR. **Proceedings...** IEEE, 2009. p. 26–32. ISBN 978-0-7695-3857-0. Available from Internet: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5336217>.

PARK, S. et al. Accurate modeling of the delay and energy overhead of dynamic voltage and frequency scaling in modern microprocessors. **Computer-Aided Design of Integrated Circuits and Systems**, IEEE Computer Society, v. 32, n. 5, p. 695–708, 2013.

PEARCE, O. et al. Quantifying the effectiveness of load balance algorithms. In: INTERNATIONAL CONFERENCE ON SUPERCOMPUTING (ICS), San Servolo Island, Venice, Italy. **Proceedings...** ACM, 2012. p. 185–194. ISBN 978-1-4503-1316-2. Available from Internet: <http://doi.acm.org/10.1145/2304576.2304601>.

PELLEGRINI, F.; ROMAN, J. Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In: HIGH-PERFORMANCE COMPUTING AND NETWORKING (HPCN), London, UK. **Proceedings...** Springer-Verlag, 1996. p. 493–498. ISBN 3-540-61142-8. Available from Internet: <http://dl.acm.org/citation.cfm?id=645560.658570>.

PERAZA, J. et al. PMaC's green queue: a framework for selecting energy optimal DVFS configurations in large scale MPI applications. **Concurrency and Computation: Practice and Experience**, John Wiley & Sons, Ltd, p. 1–20, 2013. ISSN 1532-0634. Available from Internet: <http://dx.doi.org/10.1002/cpe.3184>.

PHILLIPS, J. C. et al. NAMD: Biomolecular simulation on thousands of processors. In: CONFERENCE SUPERCOMPUTING (SC), Baltimore, Maryland. **Proceedings...** IEEE Computer Society Press, 2002. p. 1–18. Available from Internet: <http://dl.acm.org/citation.cfm?id=762761.762810>.

PILLA, L. L. **Topology-Aware Load Balancing for Performance Portability over Parallel High Performance Systems**. 125 p. Thesis (PhD) — Programa de Pós-Graduação em Computação da UFRGS, Porto Alegre, BR, 07 2014.

PILLA, L. L. et al. Comprehensivebench: a benchmark for the extensive evaluation of global scheduling algorithms. **Journal of Physics: Conference Series**, v. 649, n. 1, p. 012007, 2015. Available from Internet: <http://stacks.iop.org/1742-6596/649/i=1/a=012007>.

PILLA, L. L. et al. A hierarchical approach for load balancing on parallel multi-core systems. In: INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING (ICPP), Pittsburgh, PA. **Proceedings...** [S.l.]: IEEE Computer Society, 2012. p. 118–127. ISSN 0190-3918.

PILLA, L. L. et al. A topology-aware load balancing algorithm for clustered hierarchical multi-core machines. **Future Generation Computer Systems**, Elsevier, v. 30, p. 191–201, 2014.

RAGHU, H.; SAURAV, S. K.; BAPU, B. S. Paas: Power aware algorithm for scheduling in high performance computing. In: INTERNATIONAL CONFERENCE ON UTILITY AND CLOUD COMPUTING, Dresden. **Proceedings...** [S.l.]: IEEE Computer Society, 2013. p. 327–332.

RAJOVIC, N. et al. Tibidabo: Making the case for an arm-based hpc system. **Future Generation Computer Systems**, Elsevier, v. 36, p. 322–334, 2014.

REN, D. Q.; SUDA, R. Investigation on the power efficiency of multi-core and gpu processing element in large scale simd computation with cuda. In: INTERNATIONAL CONFERENCE ON GREEN COMPUTING, Chicago, IL. **Proceedings...** [S.l.]: IEEE, 2010. p. 309–316.

RODRIGUES, E. et al. Optimizing an mpi weather forecasting model via processor virtualization. In: INTERNATIONAL CONFERENCE ON HIGH PERFORMANCE COMPUTING (HIPC), Goa, India. **Proceedings...** [S.l.], 2010. p. 1–10.

RODRIGUES, E. R. et al. Multi-core aware process mapping and its impact on communication overhead of parallel applications. In: SYMPOSIUM ON COMPUTERS AND COMMUNICATIONS (ISCC), Sousse Tunisia. **Proceedings...** [S.l.]: IEEE Computer Society, 2009. p. 811–817.

RODRIGUES, E. R. et al. A comparative analysis of load balancing algorithms applied to a weather forecast model. In: INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE AND HIGH PERFORMANCE COMPUTING (SBAC-PAD), Petropolis. **Proceedings...** [S.l.]: IEEE Computer Society, 2010. p. 71–78.

ROTEM, E. et al. Power-management architecture of the intel microarchitecture code-named sandy bridge. **Micro**, IEEE Computer Society, v. 32, n. 2, p. 20–27, 2012. ISSN 0272-1732.

ROUNTREE, B. et al. Beyond DVFS: A first look at performance under a hardware-enforced power bound. In: INTERNATIONAL PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM WORKSHOPS PHD FORUM (IPDPSW), Shanghai. **Proceedings...** [S.l.]: IEEE Computer Society, 2012. p. 947–953.

SAROOD, O.; MENESES, E.; KALÉ, L. V. A 'cool' way of improving the reliability of HPC machines. In: INTERNATIONAL CONFERENCE ON HIGH PERFORMANCE COMPUTING, NETWORKING, STORAGE AND ANALYSIS (SC), Denver, Colorado. **Proceedings...** [S.l.]: ACM, 2013. p. 58:1–58:12. ISBN 978-1-4503-2378-9.

SAROOD, O. et al. Cool, load balancing for high performance computing data centers. **Transactions on Computers**, IEEE Computer Society, v. 61, n. 12, p. 1752–1764, 2012. ISSN 0018-9340.

SCOGLAND, T.; SUBRAMANIAM, B.; FENG, W. The Green500 List: Escapades to Exascale. **Springer Computer Science-Research and Development**, v. 28, n. 2, p. 109–117, 2012. ISSN 1865-2042. Available from Internet: <http://dx.doi.org/10.1007/s00450-012-0212-6>.

SCOTCH. **Static Mapping, Graph, Mesh and Hypergraph Partitioning, and Parallel and Sequential Sparse Matrix Ordering Package, 2007**. 2015. Available from Internet: <http://www.labri.fr/perso/pelegrin/scotch/>.

SEMERARO, G. et al. Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling. In: INTERNATIONAL SYMPOSIUM ON HIGH-PERFORMANCE COMPUTER ARCHITECTURE, Cambridge. **Proceedings...** [S.l.]: IEEE Computer Society, 2002. p. 29–40.

SHIERS, J. The worldwide LHC computing grid. **Computer physics communications**, Elsevier, v. 177, n. 1-2, p. 219–223, 2007.

SPILIOPOULOS, V. et al. Introducing DVFS-management in a full-system simulator. In: INTERNATIONAL SYMPOSIUM ON MODELLING, ANALYSIS & SIMULATION OF COMPUTER AND TELECOMMUNICATION SYSTEMS (MASCOTS), San Francisco, CA. **Proceedings...** [S.l.]: IEEE Computer Society, 2013. p. 535–545.

SPRINGER, R. et al. Minimizing execution time in MPI programs on an energy-constrained, power-scalable cluster. In: SYMPOSIUM ON PRINCIPLES AND PRACTICE OF PARALLEL PROGRAMMING (SIGPLAN), New York, New York, USA. **Proceedings...** ACM, 2006. p. 230–238. ISBN 1-59593-189-9. Available from Internet: <http://doi.acm.org/10.1145/1122971.1123006>.

STEIGERWALD, B. et al. **Energy Aware Computing: Powerful Approaches for Green System Design**. [S.l.]: Intel Press, 2011. 260 p. ISBN 978-1934053416.

SUBRAMANIAM, B.; FENG, W.-c. Understanding Power Measurement Implications in the Green500 List. In: CONFERENCE ON GREEN COMPUTING AND COMMUNICATIONS (GREENCOM), Washington, DC, USA. **Proceedings...** [S.l.]: IEEE Computer Society, 2010. p. 245–251. ISBN 978-0-7695-4331-4.

SUBRAMANIAM, B.; FENG, W.-c. Towards energy-proportional computing for enterprise-class server workloads. In: INTERNATIONAL CONFERENCE ON PERFORMANCE ENGINEERING, Prague, Czech Republic. **Proceedings...** ACM, 2013. p. 15–26. ISBN 978-1-4503-1636-1. Available from Internet: <http://doi.acm.org/10.1145/2479871.2479878>.

TALLENT, N. R.; ADHIANTO, L.; MELLOR-CRUMMEY, J. M. Scalable identification of load imbalance in parallel executions using call path profiles. In: INTERNATIONAL CONFERENCE FOR HIGH PERFORMANCE COMPUTING, NETWORKING, STORAGE AND ANALYSIS, New Orleans, LA. **Proceedings...** [S.l.]: IEEE Computer Society, 2010. p. 1–11.

TAN, L. et al. A2e: Adaptively aggressive energy efficient dvfs scheduling for data intensive applications. In: INTERNATIONAL PERFORMANCE COMPUTING AND COMMUNICATIONS CONFERENCE (IPCCC), San Diego, CA. **Proceedings...** [S.l.], 2013. p. 1–10.

TCHIBOUKDJIAN, M. et al. A work stealing scheduler for parallel loops on shared cache multicores. In: **Euro-Par 2010 Parallel Processing Workshops**. Springer, 2011. p. 99–107. ISBN 978-3-642-21877-4. Available from Internet: <http://dx.doi.org/10.1007/978-3-642-21878-1_13>.

TESSER, R. K. et al. Improving the performance of seismic wave simulations with dynamic load balancing. In: EUROMICRO INTERNATIONAL CONFERENCE ON PARALLEL, DISTRIBUTED AND NETWORK-BASED PROCESSING (PDP), Torino, IT. **Proceedings...** [S.l.]: IEEE Computer Society, 2014. p. 196–203. ISSN 1066-6192.

TESSER, R. K. et al. Using AMPI to improve the performance of the Ondes3D seismic wave simulator. In: JOINT LABORATORY ON PETASCALE COMPUTING (JLPC), Illinois, USA. **Proceedings...** [S.l.]: INRIA, 2014. p. 118–127.

TORRELLAS, J. Architectures for extreme-scale computing. **Computer**, IEEE, v. 42, n. 11, p. 28–35, nov. 2009. ISSN 0018-9162. Available from Internet: <http://dx.doi.org/10.1109/MC.2009.341>.

WALKO, R. L. et al. Coupled atmosphere-biophysics-hydrology models for environmental modeling. **Journal of applied meteorology**, v. 39, n. 6, p. 931–944, 2000.

WATTS, J.; TAYLOR, S. A practical approach to dynamic load balancing. **Parallel and Distributed Systems, IEEE Transactions on**, IEEE, v. 9, n. 3, p. 235–248, 1998.

WEAVER, V. et al. Measuring energy and power with PAPI. In: INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING WORKSHOPS (ICPPW), Pittsburgh, PA. **Proceedings...** [S.l.], 2012. p. 262–268. ISSN 1530-2016.

WEHNER, M.; OLIKER, L.; SHALF, J. A Real Cloud Computer. **IEEE Spectrum**, v. 46, n. 10, p. 24–29, 2009. ISSN 0018-9235.

YOUNGE, A. et al. Efficient resource management for cloud computing environments. In: INTERNATIONAL GREEN COMPUTING CONFERENCE (IGCC), Chicago, IL. **Proceedings...** [S.l.]: IEEE Computer Society, 2010. p. 357–364.

ZHENG, G. **Achieving high performance on extremely large parallel machines performance prediction and load balancing**. 277 p. Thesis (PhD) — Graduate College of the University of Illinois, 08 2005.

ZHENG, G. et al. Periodic hierarchical load balancing for large supercomputers. **International Journal of High Performance Computing Applications**, SAGE Publications, v. 25, n. 4, p. 371–385, 2011.

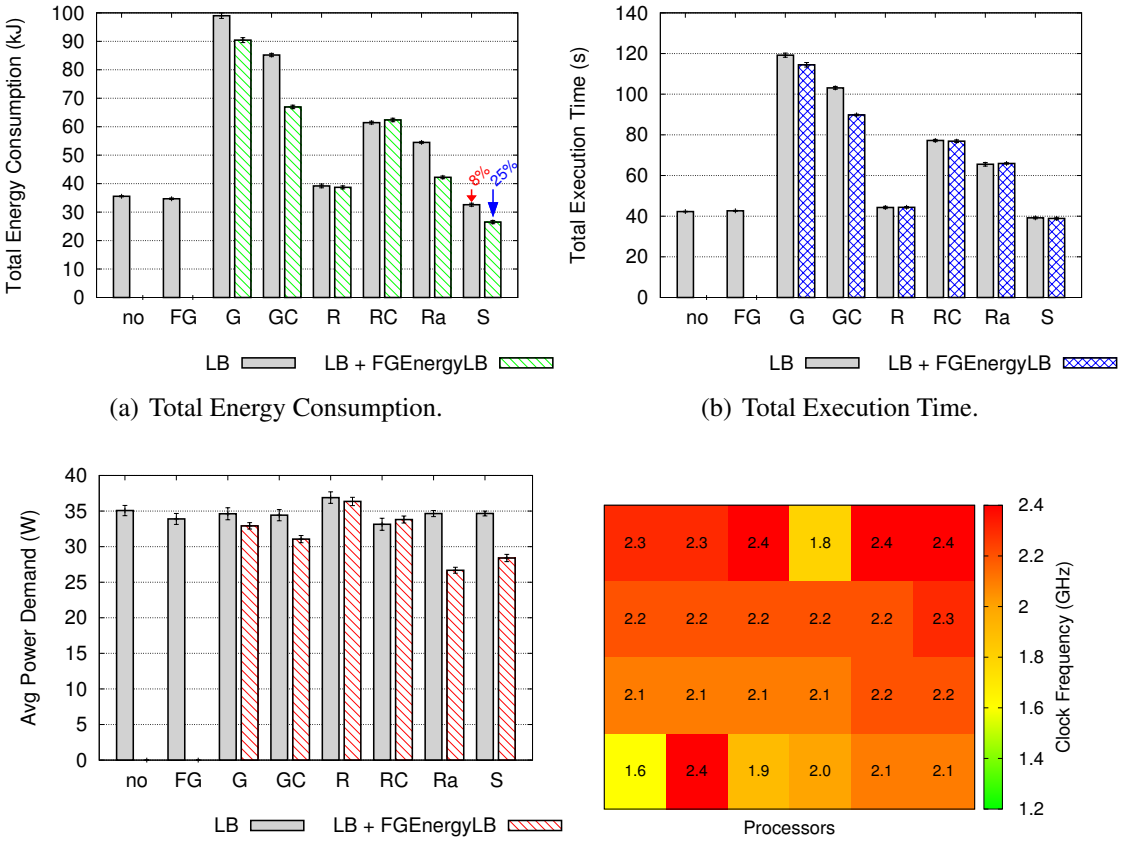
ZHENG, G. et al. Hierarchical load balancing for charm++ applications on large supercomputers. In: INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING WORKSHOPS (ICPPW), San Diego, CA. **Proceedings...** [S.l.]: IEEE, 2010. p. 436–444. ISSN 1530-2016.

APPENDIX A — FG-ENERGYLB EVALUATION

A.1 Evaluation on Benchmarks

- Benchmark: *kNeighbor*

Figure A.1: Evaluation of FG-ENERGYLB with *kNeighbor* benchmark.



(a) Total Energy Consumption.

(b) Total Execution Time.

(c) Average Power Demand.

(d) Final Clock Frequency to FG-ENERGYLB alone.

Source: The author

i) FG-ENERGYLB Evaluation

Differently from *lb_test*, *kNeighbor* is a communication benchmark and has a balanced state at start. The energy consumption of *kNeighbor* was reduced by only 2.4% (from 35.56 kJ to 34.69 kJ) when FG-ENERGYLB was applied alone. This energy saving occurs due the reduction of the average power demand of the cores from 35.1 W to 33.9 W (3.4%). This way, adjusting the clock frequency during the execution, it was able to reduce the clock to 1.898 GHz on average.

Thus, when the execution finishes using FG-ENERGYLB alone, only four cores were kept on the maximum clock frequency, and the lowest clock frequency registered

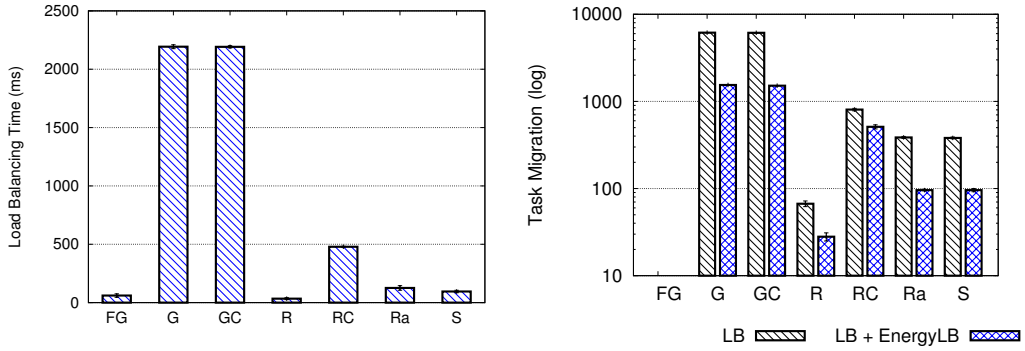
was 1.6 GHz, as show the Figure A.1(d).

ii) LB + FG-ENERGYLB Evaluation

When used in conjunction with the other load balancers, FG-ENERGYLB reduced the total energy consumption of *kNeighbor* by up to 9% and 21% with GREEDYLB and GREEDYCOMMLB, and 22.5% and 18.7% with RANDCENTLB and SCOTCHLB, which represents a 10.5% reduction on average in these tests. These centralized load balancers increased the total energy consumption of *kNeighbor* by increasing its total execution time. This happened because the task migrations resulted in worse communication times for the tasks while their load is balanced.

The use the some load balancers increase of the total execution time. For GREEDYLB and GREEDYCOMMLB this increase happens because of the large amount of migrated tasks, since *kNeighbor* was executed with 8 times more tasks than *lb_test*. In test with these load balancers were migrated 1,536 tasks on average, which represent 138.3 MB of data transferred each load balancing call. To perform these amounts of migrations, the load balancers have the higher overhead, which spent 2.2 seconds on average per load balancing call, as show the Figure A.2(a).

Figure A.2: Load balancing overhead and Total task migration to *kNeighbor* load balancer with different load balancers.



(a) Load balancing overhead. (b) Total task migration.

Source: The author

When FG-ENERGYLB was employed together with GREEDYLB, GREEDYCOMMLB the amount total of task migrations was reduced from 6,141 to 1,520 tasks on average, which represents reduction of up to 72%. This reduction in the total task migrated (Figure A.2(b)), results in a reduction of total execution time of 3.9% and 12.9% respectively. Besides of this reduction on runtime, the biggest energy savings is achieve by reduction of the average power demand, as show the Figure A.1(c).

Working over residual imbalance of the GREEDYLB, GREEDYCOMMLB, RANDCENTLB and SCOTCHLB, FG-ENERGYLB is able to reduce the average power demand in 4.9%, 9.7%, 22.9% and 16.7% respectively. These reductions were achieved, once that FG-ENERGYLB detect the residual imbalance and adjust the clock frequency of each core relative to more overload core. Applying FG-ENERGYLB in these load balancers the average clock frequency during runtime is reduced to 1.987 GHz, 1.813 GHz, 1.680 GHz and 1.780 GHz respectively.

The total execution time also increase to REFINECOMMLB and RANDCENTLB. Using RANDCENTLB load balancer, almost all tasks are migrating every load balancing calls, while that using REFINELB, in the first call 608 tasks were migrated, reducing this value on other load balancing calls. These migrations represent 139 and 32.24 MB of data on average transferred, which influence the load balancing times. In these tests experiments, REFINELB and RANDCENTLB have overhead of 0.47 and 0.12 seconds per load balancing call, respectively.

For *kNeighbor*, the biggest amount of energy saving is achieved when SCOTCHLB is applied alone. This load balancer reduced the total execution time in 7.28% compared to baseline. Such reduction in runtime results in an equivalent amount of energy saving. However, when FG-ENERGYLB is used together with SCOTCHLB the average power demand is also reduced up to 18%, from 34.66 W to 28.4 W. In this mode, the total of energy saving achieve 25.48%.

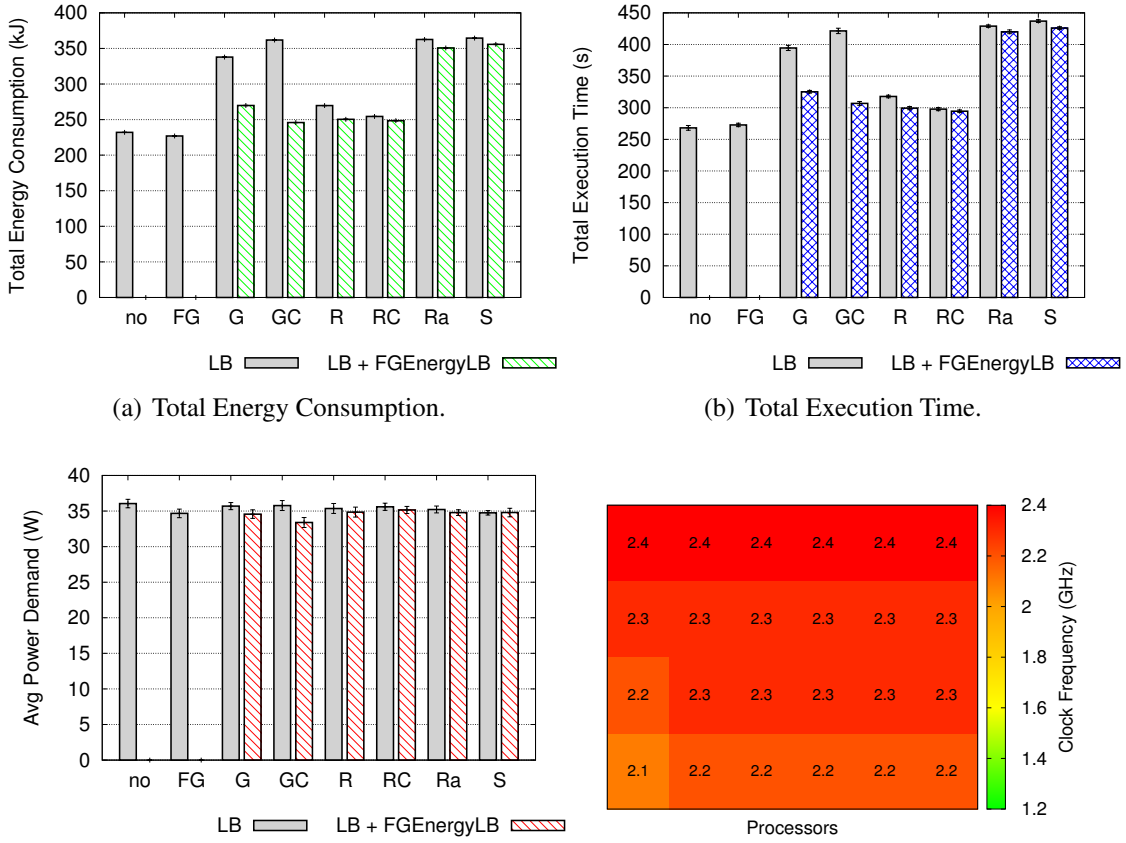
In this test, SCOTCHLB + FG-ENERGYLB, the average clock frequency during the runtime is reduced to 1.780 GHz and the execution finishes having 14 cores in minimum value available, 4 cores with intermediate frequency between 1.4 and 1.6 GHz and only one core with maximum frequency.

- Benchmark: *stencil 4D*

i) FG-ENERGYLB Evaluation

Different to other benchmarks, a small amount energy saving was observed on *stencil 4D*. Although this benchmark presents an initial imbalance, using these input parameters, the original task distribution generates a balanced load in our experimental platform. In this form, when FG-ENERGYLB is used alone, it is able to reduce the clock frequency during the runtime only to 2.248 GHz, which reduce its energy consumption by only 2.1% (from 232.01 kJ to 227.00 kJ).

Figure A.3: Evaluation of FG-ENERGYLB with *stencil 4D* benchmark.



(a) Total Energy Consumption.

(b) Total Execution Time.

(c) Average Power Demand.

(d) Final Clock Frequency to FG-ENERGYLB alone.

Source: The author

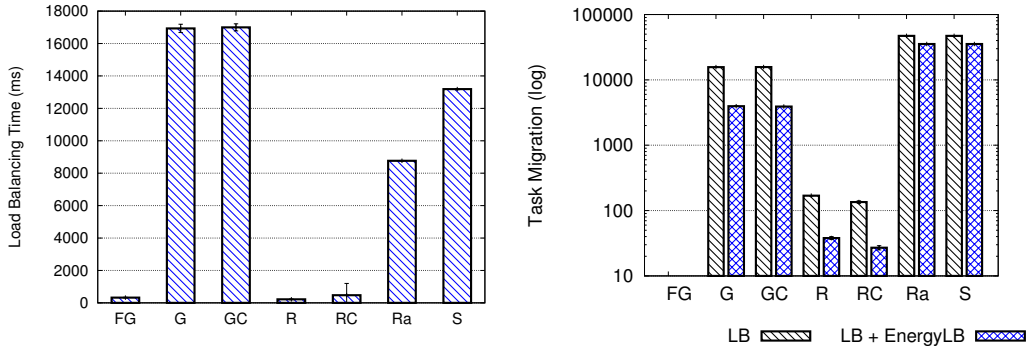
The *stencil 4D* execution finishes with FG-ENERGYLB having 6 cores with the maximum clock frequency, and the lowest clock frequency registered was 2.1 GHz, as shows the Figure A.3(d).

ii) LB + FG-ENERGYLB Evaluation

The tasks have a large memory footprint in *stencil 4D*, their migrations can be costly. So, all the load balancers tested increase the total execution time of this benchmark. GREEDYLB, GREEDYCOMMLB, RANDCENTLB and SCOTCHLB present the increase more significant in the execution time. This increase occurs due to two factors. The first is the high load balancing times of these load balancers. They spend 16.9, 16.9, 8.7 and 13.1 seconds on average per load balancing call, while that FG-ENERGYLB, REFINELB and REFINECOMMLB spend only 0.3, 0.2 and 0.4 seconds, respectively (Figure A.4(b)). The second is the large amount of migrated tasks. These algorithms migrate 3,916 tasks on average every call from a total of 4096 tasks. These migrations resulting in 939.84 MB of data transferred.

On the other hand, FG-ENERGYLB applied alone no migrates any task, while that REFINELB and REFINECOMMLB migrate only 42 and 33 respectively. This large difference in total task migration and the load balancing time is shown in the Figure A.4.

Figure A.4: Load balancing overhead and Total task migration to *stencil 4D* load balancer with different load balancers.



(a) Load balancing overhead. (b) Total task migration.

Source: The author

Using FG-ENERGYLB together the selected load balancers the total energy consumption is reduced. This happens mainly due the reduction of total execution time, once FG-ENERGYLB perform DVFS instead of migrate task.

In this test, the energy consumption is reduced using FG-ENERGYLB over the residual imbalance GREEDYLB, GREEDYCOMMLB and REFINELB by 20%, 32% and 7%, respectively. These reductions are also due to a change in the power demand of the cores of 3.1% on average, and to a reduction in the total execution time of 13% in relation to executions with this load balancer alone.

Different to other load balancers, FG-ENERGYLB also reduces the amount of task migrations of REFINELB and REFINECOMMLB on *stencil 4D*.

The greatest amount of energy saving is achieved using FG-ENERGYLB alone. It has an overhead of only 1.73% and reduces the average power in 3.82% on average if compared to baseline. So, the total energy consumption is reduced in up to 2.16%.

A.1.1 Percentage of Energy Spent on Load Balancing

Although each load balancer algorithm uses different approach to make its decisions and each benchmark has different load imbalance, different runtime, power and energy improvements were achieved when these strategies were applied. However, load balancing incur in overhead over application runtime, which is also responsible for a per-

centage of the total energy spent.

To better analyze the total energy consumption with load balancing, for each benchmark, we select the measured time overhead of each load balancers, which are depicted in Table A.1. These times are composed of the time spent in data collection, strategy make decisions and the task migration. These load balancing costs generally are relative to techniques employed by each algorithm. Are also related to system configuration and application factors, once it need information of both to make decisions and migrate tasks every load balancing call.

Table A.1: Average load balancing duration in milliseconds for each benchmark.

Load Balancer	*- - lb_test - -*		*- - kNeighbor - -*		*- - stencil 4D - -*		*- ComprehensiveBench -*	
	LB	**LB+FG**	**LB**	**LB+FG**	**LB**	**LB+FG**	**LB**	**LB+FG**
FG-ENERGYLB	5.52	-	60.63	-	325.45	-	5.19	-
GREEDYLB	5.74	5.41	2192.86	566.29	16934.57	7761.05	4.70	10.64
GREEDYCOMMLB	5.56	5.71	2191.99	569.89	16998.38	7824.63	4.61	11.54
REFINELB	4.32	5.43	34.08	36.62	217.46	317.99	3.50	8.89
REFINECOMMLB	4.88	4.91	478.72	376.44	474.61	372.17	3.56	8.60
RANDCENTLB	12.12	7.40	125.80	117.89	8767.19	2427.36	7.53	6.88
SCOTCHLB	9.23	6.90	95.41	87.43	13187.10	6234.52	6.21	5.18

Source: The author

For FG-ENERGYLB the energy improvements were even greater. This happen once it employs a strategy that benefits from other the load balancing framework when imbalance is large or uses DVFS to adjust the clock frequency of the processors according with their relative load.

FG-ENERGYLB has overhead similar to CHARM++ load balancers when applied on *lb_test* and *ComprehensiveBench*. However, it is 36 and 52 times lower than GREEDYLB and GREEDYCOMMLB on *kNeighbor* and *stencil 4D*, as shown in the Table A.1. Using FG-ENERGYLB over *kNeighbor* and *stencil 4D* the average load balancing time is reduced from 739.9 and 8129.2 ms to 292.43 and 4156.29 ms, respectively. This way, for these benchmarks FG-ENERGYLB is able to saving energy by reduce the total execution time obtained with the reducing the number of migrations, however the greater gains come from reductions on average power demand.

Load balancers can increase the application runtime when the time spent to make decisions and task migration is greater than the gains achieved with the new load distribution. Load balancers as GREEDYLB, GREEDYCOMMLB, and RANDCENTLB do not take into account the current task mapping, which result in several task migrations at each load balancing call. Their load balancing times were responsible by the increase of total execution time, which also increase of the total energy consumption, as observed in tests

Table A.2: Percentage of energy spend with load balancing over the total energy consumption for each benchmark.

Load Balancer	*- - lb_test - -*		*- - kNeighbor - -*		*- - stencil 4D - -*		*- ComprehensiveBench -*	
	LB %	**LB+FG %**	**LB %**	**LB+FG %**	**LB %**	**LB+FG %**	**LB %**	**LB+FG %**
FG-ENERGYLB	0.16	-	0.57	-	0.48	-	0.01	-
GREEDYLB	0.29	0.29	7.36	1.98	17.17	9.55	0.01	0.03
GREEDYCOMMLB	0.28	0.30	8.50	2.54	16.13	10.20	0.01	0.03
REFINELB	0.23	0.28	0.31	0.33	0.27	0.42	0.01	0.03
REFINECOMMLB	0.26	0.26	2.48	1.96	0.64	0.51	0.01	0.03
RANDCENTLB	0.34	0.20	0.77	0.72	8.17	2.31	0.01	0.01
SCOTCHLB	0.45	0.36	0.97	0.90	12.07	5.85	0.02	0.01

Source: The author

with *kNeighbor* and *stencil 4D*.

On the other hand, REFINELB and REFINECOMMLB were able to reduce or keep the execution time of the four selected benchmarks, once that these load balancers use the current task mapping, this way having few task migrations.

Other factor that also has influence on these load balancing costs is the amount of memory used during the runtime each application. *lb_test*, *kNeighbor*, *stencil 4D* and *ComprehensiveBench* using 2.62 MB, 36.30 MB, 1.61 GB and 2.70 MB of memory on average respectively.

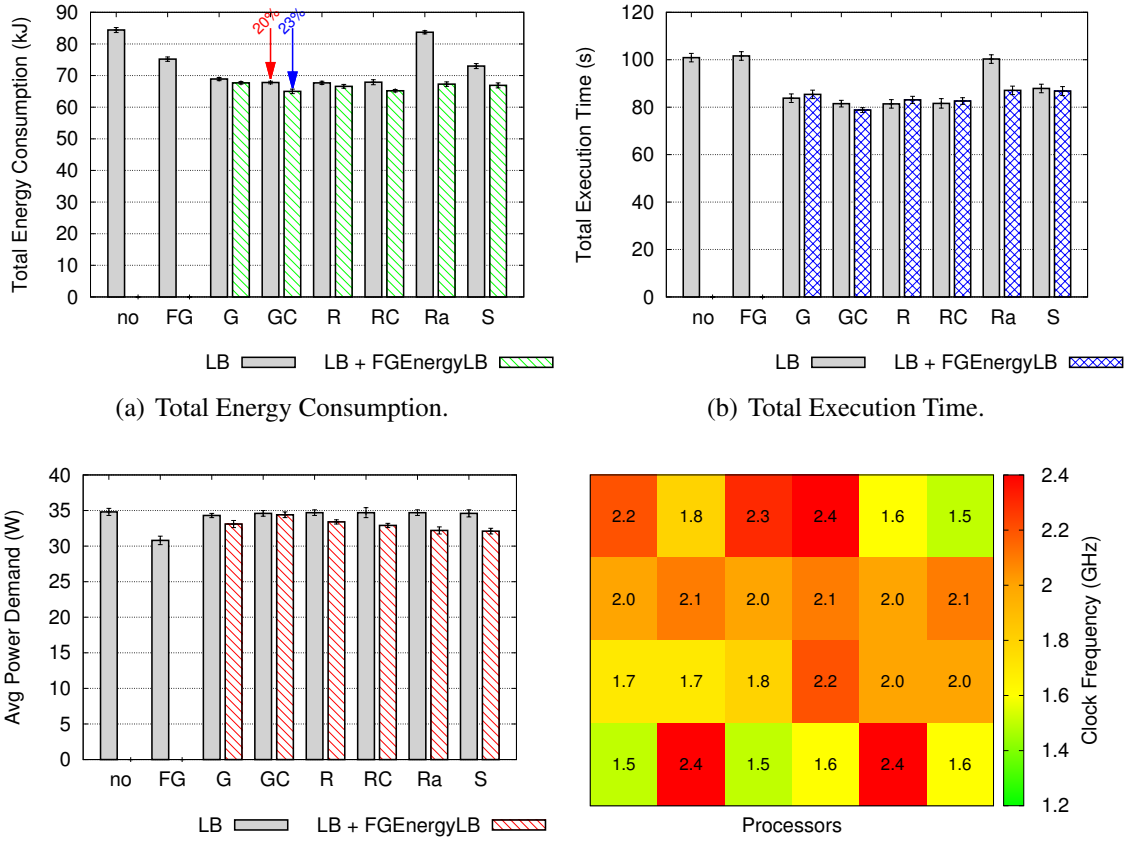
However, the load balancer generates an overhead and when this cost exceeds its benefits, the total execution time and energy consumption are increased. In this context, we relate the total energy spent with load balancing to each test performed with the total energy consumption on execution, as depicted in the Table A.2.

The energy spent with load balancing is 0.29% and 0.01%, on average, of total energy consumption for *lb_test* and *ComprehensiveBench*. Applying FG-ENERGYLB over these benchmarks, the improvement percentage is very similar. However, in tests with *kNeighbor* and *stencil 4D*, FG-ENERGYLB is able to reduce the percentage load balancing energy from 2.99 and 7.85% to 1.4% and 4.81%, respectively. These reductions come from reduce load balancing overhead and mainly the reductions on average power demand. We evaluate how FG-ENERGYLB behaves in real applications in the next section.

A.2 Evaluation on Real Applications

- Application: *Lassen*

Figure A.5: Evaluation of FG-EnergyLB with *Lassen* application.



(a) Total Energy Consumption.

(b) Total Execution Time.

(c) Average Power Demand.

(d) Final Clock Frequency to FG-EnergyLB alone.

Source: The author

i) FG-EnergyLB Evaluation

Among the real applications analyzed, *Lassen* is the one with the highest irregularity besides its significant load dynamicity as shows the Figure A.6(a). Taking into account the original mapping of its 256 tasks, the imbalance is up to 30 times at a given instant of the execution (Figure A.6), but in general the imbalance is up 5 times.

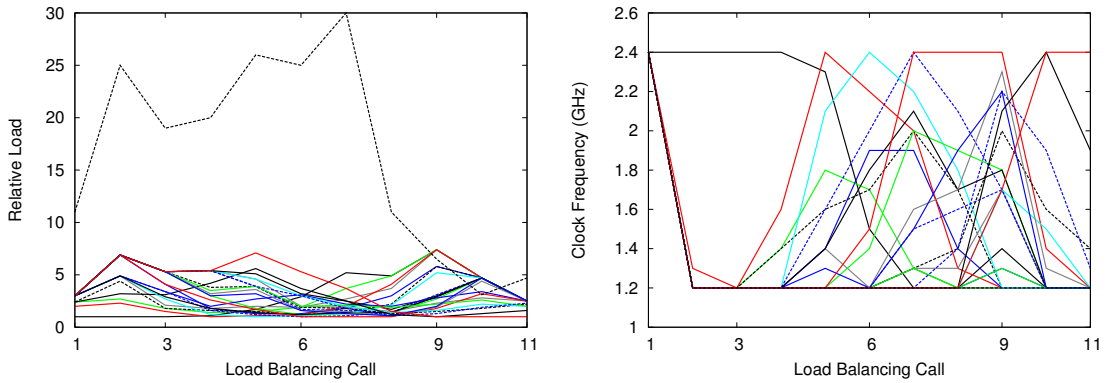
The total execution times obtained on the tests with *Lassen* are depicted in Figure A.5(b). The use of FG-EnergyLB alone increases the total execution time from 100.9 to 101.6 seconds, which represents an overhead of 0.64%.

Considering the load imbalance of this application and its dynamicity, FG-EnergyLB make several DVFS to adjust the clock frequency of the cores according their relative load, as shown in Figure A.6(b). In this test, the average clock frequency is 1.490 GHz, which

incurs in a reduction of power demand from 34.8 W to 30.8 W (11.48%).

In this test, when FG-EnergyLB was applied alone, the execution finishes with 9 cores having clock lower than 2.0 GHz and only 3 cores remains with its maximum frequency. So, given these clock reductions during runtime, FG-EnergyLB achieves an energy saving of up to 10.9%, reducing the total energy consumption from 84.4 to 75.2 kJ.

Figure A.6: Comparison of Instantaneous Relative Load with Instantaneous Clock Frequency for *Lassen* when used FG-EnergyLB alone.



(a) Instantaneous Relative Load. (b) Instantaneous Clock Frequency.

Source: The author

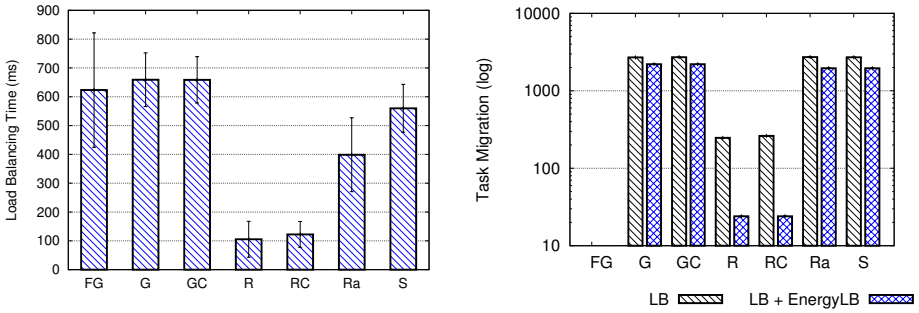
ii) LB + FG-EnergyLB Evaluation

Different from other applications, over *Lassen* all load balancers improve the runtime over the baseline. Except RandCentLB, the other load balancers have speedup of 1.18 on average. This improvements were achieved even GreedyLB, GreedyCommLB, RandCentLB and ScotchLB having a highest load balancing times, which represent 569 ms on average, per load balancing call. However, this time overhead is offset by the reduction in the total execution time achieved with the use of the load balancing. These load balancers migrated 245 tasks on average per call, which represent 4.9 MB of data transferred. On the other hand, RefineLB and RefineCommLB migrate only 26 tasks. Figure A.7 summarizes the load balancing costs and total task migration of *Lassen*.

Applying FG-EnergyLB in conjunction with other load balancers attains different energy saving levels. Performing DVFS according to load imbalance in this application FG-EnergyLB reduces the average power in up to 5.23%, which result in a reduction in energy consumption of 6.55%.

The lower energy consumption has been achieved with GreedyLB, GreedyCommLB, RefineLB, RefineCommLB. Using these load balancers alone 19.8% of total energy was saving. In addition to the clock adjustment, using the FG-EnergyLB has a total energy consumption reduction of up to 22.91%.

Figure A.7: Load balancing overhead and Total task migration to *Lassen* with different load balancers.



(a) Load balancing overhead. (b) Total task migration.

Source: The author

iii) Percentage of Energy Spent on Load Balancing

Table A.3 depicts the comparison of load balancing times incurring on FG-ENERGYLB and other load balancer strategies.

Table A.3: Average load balancing duration in seconds for each real application.

Load Balancer	*- - Ondes3D - -*		*- - Lulesh - -*		*- - Lassen - -*	
	LB	**LB+FG**	**LB**	**LB+FG**	**LB**	**LB+FG**
FG-ENERGYLB	0.30	-	0.25	-	0.32	-
GREEDYLB	4.01	0.57	0.35	0.29	0.66	0.61
GREEDYCOMMLB	4.16	0.65	0.40	0.26	0.66	0.60
REFINELB	1.43	0.88	0.16	0.23	0.11	0.55
REFINECOMMLB	1.43	0.64	0.24	0.24	0.12	0.48
RANDCENTLB	6.70	0.62	0.30	0.20	0.40	0.49
SCOTCHLB	3.01	0.58	0.38	0.23	0.56	0.55

Source: The author

The total energy consumption on execution, as depicted in the Table A.4.

The greater reduction is achieved for *Ondes3D*. The energy spent with load balancing was reduced from 7.87% to 2.07%, on average. On the other hand, for *Lassen* application, FG-ENERGYLB alone spent 2.9% of energy with load balancing and it applied together with REFINELB and REFINECOMMLB increases the overhead. Thus, the average load balancing energy was increased from 4.19% to 5.93%.

Figure A.8 depicts the energy spend with load balancing (LB Energy) over total energy consumption for *Lassen* application (App Energy).

iv) Threshold Evaluation

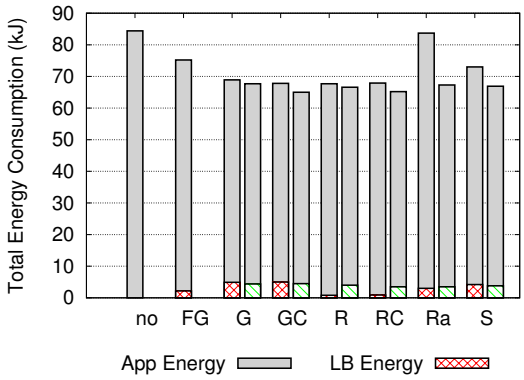
For *Lassen*, the experiments were performed using 256 tasks mapped in 24 cores, being each task with 550 iteration. The execution without load balancer (NOLB) spends

Table A.4: Percentage of energy spend with load balancing over the total energy consumption for each real application.

Load Balancer	*- - Ondes3D - -*		*- - Lulesh - -*		*- - Lassen - -*	
	LB %	**LB+FG %**	**LB %**	**LB+FG %**	**LB %**	**LB+FG %**
FG-ENERGYLB	0.98	-	4.12	-	2.90	-
GREEDYLB	11.45	1.61	5.83	4.88	7.16	6.49
GREEDYCOMMLB	11.75	1.73	6.64	4.38	7.35	6.91
REFINELB	5.38	3.59	3.10	4.39	1.18	6.03
REFINECOMMLB	5.28	2.33	4.46	4.41	1.36	5.32
RANDCENTLB	11.52	1.39	3.52	2.43	3.62	5.13
SCOTCHLB	8.75	1.74	6.74	4.13	5.80	5.72

Source: The author

Figure A.8: Energy spend with load balancing over total energy consumption for each real application.



(a) *Lassen*.

Source: The author

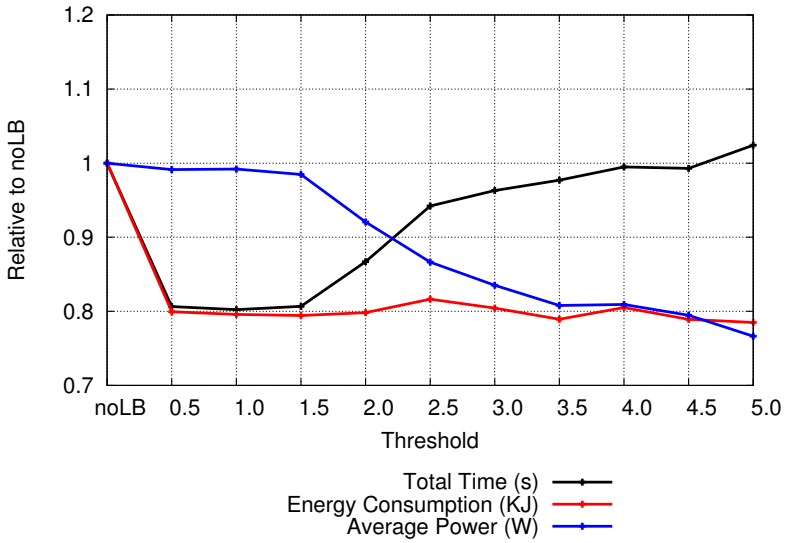
84.4 kJoules and takes 100.9 seconds, which represent an average power demand of 34.9 W. Similar to other applications, these values are taken as reference (NOLB) in the Figure A.9.

Application is executed with load balancer call every 50 iteration, so, it is performed 10 times. Using threshold of 0.5 and 1.0, FG-ENERGYLB not adjust the clock frequency any time. It always call GREEDYLB to migrate tasks. In these tests, the average power remain close to 35 W. A reduction in the total energy consumption is achieved once runtime is reduced due a better distribution of tasks.

Increasing the threshold to 1.5, FG-ENERGYLB adjust the clock only 1 time, and in other 9 times, it performs task migrations. In this case, the average power is reduced in 1.52% (34.3 W), maintaining constant the runtime.

To threshold equal to 2.0, FG-ENERGYLB performs 2 times DVFS, which is able to reduce the power demand to 32.1 W, a reduction of 7.93% compared to NOLB. How-

Figure A.9: FG-ENERGYLB comparison with different threshold value on *Lassen*.



Source: The author

ever, the runtime increases 7.47% compared to threshold equal to 1.5. In this way, the total energy consumption increase in 0.4%.

In the tests with threshold value greater than 2.0, a greater amount of DVFS is performed, resulting in lower average power demand. However, such reductions cause an equivalent increase in the total execution time, thus maintaining the energy consumption constant.

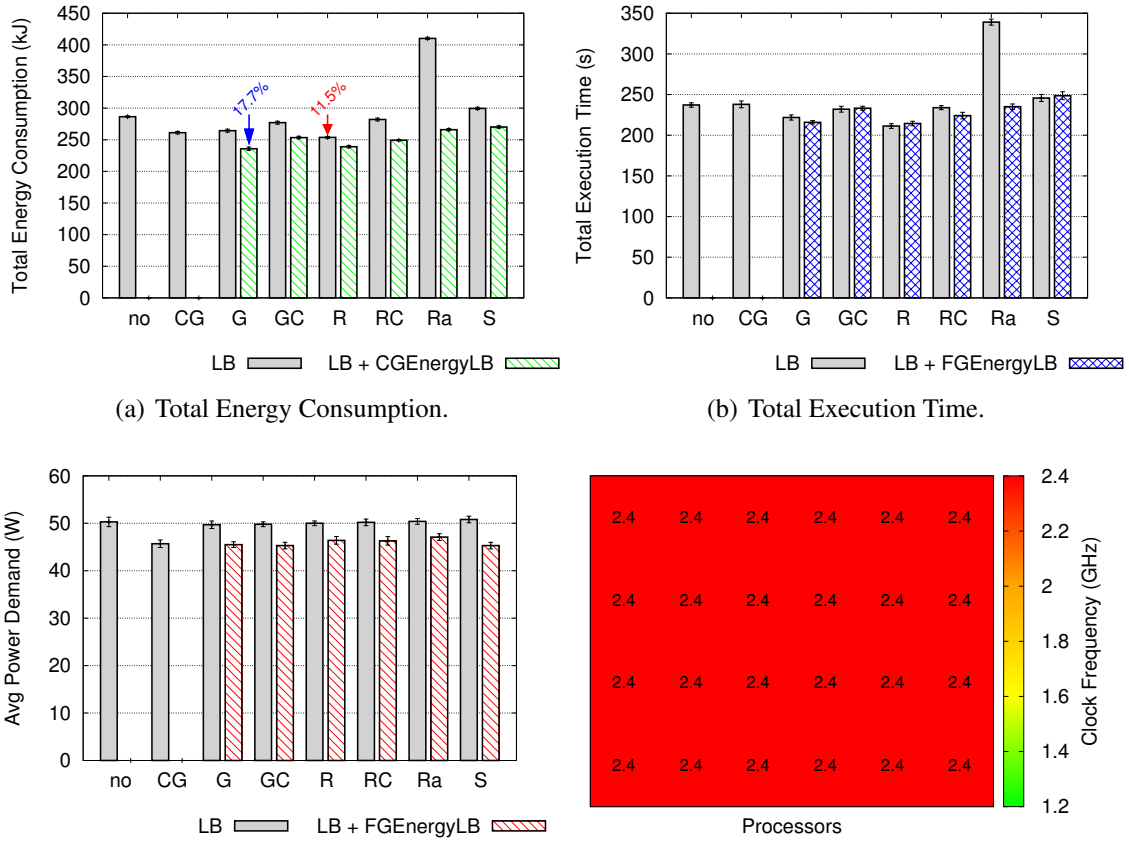
In the execution com threshold equal to 5.0, FG-ENERGYLB adjusts the frequency 8 times, which result in a reduction of the power in 23.36% and reduction of energy in 21.50%. However, the runtime exceeds the baseline in 2.42%.

APPENDIX B — CG-ENERGYLB EVALUATION

B.1 Evaluation on Real Applications

- Application: *Lassen*

Figure B.1: Evaluation of CG-ENERGYLB with *Lassen* application.



(a) Total Energy Consumption.

(b) Total Execution Time.

(c) Average Power Demand.

(d) Final Clock Frequency to CG-ENERGYLB alone.

Source: The author

i) CG-ENERGYLB Evaluation

As discussed in the evaluation of FG-ENERGYLB in the Subsection 5.2.2, *Lassen* has a great irregularity and a significant load dynamics, comparing to other real applications. Using CG-ENERGYLB alone over *Lassen* application, it increases the total execution time from 237.2 to 238.0 seconds, an overhead of 0.36%. The CG-ENERGYLB load balancing time for this application is 374 ms on average, per load balancing call.

The imbalance and the dynamicity of its 512 tasks over 192 cores of this application leave some cores underloaded. This way, several DVFS calls are realized to adjust the clock frequency of cores according the relative load during its execution. Performing this,

CG-ENERGYLB is able to reduce of power demand from 50.34 W to 45.72 W during the runtime, which represent a reduction of 9.17% on average. This is result of reduction of the clock frequency to 2.2 GHz.

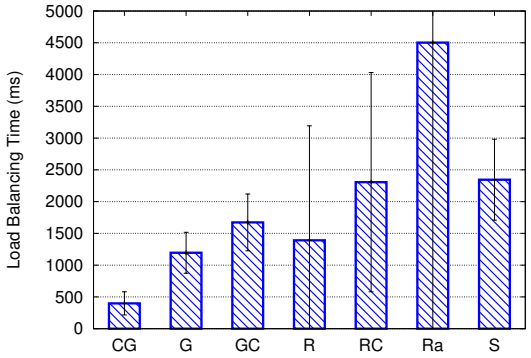
Due to these clock reductions during the runtime, CG-ENERGYLB achieves an energy saving of 8.85%, reduction the total energy spend from 286.5 to 261.2 kJ. However, the load imbalance reduces at end in this execution when CG-ENERGYLB was applied alone and all cores finish their task using the maximum clock frequency available on processors, as shown in Figure B.1(d).

ii) LB + CG-ENERGYLB Evaluation

When CG-ENERGYLB was employed over the residual imbalance of other load balancing algorithms, the energy consumption of *Lassen* was reduced between 5.8% (REFINELB) and 35.2% (RANDCENTLB).

GREEDYLB, GREEDYCOMMLB, REFINELB and REFINECOMMLB improve the runtime over 192 cores in up to 5.23%, on average. These improvements represent a speedup of 1.06 over baseline. The aforementioned load balancers have overheads of 1.195, 1.673, 1.391 and 2.306 seconds on average per load balancing call. On the other hand, the load balancers RANDCENTLB and SCOTCHLB increase the total execution time. A cause of this increase is their overhead of load balancing of 4.502 and 2.345 seconds, as shown in Figure B.2.

Figure B.2: Average load balancing overhead to *Lassen* application.



Source: The author

Other factor that has influence on overhead is the amount of tasks migrated by these load balancers. GREEDYLB, GREEDYCOMMLB, RANDCENTLB and SCOTCHLB migrate 508 tasks on average every call. These migrations move 15.24 MB of date between the cores. Different from those, REFINELB and REFINECOMMLB migrate less tasks. For these load balancers were migrated only 49 tasks, which represent 0.49 MB of

data.

Similar to *Lulesh*, when CG-ENERGYLB is used together other load balancers, it is able to reduce the total energy consumption in 12.95% on average. These energy saving are achieved due the use of DVFS, which reduces the power demand in 8.83% on average for this application.

In this real application, the lower energy consumption is achieved with REFINELB. The energy saving using these load balancers alone is 11.5%. However, the lower energy consumption is achieved using GREEDYLB and CG-ENERGYLB. In this test, the total energy consumption reduces 17.7% compared to baseline, being the energy reduced from 286.5 to 235.8 kJ.

iii) Percentage of Energy Spent on Load Balancing

Table B.1 presents a comparison of load balancing times incurring on different strategies for load balancing step with our proposed CG-ENERGYLB.

Table B.1: Average load balancing duration in seconds for each real application.

Load Balancer	*- - Ondes3D - -*		*- - Lulesh - -*		*- - Lassen - -*	
	LB	**LB+CG**	**LB**	**LB+CG**	**LB**	**LB+CG**
CG-ENERGYLB	0.31	-	0.38	-	0.37	-
GREEDYLB	4.08	3.85	6.84	4.72	1.20	1.09
GREEDYCOMMLB	4.20	3.95	5.98	4.78	1.67	1.20
REFINELB	2.14	2.83	2.78	2.17	1.39	1.47
REFINECOMMLB	1.56	1.90	6.65	5.98	2.31	2.04
RANDCENTLB	11.72	9.46	7.09	4.26	4.50	1.51
SCOTCHLB	6.94	7.05	9.40	4.05	2.35	1.96

Source: The author

The total energy consumption on execution, as presented in the Table B.2.

Figure B.3 depicts the energy spend with load balancing over total energy consumption for each selected real application.

iv) Threshold Evaluation

For *Lassen*, the experiments were performed using 512 tasks mapped in 192 cores, being each task with 550 iteration. Execution without load balancer (NOLB) spends 286.5 kJoules and takes 237.2 seconds, which represent an average power demand of 50.34 W. Similar to other applications, these values are taken as reference (NOLB) in the Figure B.4.

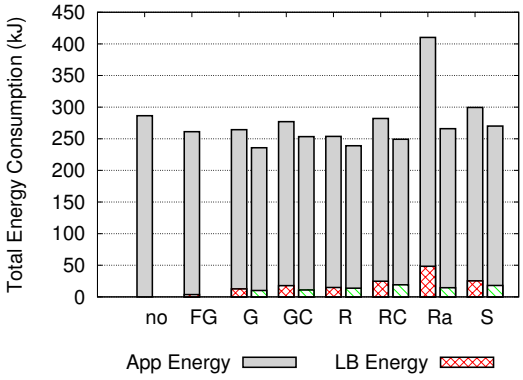
Figure B.5(a) shows instantaneous power measured when the application is exe-

Table B.2: Percentage of energy spend with load balancing over the total energy consumption for each real application.

Load Balancer	*- - Ondes3D - -*		*- - Lulesh - -*		*- - Lassen - -*	
	LB %	**LB+CG %**	**LB %**	**LB+CG %**	**LB %**	**LB+CG %**
CG-ENERGYLB	3.04	-	0.92	-	1.40	-
GREEDYLB	31.47	26.31	13.72	8.50	4.80	4.28
GREEDYCOMMLB	31.51	26.80	12.02	8.49	6.42	4.35
REFINELB	26.27	24.57	7.32	5.11	5.86	5.78
REFINECOMMLB	19.22	16.58	11.75	8.06	8.78	7.69
RANDCENTLB	37.25	22.08	9.99	10.00	11.82	5.44
SCOTCHLB	41.34	30.96	11.66	10.59	8.49	6.69

Source: The author

Figure B.3: Energy spend with load balancing over total energy consumption for each real application.



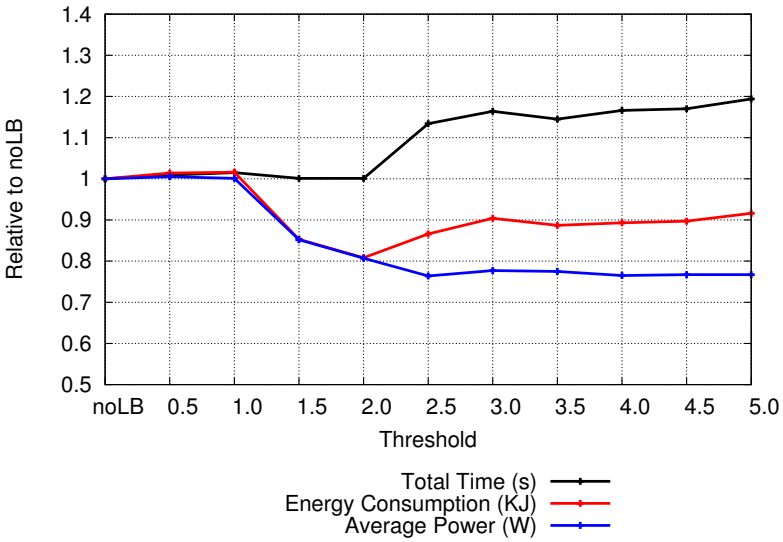
(a) *Lassen*.

Source: The author

cuted with threshold equal to 0.5. For this threshold value, any time is performed DVFS an in every calls (11) tasks are migrated by REFINELB, which result in a high average power demand 50.3 W.

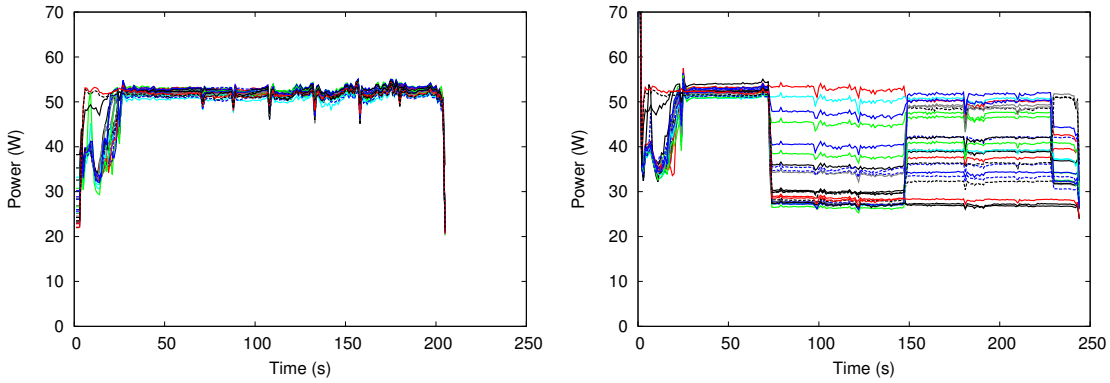
Using threshold equal to 1.5 (Figure B.5(b)), DVFS is performed 3 times. This way, the power demand is reduced in 14.78%, resulting in an average of 42.9 W. However, the greater energy saving is achieve with threshold equal to 5.0 (Figure B.5(c)). In this test 4 times DVFS and 7 times REFINELB is performed, which reduce the power to 38.6 W, a reduction of 23.28%.

Figure B.4: CG-ENERGYLB comparison with different threshold value on *Lassen*.



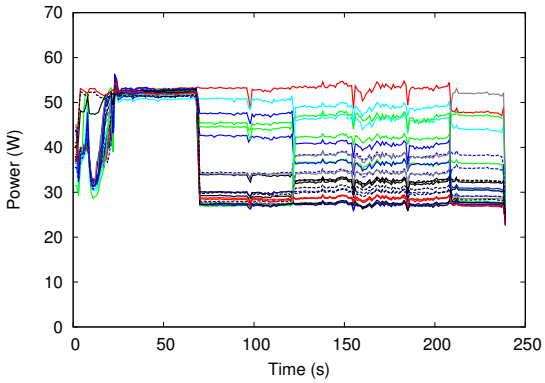
Source: The author

Figure B.5: Power evaluation to different threshold value on *Lassen*



(a) Threshold = 0.5.

(b) Threshold = 1.5.



(c) Threshold = 5.0

Source: The author