Energy consumption optimization of parallel applications with Iterations using CPU frequency scaling

Ahmed Badri Muslim Fanfakh

SPIM

Thèse de Doctorat

école doctorale **sciences pour l'ingénieur et microtechniques**

UFC UNIVERSITÉ DE FRANCHE-COMTÉ

Energy Consumption Optimization of Parallel Applications with iterations using CPU Frequency Scaling

By

Ahmed Badri Muslim FANFAKH

A Dissertation Submitted to the

University of Franche-Comté

in Partial Fulfillment of the Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in Computer Science

Thèse soutenue à Besançon le : 17 octobre 2016

Dissertation Committee:

DR FABIENNE JÉZÉQUEL(HDR)	University of Paris 6	Reviewer
PR JEAN-MARC PIERSON	University of Toulouse 3	Reviewer
PR NABIL ABDENNADHER	University of HES-SO, Switzerland	Examiner
PR RAPHAËL COUTURIER	University of Franche-Comté	Supervisor
DR JEAN-CLAUDE CHARR	University of Franche-Comté	Co-supervisor

ABSTRACT

Energy Consumption Optimization of Parallel Applications with Iterations using CPU Frequency Scaling

Ahmed Badri Muslim FANFAKH
University of Franche-Comté, 2016

Supervisors: Raphaël Couturier and Jean-Claude Charr

In recent years, green computing has become an important topic in the supercomputing research domain. However, the computing platforms are still consuming more and more energy due to the increase in the number of nodes composing them. To minimize the operating costs of these platforms many techniques have been used. Dynamic voltage and frequency scaling (DVFS) is one of them. It can be used to reduce the power consumption of the CPU while computing, by lowering its frequency. However, lowering the frequency of a CPU may increase the execution time of the application running on that processor. Therefore, the frequency that gives the best trade-off between the energy consumption and the performance of an application must be selected. This thesis, presents the algorithms developed to optimize the energy consumption and the performance of synchronous and asynchronous message passing applications with iterations running over clusters or grids. The energy consumption and performance models for each type of parallel application predicts its execution time and energy consumption for any selected frequency according to the characteristics of both the application and the architecture executing this application.

The contribution of this thesis can be divided into three parts: Firstly, optimizing the trade-off between the energy consumption and the performance of the message passing applications with synchronous iterations running over homogeneous clusters. Secondly, adapting the energy and performance models to heterogeneous platforms where each node can have different specifications such as computing power, energy consumption, available frequency gears or network's latency and bandwidth. The frequency scaling algorithm was also modified to suit the heterogeneity of the platform. Thirdly, the models and the frequency scaling algorithm were completely rethought to take into considerations the asynchronism in the communication and computation. All these models and algorithms were applied to message passing applications with iterations and evaluated over either SimGrid simulator or Grid'5000 platform. The experiments showed that the

proposed algorithms are efficient and outperform existing methods such as the energy and delay product. They also introduce a small runtime overhead and work online without any training or profiling.

KEY WORDS: Dynamic voltage and frequency scaling, Grid computing, Energy optimization, parallel applications with iterations and online frequency scaling algorithm.

RÉSUMÉ

Optimisation de la consommation énergétique des applications parallèles avec des itérations en réduisant la fréquence des processeurs

Ahmed Badri Muslim Fanfakh
Université de Franche-Comté, 2016

Encadrants: Raphaël Couturier and Jean-Claude Charr

Au cours des dernières années, l'informatique "green" est devenue un sujet important dans le calcul intensif. Cependant, les plates-formes informatiques continuent de consommer de plus en plus d'énergie en raison de l'augmentation du nombre de noeuds qui les composent. Afin de minimiser les coûts d'exploitation de ces plates-formes de nombreuses techniques ont été étudiées, parmi celles-ci, il y a le changement de la fréquence dynamique des processeurs (DVFS en anglais). Il permet de réduire la consommation d'énergie d'un CPU, en abaissant sa fréquence. Cependant, cela augmente le temps d'exécution de l'application. Par conséquent, il faut trouver un seuil qui donne le meilleur compromis entre la consommation d'énergie et la performance d'une application. Cette thèse présente des algorithmes développés pour optimiser la consommation d'énergie et les performances des applications parallèles avec des itérations synchrones et asynchrones sur des clusters ou des grilles. Les modèles de consommation d'énergie et de performance proposés pour chaque type d'application parallèle permettent de prédire le temps d'exécution et la consommation d'énergie d'une application pour toutes les fréquences disponibles.

La contribution de cette thèse peut être divisé en trois parties. Tout d'abord, il s'agit d'optimiser le compromis entre la consommation d'énergie et les performances des applications parallèles avec des itérations synchrones sur des clusters homogènes. Deuxièmement, nous avons adapté les modèles de performance énergétique aux plates-formes hétérogènes dans lesquelles chaque noeud peut avoir des spécifications différentes telles que la puissance de calcul, la consommation d'énergie, différentes fréquences de fonctionnement ou encore des latences et des bandes passantes réseaux différentes. L'algorithme d'optimisation de la fréquence CPU a également été modifié en fonction de l'hétérogénéité de la plate-forme. Troisièmement, les modèles et l'algorithme d'optimisation de la fréquence CPU ont été complètement repensés pour prendre en

considération les spécificités des algorithmes itératifs asynchrones. Tous ces modèles et algorithmes ont été appliqués sur des applications parallèles utilisant la bibliothèque MPI et ont été exécutés avec le simulateur Simgrid ou sur la plate-forme Grid'5000. Les expériences ont montré que les algorithmes proposés sont plus efficaces que les méthodes existantes. Ils n'introduisent qu'un faible surcoût et ne nécessitent pas de profilage au préalable car ils sont exécutés au cours du déroulement de l'application.

MOTS-CLÉS: l'ajustement dynamique de la tension et de la fréquence d'un processeur, Grille de calcul, l'optimisation de l'énergie des applications parallèles avec des itérations.

CONTENTS

List of Figures

List of Tables

LIST OF ALGORITHMS

ABBREVIATIONS

AIAC ..... **A**synchronous **I**terations and **A**synchronous **C**ommunications

BLP ..... **B**it **L**evel **P**arallelism

BT ..... **B**lock **T**ridiagonal

CG ..... **C**onjugate **G**radient

CPU ..... **C**entral **P**rocessing **U**nit

CUDA .... **C**ompute **U**nified **D**evice **A**rchitecture

DLP ..... **D**ata **L**evel **P**arallelism

DVFS ..... **D**ynamic **V**oltage and **F**requency **S**caling

EDP ..... **E**nergy and **D**elay **P**roduct

EP ..... **E**mbarrassingly **P**arallel

EPSA .... **E**nergy and **P**erformance **S**caling **A**lgorithm

FLOPS ... **FL**oating-point **O**perations **P**er **S**econd

FT ..... Fast **F**ourier **T**ransform

GMRES .. **G**eneral **M**inimum **Res**idual

GPU ..... **G**raphical **P**rocessing **U**nit

HSA ..... **H**eterogeneous **S**caling **A**lgorithm

ILP ..... **I**nstruction **L**evel **P**arallelism

LAN ..... **L**ocal **A**rea **N**etwork

LLP ..... **L**oop **L**evel **P**arallelism

LU ..... **L**ower-**U**pper

MaxDist .. **Max**imum **Dist**ance

MG ..... **M**ulti-**G**rid

MIMD ..... **M**ultiple **I**nstruction and **M**ultiple **D**ata

MISD ..... **M**ultiple **I**nstruction and **S**ingle **D**ata

MPI ..... **M**essage **P**assing **I**nterface

MPICH ... **M**essage **P**assing **I**nterface of **Ch**ameleon

MS ..... **M**ulti-**S**plitting

NAS ..... **N**umerical **A**eronautical **S**imulation

NASA .... **N**ational **A**eronautics and **S**pace **A**dministrations

OPENCL . **Open C**omputing **L**anguage

OPENMP . **Open M**ulti-**P**rocessing

RENATER **Ré**seau **Na**tional de **T**élécommunications pour la **T**echnologie, l'**E**nseignement et la **R**echerche

SIAC ..... **S**ynchronous **I**terations and **A**synchronous **C**ommunications

SIMD ..... **S**ingle **I**nstruction and **M**ultiple **D**ata

SISC ..... **S**ynchronous **I**terations and **S**ynchronous **C**ommunications

SISD ..... **S**ingle **I**nstruction and **S**ingle **D**ata

SP ..... **S**calar **P**entadiagonal

TLP ..... **T**hread **L**evel **P**arallelism

WAN ..... **W**ide **A**rea **N**etwork

DEDICATION

I dedicate this dissertation to my beloved wife Dania, my children: Elias, Yasser and Mehdi. My family, your unwavering support, encouragement, and constant love throughout this Ph.D. study was incredible. My wife and children have supported me with prayers, encouraging words that gave me strength to make this dream a reality. I am truly thankful to my God for having you in my life. I love you. This dissertation has also been dedicated to my parents. From an early age, they instilled in me a desire to learn, and made sacrifices so as to I have access to a high- quality education. Without their support and guidance, I would not be where I am today. Dear Mom and Dad, I can not thank you enough for all the support and love you have given me. I know you would have been very proud of me. Thank you so much for your love. I dedicate this work as well to my brothers, sisters, and their families, whose support and encouragement helped me to follow through and not give up. I would like to extend my dedication to my friends who supported me.

ACKNOWLEDGEMENTS

The long journey of my Ph.D. study has finished. It is with great pleasure that I acknowledge my debts to those who have greatly contributed to the success of this dissertation. It was only through support and encouragement of many that I have been able to complete this amazing journey.

Foremost, I would like to express my sincere gratitude to my supervisors: Prof. Dr. Raphaël Couturier and Asst. Prof. Dr. Jean-Claude Charr for their continuous support, encouragement, and advice they have provided throughout my Ph.D. study. Their patience, motivation, enthusiasm, and immense knowledge taught me a lot. Their tireless guidance has helped me immensely in researching and writing this dissertation. I have been extremely lucky to have supervisors who cared so much about my work, and who responded to my questions and queries so promptly.

Besides my supervisors, I would like to express my gratitude to Prof. Dr. Jean-Marc Pierson and Assoc Prof. Dr. Fabienne Jézéquel (HDR) for accepting to review my dissertation and for their insightful and appreciated comments. I would like to thank also Prof. Dr. Nabil Abdennadher for accepting to participate in my dissertation committee.

I would like to gratefully acknowledge the University of Babylon, Iraq for financial support as well as I would also like to express my thanks to University of Franche-Comté for the received support.

My appreciation and thanks go to the members of the team AND (*Algorithmique Numérique Distribuée*) for the warm and friendly atmosphere in which they allowed me to work. These include Jacques Bahi, Pierre-Cyrille Héam, Abdallah Makhoul, Jean-François Couchot, Ahmed Mostefaoui, Yousra Ahmed Fadil, Pierre Saenger, Zeinab Fawaz, Amor Lalama, Nesrine Khernane, Stéphane Domas, Mohammed Bakiri, Michel Salomon, Karine Deschinkel, Christophe Guyeux, Mourad Hakem, David Laiymani, Gilles Perrot, Fabrice Ambert, Christian Salim, Santiago Costarelli, Carol Habib, Hassan Moustafa Harb and Ke Du. I would like to give a special thanks for Asst. Prof. Dr. Arnaud Giersch who helped me understand the simulation over SimGrid simulator. I would like also to thank my colleague, Dr. Ali Kadhum Idrees University of Babylon for his help. I would like to express my thanks and my best wishes to Ingrid Couturier for all the received assistance during my study.

Finally, I would like to thank all my friends and people who encouraged and supported me along the way.

INTRODUCTION

1. GENERAL INTRODUCTION

The need and the demand for more computing power have been increasing since the birth of the first computing unit and they are not expected to slow down in the coming years. To meet these demands, at first the frequency of the CPU was regularly increased until reaching the thermal limit. Then, researchers and supercomputers constructors have been regularly increasing the number of computing cores and processors in supercomputers. Many parallel and distributed architectures, such as multi-core, clusters and grids, were implemented in order to obtain more computing power. This approach consists in using at the same time many computing nodes to solve a big problem that cannot be solved on a single node. These two approaches are the most common up to now to get more computing power, but they increase the energy consumption of the resulting computing architecture. Indeed, the power consumed by a processor exponentially increases when its frequency is increased and a platform consisting of N computing nodes consumes as much as the sum of the power consumed by each computing node. As an example, the Chinese supercomputer Tianhe-2 had the highest FLOPS in November 2015 according to the Top500 list [7]. However, it was also the most power hungry platform with more than 3 million cores consuming around 17.8 megawatts. Moreover, according to the U.S. annual energy outlook 2015 [8], the price of energy for 1 megawatt per hour was approximately equal to $70. Therefore, the price of the energy consumed by the Tianhe-2 platform is approximately more than $10 million each year. Moreover, the platform generates a lot of heat and to prevent it from overheating a cooling infrastructure [85] which consumes a lot of energy must be implemented. High CPU's temperatures can also drastically increase its energy consumption, see [86] for more details. An efficient computing platform must offer the highest number of FLOPS per watt possible, such as the Shoubu-ExaScaler from RIKEN which became the top of the Green500 list in November 2015 [6]. This heterogeneous platform executes more than 7 GFlops per watt while only consuming 50.32 kilowatts.

For all these reasons energy reduction has become an important topic in the high performance computing (HPC) field. To tackle this problem, many researchers use DVFS (Dynamic Voltage and Frequency Scaling) operations which reduce dynamically the frequency and voltage of cores and thus their energy consumption [69]. Indeed, modern CPUs offer a set of acceptable frequencies which are usually called gears, and the user or the operating system can modify the frequency of the processor according to its needs. However, DVFS reduces the number of FLOPS executed by the processor which may increase the execution time of the application running over that processor. Therefore researchers try to reduce the frequency to the minimum when processors are idle (waiting for data from other processors or communicating with other processors). Moreover, depending on their objectives, they use heuristics to find the best frequency scaling factor during the computation. If they aim for performance they choose the best frequency

scaling factor that reduces the consumed energy while affecting as little as possible the performance. On the other hand, if they aim for energy reduction, the chosen frequency scaling factor must produce the most energy efficient execution without considering the degradation of the performance. Whereas, it is important to notice that lowering the frequency to the minimum value does not always give the most energy efficient execution due to energy leakage that increases the total energy consumption of the CPU when the execution time increases. However, a more important question is how to select the best frequency gears that minimize the total energy consumption and the maximize the performance of a parallel application, running over a parallel platform, at the same time?

2. Motivation of the Dissertation

The main objective of an HPC system such as clusters, grids and supercomputers is to execute as fast as possible a given task over that system. Hence, using DVFS to scale down the frequencies of the CPUs composing the system to reduce their energy consumption, it can also significantly degrade the performance of the executed program, especially if it is compute bound. A compute bound program contain a lot of computations and a relatively small amount of communicators and Inputs/Outputs operations. The execution time of the program is directly dependent on the computing powers of the CPUs and their selected frequencies. Therefore, the chosen frequency scaling factor must give the best possible trade-off between the energy reduction and the performance of the parallel application.

On the other hand, the relation between energy consumption and the execution time of parallel applications is complex and non-linear. It is very hard to optimize both the energy consumption and the performance of parallel applications when scaling the frequency of the processors executing them because one affects the other. In order to evaluate the impact of scaling down the CPU's frequency on its energy consumption and computing power, mathematical models should be defined to predict them for different frequencies.

Furthermore, researchers use different optimization strategies to select the frequencies of the CPUs. They might be executed during the execution of the application (online) or during a pre-execution phase (offline). In our opinion a good approach should minimize the energy consumption while preserving the performance at the same time. Finally, it should also be applied to the application during its execution without requiring any training or profiling and with minimal overhead.

3. Main Contributions of this Dissertation

The main objective of this work is to minimize the energy consumption of parallel applications with iterations running over clusters and grids while preserving their performance. The main contributions of this work can be summarized as follows:

I) Energy consumption and performance models for synchronous and asynchronous message passing applications with iterations were developed. These models take into consideration both the computation and communications times of these applications in addition to their relation to the frequency scaling factors.

II) The parallel applications with iterations were executed over different parallel architectures such as: homogeneous local cluster, heterogeneous local cluster and distributed clusters (grid platform). The main goal behind using these different platforms is to study the effect of the heterogeneity in the computing powers of the the commuting nodes and the heterogeneity in the communication networks which connect these nodes on the energy consumption and the performance of parallel applications with iterations.

III) Depending on the proposed energy consumption and the performance models, a new objective function to optimize both the energy consumption and the performance of the parallel applications with iterations at the same were defined. It computes the maximum distance between the predicted energy consumption and the predicted performance curves to define the best possible trade-off between them.

IV) New online frequency selecting algorithms for clusters and grids were developed. They use the new objective function and select the frequency scaling factors that simultaneously optimize both the energy consumption and performance. They have a very small overhead when comparing them to other methods in the state of the art and they work without training and profiling.

V) The proposed algorithms were applied to the NAS parallel benchmarks [57] and the Multi-splitting method. These applications offer different computations to communications ratios and a good testbed to evaluate the proposed algorithm in different scenarios.

VI) The proposed algorithms were evaluated over the SimGrid simulator [18] which offers flexible and easy tools to built different types of parallel architectures. Furthermore, real experiments were conducted over Grid'5000 testbed [3] and compared with the simulated ones. The experiments were conducted over different number of nodes and different platform scenarios.

VII) All the proposed methods were compared with either Rauber and Rünger [66] method or Spiliopoulos et al. [75] objective function. Both the simulation and real experiments showed that the proposed methods give better energy to performance trade-offs than the other methods.

4. Dissertation Outline

The dissertation is organized as follows: chapter 1 presents different types of parallel architectures and parallel applications with iterations. It also presents an energy consumption model from the state of the art that can be used to measure the energy consumption of these applications. Chapter 2 describes the proposed energy and performance optimization method for synchronous applications with iterations running over homogeneous clusters. Chapter 3 presents two algorithms for the energy and performance optimization of synchronous applications with iterations running over heterogeneous clusters and grids. In chapter 4 the energy and performance models and the optimization method are adapted for asynchronous iterative applications running over grids. Finally, this dissertation ends with a summary and some perspective works.

I

SCIENTIFIC BACKGROUND

1

Parallel Architectures and Iterative Applications

1.1/ Introduction

Most of the software applications are structured as sequential programs. The structure of the program code is a series of instructions that are executed successively one after the other. For many years until a short time, with each new generation of microprocessors, users of sequential applications expected that these applications should run faster over them than over the previous ones. Nowadays, this idea is no longer valid since recent releases of microprocessors have many computing units that are embedded in one chip and programs are running only over one computing unit sequentially. Indeed, new applications have significantly improved their performance over new architectures in parallel compared to traditional applications. To improve the performance of applications, they should be parallelized and executed simultaneously over all available computing units. Moreover, parallel applications should be optimized to the parallel hardwares that will execute them. Therefore, parallel applications and parallel architectures are closely tied together. For example, the energy consumption of one parallel system mainly depends on both: (1) parallel applications and (2) parallel architectures. Indeed, an energy consumption model or any measurement system depends on many specifications, some of them are related to the parallel hardware features such as: (1) the frequency of processor, (2) the power consumption of processor and (3) the communication model. Others rely to the parallel application such as: (1) the computation time and (2) the communication time of the application.

This work of this thesis is focused on studying the iterative parallel applications, where different parallel architectures are used to execute them in parallel, while optimizing their energy consumptions. In this context, this chapter gives a brief overview about parallel hardware architectures and parallel iterative applications. Also, it discusses an energy model proposed by other authors used to measure the energy consumption of these applications. The reminder of this chapter is organized as follows: section 1.2 describes different types of parallelism and different types of parallel platforms. It also explains some models of parallel programming. Section 1.3 discusses both types of parallel iterative methods, synchronous and asynchronous ones and comparing them. Section 1.4, presents a well accepted energy model from the state of the art that can be used to measure the energy consumption of parallel iterative applications when the frequency of processor is changed. Finally, section 1.5 summarizes this chapter.

1.2/ PARALLEL COMPUTING ARCHITECTURES

The process of executing the calculations simultaneously over many computing units is called parallel computing. Its main principle refers to the ability of dividing a large problem into smaller sub-problems that can be solved at the same time [9]. Solving the sub-problems of one main problem in parallel is carried out in parallel on multiple processors. Indeed, a parallel architecture can be defined as a computing system that is composed of many processing elements, which are connected via a network model and some tools that are used to make the processing units work together [31]. In other words, the parallel computing architecture consists of software and hardware resources. Hardware resources are: (1) the processing units, (2) the memory model and (3) the network system that connects them. Software resources include (1) the specific operating system, (2) the programming language and (3) the compile or the runtime libraries. Besides, parallel computing may have different levels of parallelism that can be performed in a software or a hardware level. Five types of parallelism levels have been defined as follows:

- **Bit-level parallelism (BLP)**: The appearance of very-large-scale integration (VLSI) in 1970s has been viewed as the first step towards parallel computing. It is used to increase the number of bits in the word size which is processed by a processor as illustrated in the figure 1.1. For many successive years, the number of bits have been increased starting from 4 bit to 64 bit microprocessors. For example nowadays, the recent x86-64 architecture is the most common architecture. For a given application, the biggest the word size is the lesser instructions to be executed by the processor.

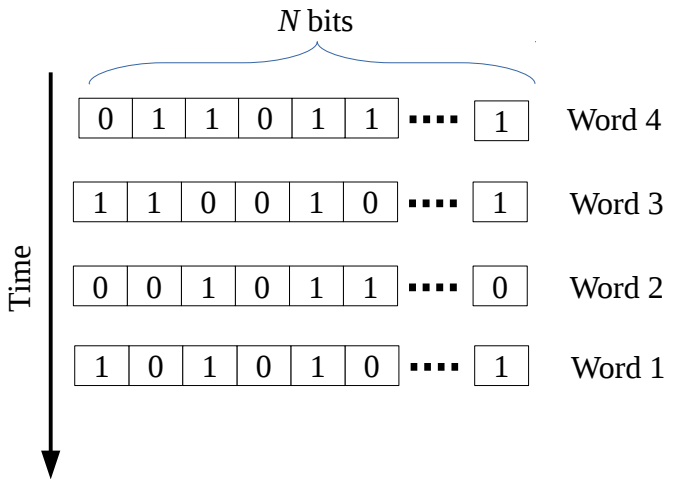

Figure 1.1: Bit-level parallelism

- **Data-level parallelism (DLP)**: Data parallelism is the process of distributing data vector between processors, where each one performs the same operations on its data sub-vector. Therefore, many arithmetic operations can be performed on the same data vector in a simultaneous manner. This type of parallelism can be used in many programs, especially in the area of scientific computing. Usually, data-parallel operations are only provided to arrays operations, for example, as shown in figure

1.2. Vector multiplication, image and signal processing can be considered as an example of applications that use this type of parallelism.

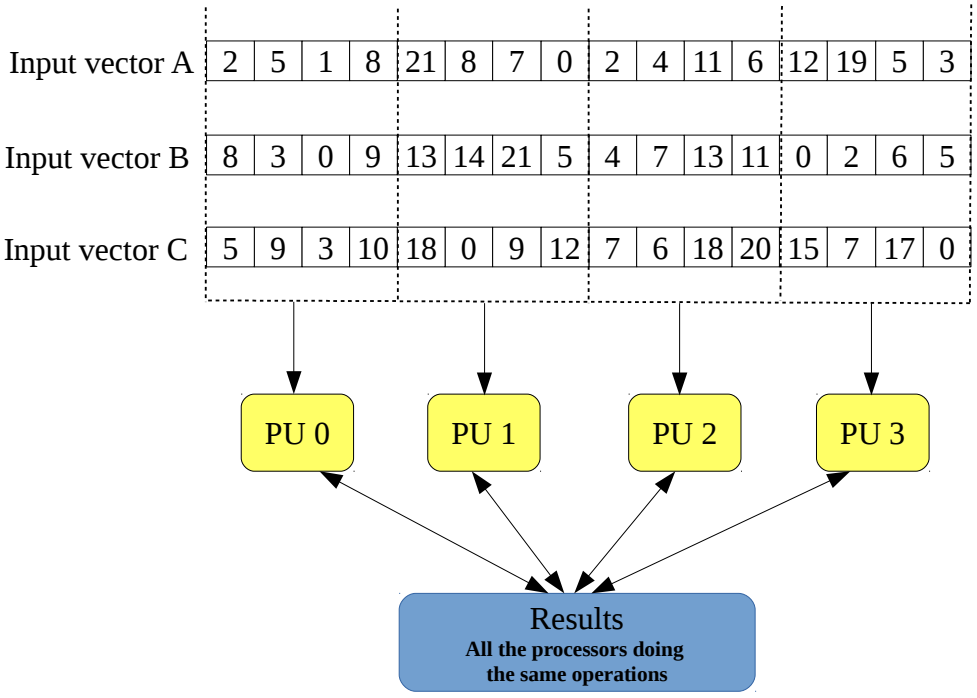

Figure 1.2: Data-level parallelism

- **Instruction-level parallelism (ILP)**: Generally, a sequential program is composed of many instructions. These instructions can be executed in parallel at the same time, if each one of them is independent from the others. In particular, the parallelism can be achieved in instruction level by using a pipeline. It means the input and output times of each instruction is overlapped by computations from other instructions. For example, if we have two instructions: I_1 and I_2, they are independent if there is no control and no data dependency between them. In pipeline stages, the execution of each instruction is divided into multiple steps. Then, they can be overlapped with the steps of other instructions by a pipeline hardware unit. Figure 1.3 demonstrates four instructions, where each one has four steps denoted as: (1) fetch, (2) decode, (3) execute and (4) write. Thus, they are implemented in hardware units by pipeline.

- **Thread-level parallelism (TLP)**: It is also known as task-level parallelism. According to Moore's law [17], the number of transistors in a processor doubles each two years to increase its performance. Cache and main memory sizes must also be increased in order to avoid data bottlenecks. However, increasing the number of transistors may generate some issues: (1) the first issue is related to drastically increase in cache size, which leads to a large access time. (2) the second issue is related to the huge increase in the number of the transistors per CPU, which can increase significantly the heat dissipation. Thus, CPUs constructors couldn't increase the frequency of the processor anymore due to these reasons. Therefore, they created multi-core processors. With multi-core processors, programmers subdivide their programs into multiple tasks which can be then executed in parallel over

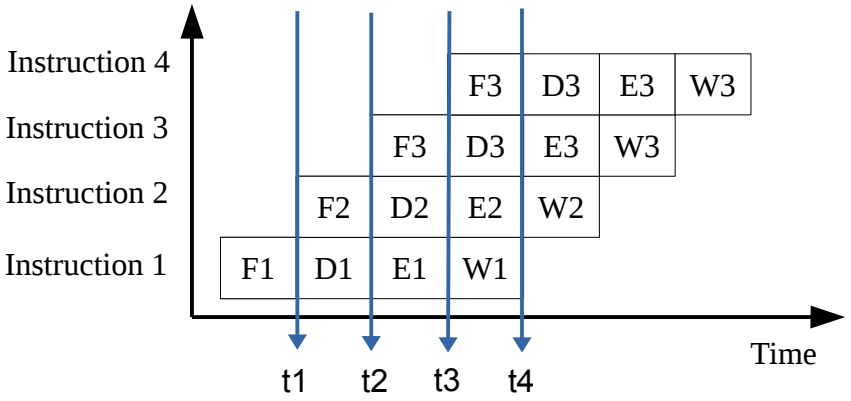

Figure 1.3: Instruction-level parallelism by pipelines

them to improve the performance, see figure 1.4. Each processor can have individual threads or multiple threads dedicated to each task. A thread can be defined as a part of the parallel program that shares processor resources with other threads.

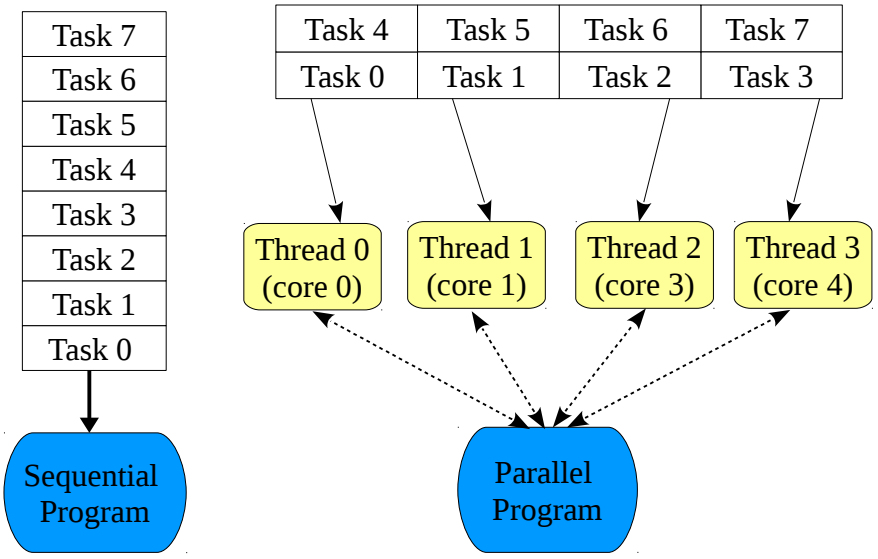

Figure 1.4: Thread-level parallelism

Therefore, the execution time of a sequential program that is composed of N tasks, is the sum of the execution times of all tasks. Thus, it is expressed as follows:

$$Sequential\ execution\ time = \sum_{i=1}^{N} T_i \tag{1.1}$$

Whereas, if tasks are executed synchronously over multiple processing units in parallel, the execution time of the program is defined as the execution time of the task that has maximum the execution time (the slowest task) as follows:

$$Parallel\ execution\ time = \max_{i=1,...,N} T_i \tag{1.2}$$

- **Loop-level parallelism (LLP)**: Many algorithms execute iteratively the same program portion, computations, many times using different forms of loop statements. At each iteration, the program needs to scan a large data structure such as an array structure to perform the arithmetic calculations. Inside the loop structure, there are many instructions that are dependent or independent. In a sequential loop execution, the i iteration must be executed after the completion of the $(i-1)$ iteration. If each iteration is independent from the others, then all iterations' instructions can be distributed over many processors to be executed in parallel, for example, see figure1.5. In the parallel programming languages, this type of loop is called the *parallel loop*.

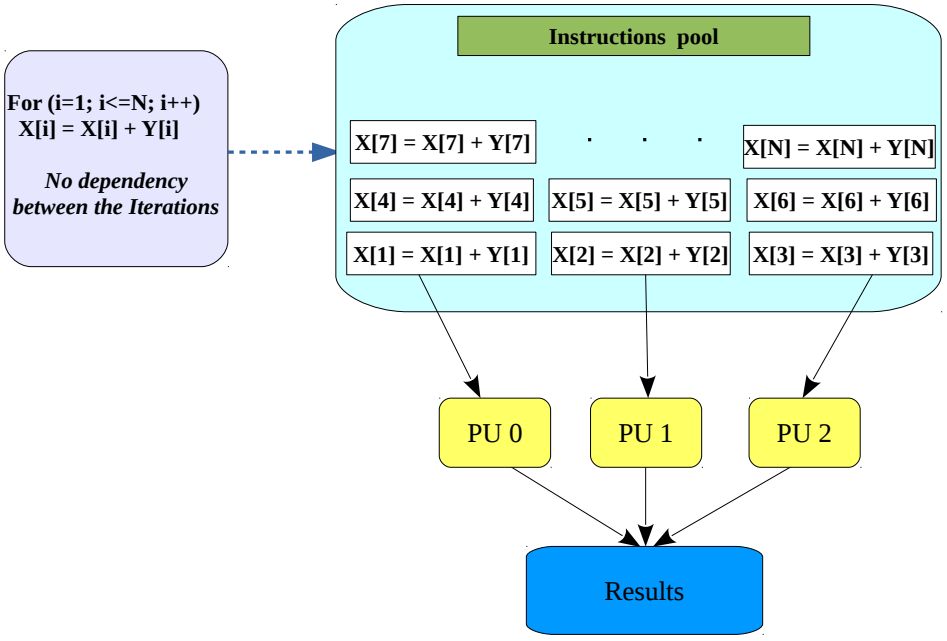

Figure 1.5: Loop-level parallelism

The execution time of the parallel loop portion can be computed as the execution time of a sequential loop portion has N_{iter} iterations divided by the number of the processing units $N_{processors}$ as follows:

$$Parallel\ loop\ time = \frac{Sequential\ loop\ time}{N_{processors}} = \frac{\sum_{i=1}^{N_{iter}} Time\ of\ iter_i}{N_{processors}} \tag{1.3}$$

For more details about the levels of parallelism see [67, 61, 38, 62].

1.2.1/ TYPES OF PARALLEL PLATFORMS

The main goal behind using a parallel architecture is to solve a big problem faster. A collection of processing elements must work together to compute the final solution of the main problem. Many different architectures have been proposed and classified according

to parallelism in instruction and data streams. In 1966, Michel Flynn has proposed a simple model to categorize all computers models that is still useful until now [33]. His taxonomy is based on considering the data and the operations performed on this data to classify the computing systems into four types as follows:

- **Single instruction, single data (SISD) stream**: A single processor that executes a single instruction stream (i.e executing one data stream stored in an individual memory model, see figure 1.6). The conventional sequential computer, according to Von Neumann model [80], also called the Uniprocessors can be viewed as an example of this type of architecture.

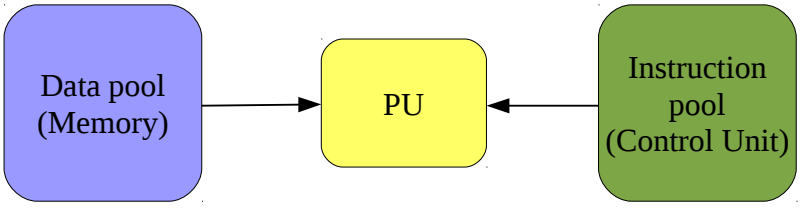

Figure 1.6: SISD machine architecture

- **Single instruction, multiple data (SIMD) stream**: All processors execute the same instructions on different data. Each processor stores the data in its local memory. Then, they communicate with each others typically via a simple communication model, see figure 1.7. Many scientific and engineering applications are referred to this type of parallel scheme. Vector and array processors are well known examples of this type. Examples about the applications executed over this architecture: (1) graphics processing, (2) video compression and (3) medical image analysis applications.

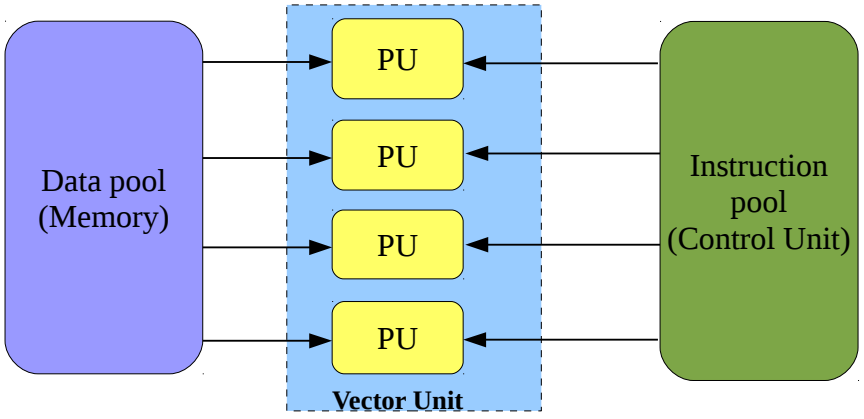

Figure 1.7: SIMD machine architecture

- **Multiple instruction, single data (MISD) stream**: Many operations from multiple processing elements are executed over the same data stream. Each processing

element has its local memory to store the private program instructions. Then, these instructions are applied to unique global memory data stream as in figure 1.8. While the MISD machine is not commonly used, there are some interesting uses such as the systolic arrays and dataflow machines.

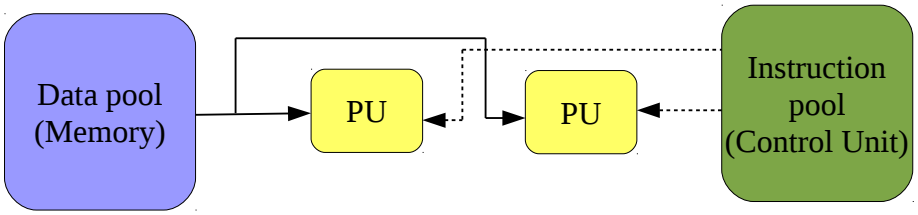

Figure 1.8: MISD machine architecture

- **Multiple instruction, Multiple data (MIMD) stream**: There are multiple processing elements, each one has a separate instruction and local data memories. At any time, different processing elements may be used to execute different instructions on different data fragment, see figure 1.9. There are two types of MIMD machines: the shared memory and the message passing MIMD machines. In the former, processors communicate via a shared memory model, while in the latter, each processor has its own local memory and all processors communicate with each others via a communication network model. The multi-core processors, local clusters and grid systems are some examples for MIMD machine. Many applications have been developed based on this architecture such as computer-aided design, computer-aided manufacturing, simulation, modeling, iterative applications and so on.

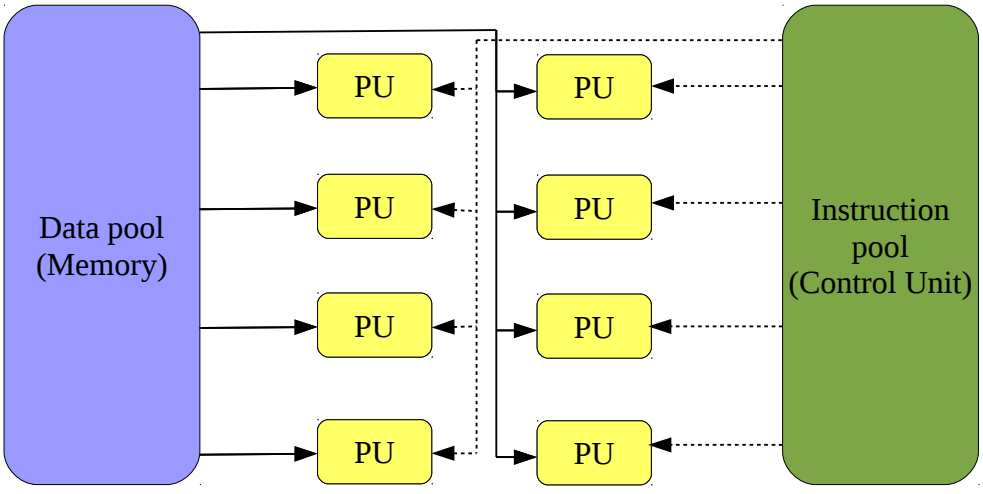

Figure 1.9: MIMD machine architecture

For more details about this architectural taxonomy see [42, 76, 60, 27].

The work of this thesis is dedicated to MIMD machine's architecture. Therefore, we discuss in this chapter some of the commonly used parallel architectures that belong to MIMD machines. As explained before, MIMD architectures can be classified into two types, the shared memory and the distributed message passing ones. Furthermore, these classifications are based on how MIMD processors access the memory model. The shared MIMD machine communication topology can be bus-based, extended or hierarchical type. Whereas, the distributed memory MIMD machine may have hypercube or mesh interconnected networks. In the following some well known MIMD parallel computing platforms are explained:

- **Multi-core processors**: The multi-core processor is a single chip component with two or more processing units. These processing units are called cores, which are connected to each other via a main memory model as in the figure 1.10. Each individual core has its own cache memory to store data. Moreover, each core may have one or more threads to execute a specific programming task as shown in the thread-level parallelism. Historically, the multi-cores of the CPU began as two-core processors, then the number of cores doubled with each semiconductor process generation [46]. The graphic processing units (GPU) use extensively the multi-core architecture, the NVIDIA GeForce TITAN Z has 5700 cores in the year of 2015 [2]. While, in the same year a general-purpose microprocessor (CPU) has a lot less cores, for example the TILE-MX processor from Tilera has 100 cores [4]. For more details about the multi-core processors see [64].

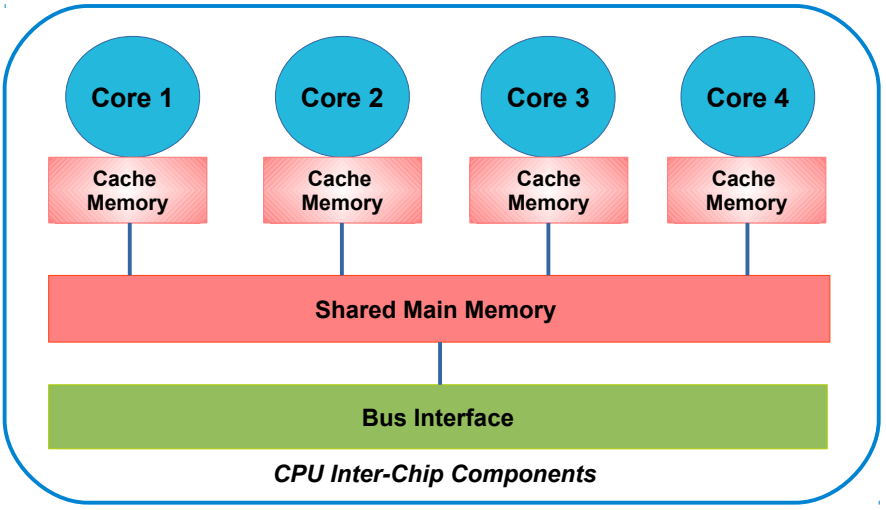

Figure 1.10: Multi-core processor architecture

- **Local Cluster**: is a collection of independent computers that are connected to each other via a high speed local area network (LAN) with low latency and big bandwidth. Moreover, each node communicates with other nodes using messages. All the nodes in the cluster must be controlled by one node called the master node, which is a specific node used to handle the scheduling and the management of the other nodes as shown in the figure 1.11. Usually, all the nodes are homogeneous, they have the same specifications in term of computing power and memory. Also, all the computing nodes in the cluster run the same operating system. See [84, 53] for more information about the cluster and its applications.

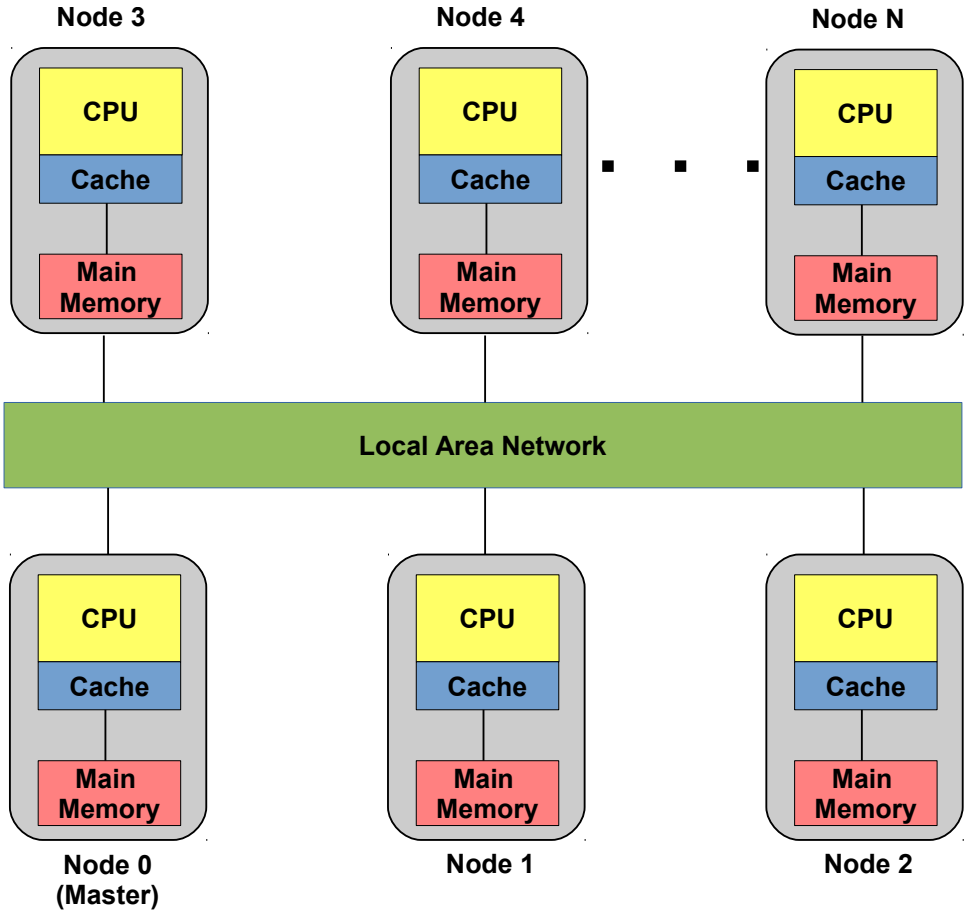

Figure 1.11: Local cluster architecture

- **Grid (Distributed clusters)**: Grid is a collection of computing clusters from different sites that are connected via a wide area network (WAN). In particular, different local clusters compose the grid are geographically located far away from each others. Usually, each cluster is composed of homogeneous nodes, which are different from nodes of the other clusters located in different sites. These nodes can be different in their hardware and software specifications (i.e their computing power, their memory size, their operating system and their network: latency and bandwidth). Figure 1.12 presents an example of a grid that is composed of three heterogeneous clusters that are located in different sites and connected via a wide area network. Furthermore, the grid can refer to an infrastructure that applies the integration and the collaboration by using a collection of different computers, networks, database servers and scientific devices, which belong to many companies and universities. Therefore, wide heterogeneous computing resources are available to be used simultaneously by different users. Note that, the main bottleneck of the grid is the high latency communications between the nodes from different sites. See [54] for more information about the grid and its applications.

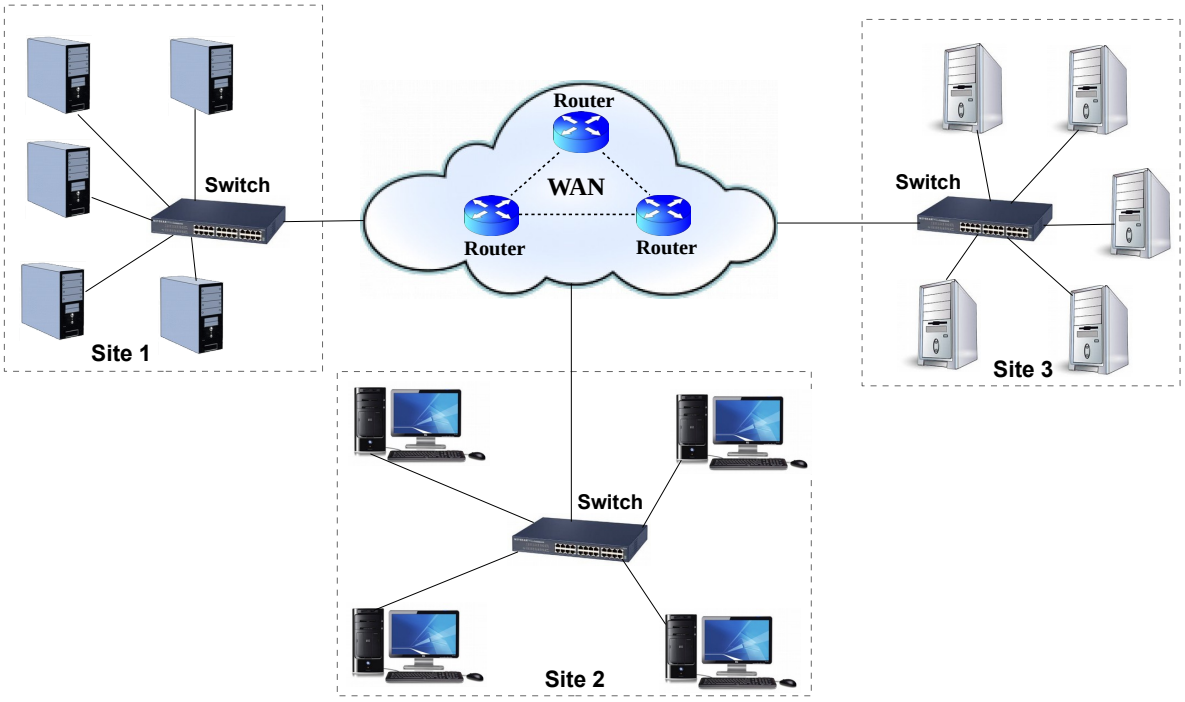

Figure 1.12: Grid architecture

1.2.2/ PARALLEL PROGRAMMING MODELS

Many parallel programming languages and libraries have been developed to explore the computing power of the parallel architectures. In this section, two types of parallel programming languages are investigated: (1) shared and (2) distributed programming models. Moreover, each type is divided into two subcategories according to their supporting level for the number of computing units from which the parallel platform is composed. Figure 1.13 presents this classification hierarchy of the parallel programming models.

Many programming interfaces and libraries have been developed to compile and run the parallel applications over the parallel architectures. In the following, some examples for each type of the parallel programming models are discussed:

- **Local cluster programming models**
  - **MPI** [39] is the Message Passing Interface and it is considered as a standardization dedicated to message passing in a distributed memory environment. The first version of MPI was designed by a group of researchers in 1991. It is a specification and have been implemented in many programming languages such as C, Fortran and Java. The MPI functions are not only limited to point to point operations for sending and receiving messages, there are many others collective operations such as gathering and reduction operations. While MPI is not designed for grid, it is widely used as the communication interface for grid applications [16]. In this work, MPI was used in programming our algorithms and applications which are implemented in both Fortran and C programming languages.

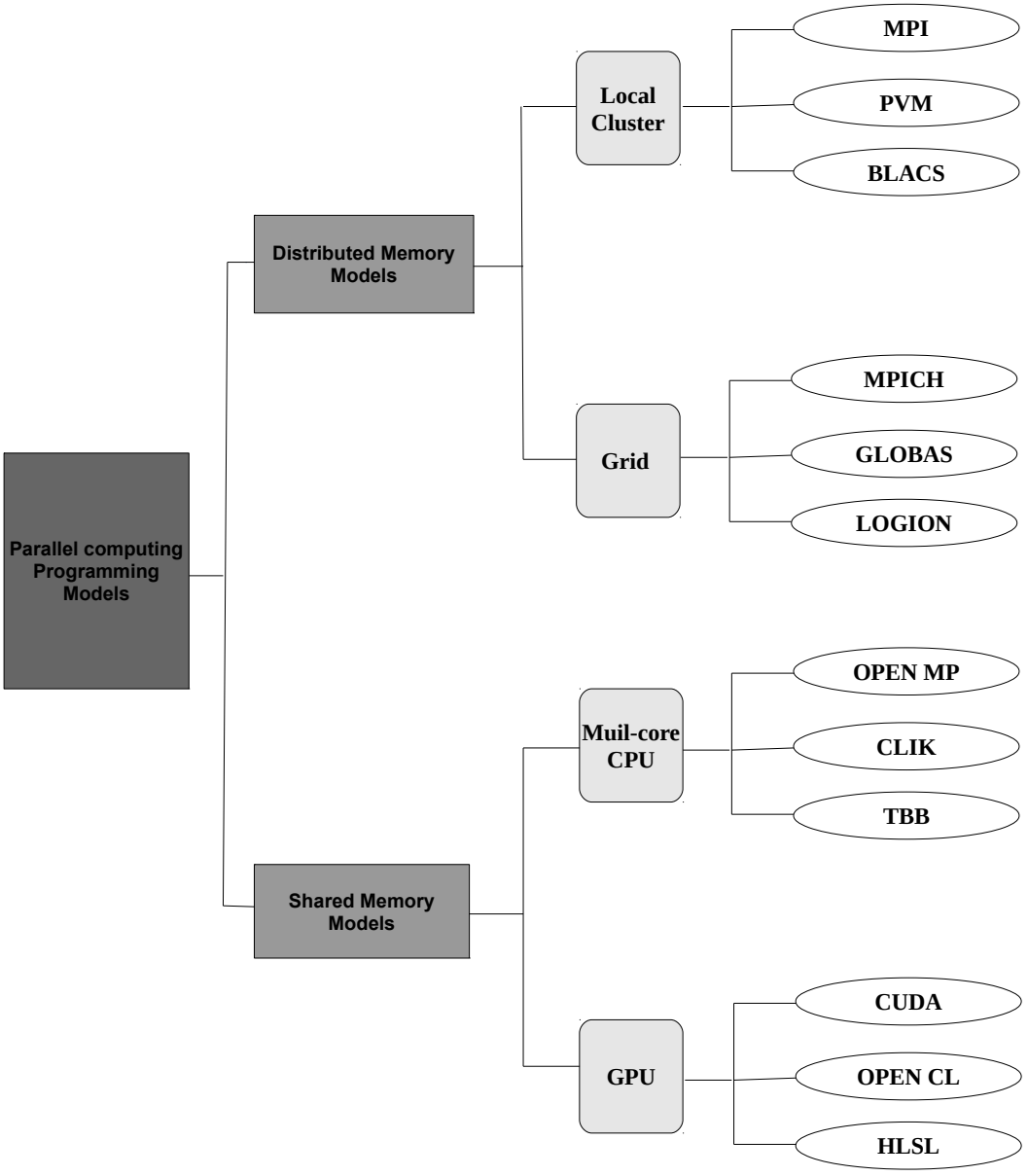

Figure 1.13: The classification of the parallel Programming Models

- **Multi-core CPU programming models**

  – **OpenMP** [19] is a parallel programming tool dedicated to shared memory architectures. The main goal of using this programming model is to provide a standard and portable API (application programming interface) to write shared memory parallel programs. It can be used with many programming languages such as C, C++ and Fortran in order to support different types of shared memory platforms such as multi-core processors. OpenMP uses multi-threading, which is a model in parallel programming that uses a master thread to control a set of slave threads. Each thread can be executed in parallel by assigning it to a processor. Moreover, OpenMP can be used with MPI to support hybrid

platforms which have shared and distributed memory models at the same time.

- **GPU programming models**
  - **CUDA** [26] Modern graphical processing units (GPUs) have increased its chip-level parallelism. Current NVIDIA GPUs consist of many-cores processors that have thousands of cores. To make their GPUs a general purpose computing processor in 2007 the NVIDIA has developed CUDA a parallel programming language. A CUDA program has two parts: host and kernels. The host code is sequentially executed over the CPU. While, the kernels are executed in parallel over the GPUs.
  - **OpenCL**[78] is for Open Computing Language. It is a parallel programming language dedicated for heterogeneous platforms composed of CPUs and GPUs. The first release of this language has initially been developed by Apple in 2008. Functions that are executed over OpenCL devices are called kernels. They are portable and can be executed on any computing hardware such as CPU or GPU cores.

1.3/ ITERATIVE METHODS

In this work, we are interested in solving system of linear equations which are very common in the scientific field. A system of linear equations can be expressed as follows:

$$Ax = b \tag{1.4}$$

Where A is a two dimensional matrix of size $N \times N$, x is the unknown vector, and b is a vector of constant, each of size N. There are two types of solution methods to solve this linear system: the **direct** and the **iterative methods**. A direct method executes a finite number of steps, depending on the size of the linear system and gives the exact solution of the system. If the problem is very big, this method is expensive or its solution is impossible in some cases. On the other hand, methods with iterations execute the same block of instructions many times. The number of iterations can be predefined or the application iterates until a criterion is satisfied. Iterative methods are methods with iterations that start from an initial guess and improve successively the solution until reaching an acceptable approximation of the exact solution. These methods are well adapted for large systems and can be easily parallelized.

A sequential iterative algorithm is typically organized as a series of steps essentially of the form:

$$X^{(k+1)} \longleftarrow F(X^k) \tag{1.5}$$

Where F is one or set of operations applied to the data vector X^k to produce the new data vector $X^{(k+1)}$. The operation F is applied sequentially many times until satisfying the convergence condition as in the algorithm 1.

The sequential iterative algorithm at each iteration computes the value of the relative error, which is called the residual and denoted as R. This error value can be computed

Algorithm 1 The iterative sequential algorithm

1: Initialize the vector X^0 randomly
2: **for** $k := 1$ to *convergence* **do**
3: $X^{(k+1)} = F(X^k)$
4: **end for**

as the maximum difference between the data components of the vectors of the last two successive iterations as follows:

$$R = \max_{i=1,\ldots,N} \left| X_i^{(k+1)} - X_i^k \right| \tag{1.6}$$

Where N is the size of the vector X. Then, the iterative sequential algorithm stops iterating if the maximum error between the last two successive solution vectors, as in 1.6, is less than or equal to a threshold value. Otherwise, it replaces the new vector $X^{(k+1)}$ with the old vector X^k and computes a new iteration.

1.3.1/ SYNCHRONOUS PARALLEL ITERATIVE METHOD

The sequential iterative algorithm 1 can be parallelized by executing it on many computing units. To solve this algorithm on M computing units, first the elements of the problem vector X must be subdivided into M sub-vectors, $X^k = (X_1^k, \ldots, X_M^k)$. Each sub-vector can be solved independently on one computing unit as follows:

$$X_i^{k+1} = F_i(X_1^k, \ldots, X_M^k) \qquad where \;\; i = 1, \ldots, M \tag{1.7}$$

Where X_i^k is the sub-vector executed over the i^{th} computing unit at the iteration k.

Algorithm 2 The synchronous parallel iterative algorithm

1: Initialize the sub-vectors $(X_1^0, \ldots, X_M^0)$
2: **for** $k := 1$ step 1 to *convergence* **do**
3: **parfor** $i := 1$ to M **do**
4: $X^{(k+1)} = F(X^k)$
5: **end parfor**
6: **end for**

The algorithm 2 represents the synchronous parallel iterative algorithm. Similarly to the sequential iterative algorithm 2, this algorithm stops iterating when the convergence condition is satisfied. We consider that the keyword **parfor** is used to make a for loop in parallel.

This algorithm needs to satisfy a convergence condition which is called the global convergence condition. In order to detect the global convergence overall computing units, first we need to compute at each iteration the local residual. Then at the end of each iteration, all the local residuals from M computing units must be reduced to one maximum value represented by the global residual. For example, in MPI this operation is directly applied using a high level communication procedure called *AllReduce*. The goal of this communication procedure is to apply the reduction operation on all local residuals computed by the computing units.

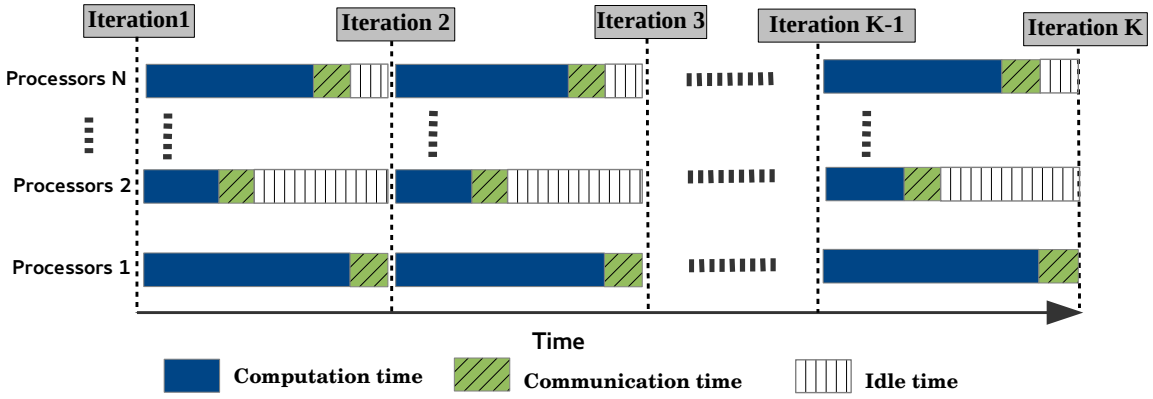

Figure 1.14: The SISC Model

In a synchronous parallel iterative algorithm, computing processors need to communicate with each others to exchange data at each iteration if there is a dependency between the parallel tasks. Algorithm 2 use synchronous iterations and synchronous communications denoted as **SISC** model. At each iteration, the computing processor waits until it receives all the computed data at the previous iteration from other processors to perform the next iteration. Figure 1.14, shows that using SISC model in a heterogeneous platform may result in big periods of the idle times represented by the white dashed spaces between two successive iterations. Indeed, this happens when the fast computing processors wait for the slower ones to finish their iterations to be able to synchronously send their data to them. Using this operation, faster processors waste a big amount of their computing power and thus consume uselessly energy. The increase in the heterogeneity in the computing powers between the processors may increase proportionally these idle times. Accordingly, this algorithm can be effectively run over a local cluster, where a high speed local network is used to reduce these idle times.

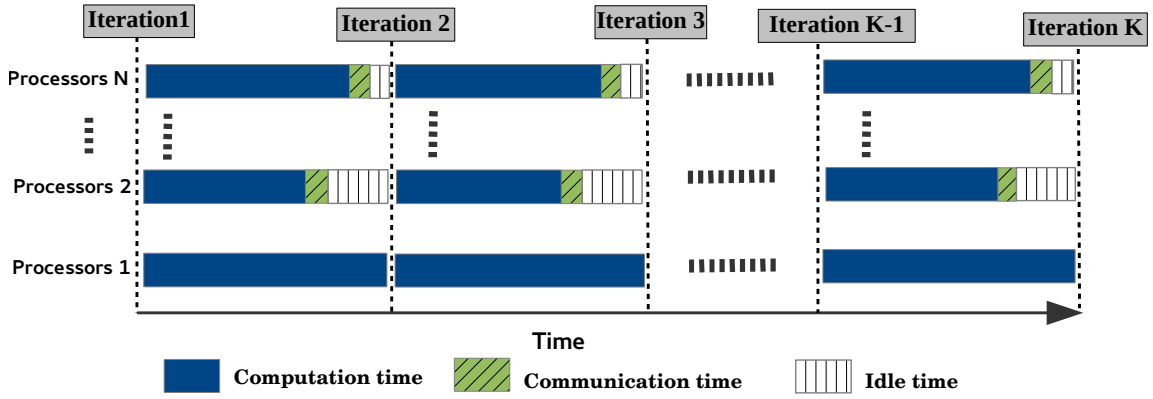

Figure 1.15: The SIAC Model

Furthermore, the communications of the synchronous iterative algorithm can be replaced by asynchronous ones. The resulting algorithm is called Synchronous Iterations

with Asynchronous Communications and denoted as **SIAC** algorithm. The main principle of this algorithm is to use synchronize iterations while exchanging the data between the computing units asynchronously. Moreover, each computing unit does not need to wait for its neighbours to receive the data messages that it has sent, while it only waits to receive data from them. This can be implemented with SISC algorithm that is programmed in MPI by replacing the synchronous send of the messages by asynchronous ones, while keeping the synchronous receive. The only advantage of this technique is to reduce the idle times between iterations by allowing the communications to overlap partially with computations, see figure 1.15. The idle times are not totally eliminated because the fast computing nodes must wait for slow ones to send their data messages. SISC and SIAC algorithms are not tolerant to the loss of data messages. Consequently, if one node crashes, all the other computing nodes are blocked.

1.3.2/ ASYNCHRONOUS PARALLEL ITERATIVE METHOD

The asynchronous iterations mean that all processors perform their iterations without considering the works of other processors. Each processor does not have to wait to receive data messages from other processors and continues to compute the next iteration using the last data received from neighbours. Therefore, there are no idle times at all between the iterations as in Figure 1.16. This figure indicates that fast processors can perform more iterations than the slower ones at the same time. The asynchronous iterative algorithm that uses an asynchronous communications is called **AIAC** algorithm. Similarly to the SISC algorithm, the AIAC algorithm subdivides the global vectors X into M subvectors between the computing units. The main difference between the two algorithms is that these M sub-vectors are not updated at each iteration in the AIAC algorithm because both iterations and communications are asynchronous.

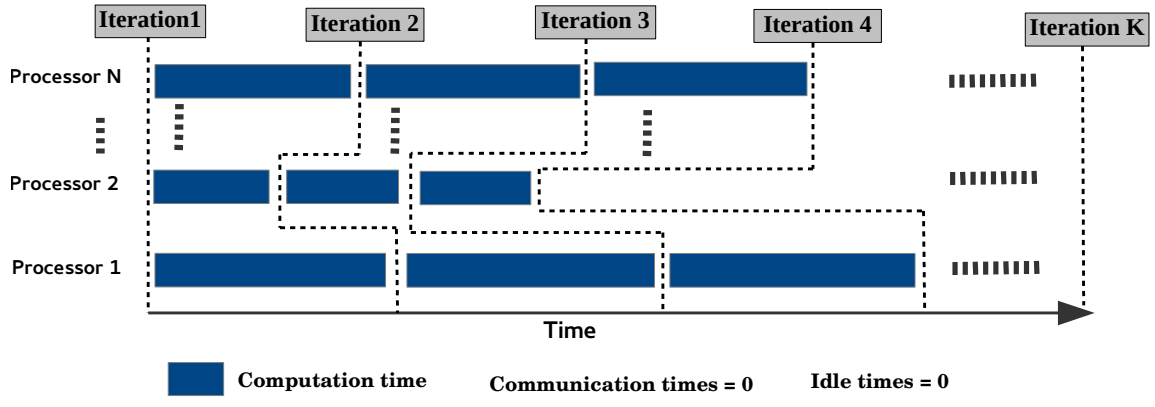

Figure 1.16: The AIAC Model

The global convergence detection of the asynchronous parallel iterative is not trivial. For more information about the convergence detection techniques of the asynchronous iterative methods, refer to [32, 15, 35, 36] for more details.

The implementation of the AIAC method is not easy, but it gives many advantages over the traditional synchronous iterative method:

- It prevents the existence of idle times, since each processor does not have to wait to receive the data messages from its neighbours to compute the next iteration.
- Less sensitive for the heterogeneous communications and nodes' computing powers. In heterogeneous platform, the fast nodes do not need to wait for the slow ones, and they can perform more iterations compared to them. While in the traditional synchronous iterative methods, the fast computing nodes perform the same number of iterations as the slow ones because they are blocked.
- The loss of data messages is totally tolerant because each computing unit is not blocked waiting for the message. If the message is lost, the destination node does not have to wait for this data message and it uses the last received data to perform its iteration independently.
- In the grid architecture, the local clusters from different sites are connected via a slow network with a high latency. The use of the AIAC model reduces the delay of sending the data message over such slow network link and thus the performance of the applications is not affected.

In addition to the difficulty of applying the asynchronous iterative model, it has some disadvantages that can be summarized by these points:

- It is not compatible with all types of the iterative applications because some of these applications need to receive data messages at each iteration or they would not converge.
- An asynchronous iterative method requires more iterations compared to the synchronous one to converge. The increase in the number of iterations may increase proportionally the execution time of the application if it is being executed on a fast homogeneous cluster.
- Since each node does not receive new data messages at each iteration, detecting the global convergence is harder than for the synchronous model. Therefore, in AIAC algorithm a process can perform many iterations without receiving any data messages from its neighbours. The absence of receiving new data messages makes the data component invariant at the computing units and thus it provides a false local convergence. At the reception of the first data message, the local subsystem will diverge after computing the next iteration. Therefore, special mechanisms are required for detecting the global convergence of a parallel iterative algorithm implemented according to the asynchronous iteration model.

In work of this thesis, we are interested in optimizing the energy consumption of parallel iterative methods running over clusters or grids.

1.4/ THE ENERGY CONSUMPTION MODEL OF A PARALLEL APPLICATION

Many researchers [88, 66, 52, 69] divide the power consumed by a processor into two power metrics: static power and dynamic power. The first one is consumed as long as the

computing unit is on, the latter is only consumed during computation times. The dynamic power P_{dyn} is related to the switching activity α, load capacitance C_L, the supply voltage V and operational frequency F, as shown in EQ (1.8).

$$P_{dyn} = \alpha \cdot C_L \cdot V^2 \cdot F \tag{1.8}$$

The static power P_{static} captures the leakage power as follows:

$$P_{static} = V \cdot N_{trans} \cdot K_{design} \cdot I_{leak} \tag{1.9}$$

Where V is the supply voltage, N_{trans} is the number of transistors, K_{design} is a design dependent parameter and I_{leak} is a technology-dependent parameter.

The dynamic voltage and frequency scaling technique (**DVFS**) is a process that is allowed in modern processors to reduce the dynamic power by scaling down the voltage and frequency of the CPU. Its main objective is to reduce the overall energy consumption of the CPU [47]. The operational frequency F depends linearly on the supply voltage V as follows:

$$V = \beta \cdot F \tag{1.10}$$

Where β is some of constant. This equation is used to study the change of the dynamic voltage with respect to various frequency values in [66]. The reduction process of the frequency can be expressed by the scaling factor S which is the ratio between the maximum and the new frequency as in EQ (1.11).

$$S = \frac{F_{max}}{F_{new}} \tag{1.11}$$

The value of the scaling factor S is greater than 1 when changing the frequency of the CPU to any new frequency value (P-state) in the governor. The CPU governor is an interface driver supplied by the operating system's kernel to lower a core's frequency [1].

Depending on the equation 1.11, the new frequency F_{new} can be calculated as follows:

$$F_{new} = S^{-1} \cdot F_{max} \tag{1.12}$$

Replacing V in 1.8 as in 1.10 gives the following equation of the dynamic power consumption as a function of the constant β instead of V:

$$P_{dyn} = \alpha \cdot C_L \cdot (\beta \cdot F)^2 \cdot F = \alpha \cdot C_L \cdot \beta^2 \cdot F^3 \tag{1.13}$$

Replacing F_{new} in 1.13 as in 1.12 gives the following equation for dynamic power consumption:

$$\begin{aligned} P_{dynNew} = \alpha \cdot C_L \cdot \beta^2 \cdot F_{new}^3 = \alpha \cdot C_L \cdot \beta^2 \cdot F_{max}^3 \cdot S^{-3} = \alpha \cdot C_L \cdot (\beta \cdot F_{max})^2 \cdot F_{max} \cdot S^{-3} \\ = \alpha \cdot C_L \cdot V^2 \cdot F_{max} \cdot S^{-3} = P_{dyn} \cdot S^{-3} \end{aligned} \tag{1.14}$$

Where P_{dynNew} and P_{dyn} are the dynamic powers consumed with the new frequency and the maximum frequency respectively.

According to (1.14) the dynamic power is reduced by a factor of S^{-3} when reducing the frequency of a processor by a factor of S. The energy consumption is measured in Joule, and can be calculated by multiplying the power consumption, measured in watts, by the execution time of the program as follows:

$$Energy = Power \cdot T \tag{1.15}$$

According to the equation 1.15, the dynamic energy consumption of the program executed in the time T over one processor is the dynamic power multiplied by the execution time. Moreover, the frequency scaling factor S increases the execution time of the processor linearly, then the new dynamic energy consumption can be computed as follows:

$$E_{dynNew} = P_{dyn} \cdot S^{-3} \cdot (T \cdot S) = S^{-2} \cdot P_{dyn} \cdot T \tag{1.16}$$

According to [88, 66], the static power consumption P_{static} does not changed when the frequency of the processor is scaled down. Therefore, the static energy consumption can be computed as follows:

$$E_{static} = S \cdot P_{static} \cdot T \tag{1.17}$$

Therefore, the energy consumption of an individual task running over one processor is the sum of both static and dynamic energies that can be computed as follows:

$$E_{ind} = E_{dynNew} + E_{static} = S^{-2} \cdot P_{dyn} \cdot T + S \cdot P_{static} \cdot T \tag{1.18}$$

The total energy consumption of N parallel task running on N processors is the summation of the individual energies consumed by all processors. This model is developed and used by Rauber and Rünger [66]. The total energy consumed by the parallel tasks running on a homogeneous platform is computed by sorting the execution time of the all parallel tasks in a descending order, then using EQ (1.19).

$$E_{\text{ all tasks}} = P_{dyn} \cdot S_1^{-2} \cdot \left(T_1 + \sum_{i=2}^{N} \frac{T_i^3}{T_1^2} \right) + P_{static} \cdot T_1 \cdot S_1 \cdot N \tag{1.19}$$

Where N is the number of parallel tasks, T_i for $i = 1, \ldots, N$ are the execution times of the sorted tasks. Therefore, $T1$ is the time of the slowest task, and S_1 its scaling factor which should be the highest because they are proportional to the time values T_i. Finally, model 1.19 can be used to measure the energy consumed by any parallel application such as the iterative parallel applications with respect to the new scaled frequency value.

There are two drawbacks in this energy model as follows:

- The message passing iterative program consists of communication and computation times. This energy model assumes that the dynamic power is consumed during both these times. While the processor during the communication times remains idle and only consumes the static power, for more details see [34].
- It is not well adapted to a heterogeneous architecture when there are different types of processors, which consume different dynamic and static powers.

Therefore, one of the most important goals of this work is to develop a new energy models that take into consideration the communication times in addition to the computation times in order to modelize and measure the energy consumptions of the parallel iterative methods. These models must be suitable to homogeneous or heterogeneous parallel architectures.

1.5/ CONCLUSION

In this chapter, three sections have been presented to describe the parallel hardware architectures, the parallel iterative applications and the energy consumption model used to measure the energy consumption of these applications. The different types of parallelism levels that can be implemented in software and hardware techniques have been explained in the first section. Afterwards, different types of parallel architectures have been discussed and classified according to the connection between the computation units and the memory model. Both shared and distributed platforms as well as their depending parallel programming models have been categorized. In the second section, the two types of parallel iterative methods: synchronous and asynchronous ones were presented. The synchronous iterative methods are well adapted to local homogeneous clusters with a high speed network link, while the asynchronous iterative methods are more suited to the distributed heterogeneous clusters. Finally, in the third section, an energy consumption model proposed in the state of the art to measure the energy consumption of parallel applications was explained. This model cannot be used for all types of parallel architectures. Since, it assumes that the dynamic power is consumed during both of the communication and computation times, while the processor involved remains idle during the communication times and only consumes the static power. Moreover, it is not well adapted to heterogeneous architectures when there are different types of processors, that consume different dynamic and static powers.

For these reasons, in the next chapters of this thesis new energy consumption models are developed to efficiently predict the energy consumed by parallel iterative methods running on both homogeneous and heterogeneous architectures. Additionally, these energy models are used in a method that optimizes both energy consumption and performance of an iterative message passing application.

II

CONTRIBUTIONS

2

ENERGY OPTIMIZATION OF HOMOGENEOUS PLATFORM

2.1/ INTRODUCTION

Dynamic Voltage and Frequency Scaling (DVFS) can be applied to modern CPUs. This technique is usually used to reduce the energy consumed by a CPU while computing. Indeed, power consumption by a processor is exponentially related to its frequency. Thus, decreasing the frequency reduces the power consumed by the CPU. However, it can also significantly affect the performance of the executed program if it is compute bound. The performance degradation ratio can even be higher than the saved energy ratio. Therefore, the chosen frequency scaling factor must give the best possible trade-off between energy reduction and performance. This chapter presents an algorithm that predicts the energy consumed with each frequency gear and selects the one that gives the best ratio between energy consumption reduction and performance. Furthermore, the main objective of HPC systems is to execute as fast as possible the application. Therefore, our algorithm selects the scaling factor online with a very small overhead. The proposed algorithm takes into account both the computation and communication times of the Message Passing Interface (MPI) programs to choose the frequency scaling factor. This algorithm has the ability to predict both energy consumption and execution time over all available scaling factors. The prediction achieved depends on some computing time information, gathered at the beginning of the runtime. We have applied this algorithm to the NAS parallel benchmarks (NPB v3.3) developed by the NASA [57]. Our experiments are executed using the simulator SimGrid/SMPI v3.10 [18] over an homogeneous distributed memory architecture.

This chapter is composed of two parts. In the first part, the proposed frequency scaling selection algorithm uses the energy model of Rauber and Rünger [66] and is compared to Rauber and Rünger's method. The comparison results show that our algorithm gives better energy-time trade-off. In the second part, a new energy model that takes into account both the communication and computation times of the MPI programs running over a homogeneous cluster is developed. It also shows the new results obtained using the new energy model. The results are compared to the ones given by Rauber and Rünger's energy model.

This chapter is organized as follows: Section 2.3 explains the execution of parallel tasks and the sources of slack times. It also presents an energy model for homogeneous platforms from other researchers. Section 2.4 describes how the performance of MPI programs can be predicted. Section 2.5 presents the energy-performance objective function

that maximizes the reduction of energy consumption while minimizing the degradation of the program's performance. Section 2.6 details the algorithm that returns the scaling factor that gives the best energy-performance trade-off for a parallel application with iterations Section 2.7 verifies the accuracy of the performance prediction model and presents the results of the proposed algorithm. It also shows the comparison results between our method and other existing methods. Section 2.8 describes the new proposed energy consumption model for homogeneous platforms. Section 2.9 presents the experimental results of using the new energy model. Finally, section 2.10 summarizes this chapter.

2.2/ Related works

In this section, some heuristics to compute the scaling factor are presented and classified into two categories: offline and online methods.

2.2.1/ Offline scaling factor selection methods

The offline scaling factor selection methods are executed before the runtime of the program. They return static scaling factor values to the processors participating in the execution of the parallel program. On the one hand, the scaling factor values could be computed based on information retrieved by analyzing the code of the program and the computing system that will execute it. In [11], Azevedo et al. detect during compilation the dependency points between tasks in a multi-task program. This information is then used to lower the frequency of some processors in order to eliminate slack times. A slack time is the period of time during which a processor that has already finished its computation, has to wait for a set of processors to finish their computations and send their results to the waiting processor in order to continue its task that is dependent on the results of computations being executed on other processors. Freeh et al. showed in [34] that the communication times of MPI programs do not change when the frequency is scaled down. On the other hand, some offline scaling factor selection methods use the information gathered from previous full or partial executions of the program. The whole program or a part of it is usually executed over all the available frequency gears and the execution time and the energy consumed with each frequency gear are measured. Then a heuristic or an exact method uses the retrieved information to compute the values of the scaling factor for the processors. In [83], Xie et al. use an exact exponential breadth-first search algorithm to compute the scaling factor values that give the optimal energy reduction while respecting a deadline for a sequential program. They also present a linear heuristic that approximates the optimal solution. In [71], Rountree et al. use a linear programming algorithm, while in [25, 24], Cochran et al. use a multi-logistic regression algorithm for the same goal. The main drawback of these methods is that they all require executing the whole program or, a part of it, on all frequency gears for each new instance of the same program.

2.2.2/ Online scaling factor selection methods

The online scaling factor selection methods are executed during the runtime of the program. They are usually integrated into iterative programs where the same block of in-

structions is executed many times. During the first few iterations, a lot of information are measured such as the execution time, the energy consumed using a multimeter, the slack times, ... Then a method will exploit these measurements to compute the scaling factor values for each processor. This operation, measurements and computing new scaling factors, can be repeated as much as needed if the iterations are not regular. Kimura, Peraza, Yu-Liang et al. [45, 63, 22] used many heuristics to select the appropriate scaling factor values to eliminate the slack times during runtime. However, as seen in [30, 74], machine learning methods can take a lot of time to converge when the number of available gears is big. To reduce the impact of slack times, in [49], Lim et al. developed an algorithm that detects the communication sections and changes the frequency during these sections only. This approach might change the frequency of each processor many times per iteration if an iteration contains more than one communication section. In [34], Rauber and Rünger used an analytical model that can predict the consumed energy and the execution time for every frequency gear after measuring the consumed energy and the execution time with the highest frequency gear. These predictions may be used to choose the optimal gear for each processor executing the parallel program to reduce energy consumption. To maintain the performance of the parallel program , they set the processor with the biggest load to the highest gear and then compute the scaling factor values for the rest of the processors. Although this model was built for parallel architectures, it can be adapted to distributed architectures by taking into account the communication times. The primary contribution of this chapter is to present a new online scaling factor selection method which has the following characteristics:

1. It is based on both Rauber and Rünger and the new energy model to predict the energy consumption of the application with different frequency gears.
2. It selects the frequency scaling factor for simultaneously optimizing energy reduction and maintaining performance.
3. It is well adapted to distributed architectures because it takes into account the communication time.
4. It is well adapted to distributed applications with imbalanced tasks.
5. It has a very small footprint when compared to other methods (e.g., [74]) and does not require profiling or training as in [25, 24].

2.3/ EXECUTION TIME AND ENERGY CONSUMPTION OF PARALLEL TASKS RUNNING ON A HOMOGENEOUS PLATFORM

2.3.1/ PARALLEL TASKS EXECUTION ON A HOMOGENEOUS PLATFORM

A homogeneous cluster consists in identical nodes in terms of hardware and software. Each node has its own memory and at least one processor which can be a multi-core. The nodes are connected via a high bandwidth network. Tasks executed on this model can be either synchronous or asynchronous. In this chapter we consider the execution of synchronous tasks on distributed homogeneous platform. These tasks can synchronously exchange data via message passing.

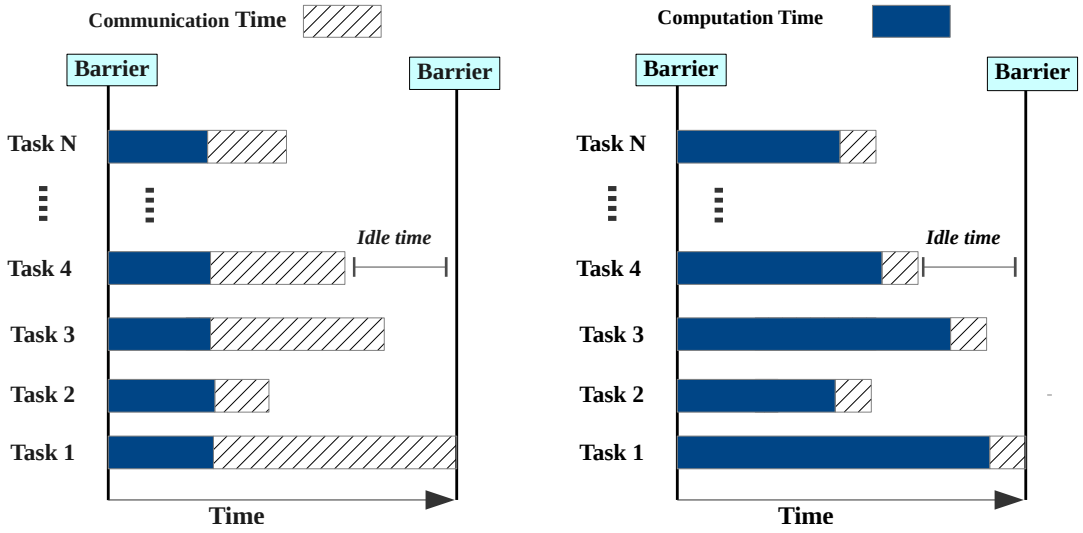

Figure 2.1: Parallel tasks execution on a homogeneous platform (a) imbalanced communications and (b) imbalanced computations

The execution time of a task consists in the computation time and the communication time. Moreover, the synchronous communications between tasks can lead to slack times while tasks wait at a synchronization barrier for other tasks to finish their tasks (see figure 2.1(a)). The imbalanced communications happen when nodes have to send/receive different amounts of data or they communicate with different numbers of nodes. Other sources of slack times are imbalanced computations. This happens when processing different amounts of data on each processor (see figure 2.1(b)). In this case the fastest tasks have to wait at the synchronization barrier for the slowest ones to continue their computations. In both cases the overall execution time of the program is the execution time of the slowest task as in EQ (2.1).

$$\textit{Program Time} = \max_{i=1,2,\ldots,N} T_i \tag{2.1}$$

where T_i is the execution time of task i and all the tasks are executed concurrently on different processors.

2.3.2/ ENERGY CONSUMPTION MODEL FOR A HOMOGENEOUS PLATFORM

The total energy for a parallel homogeneous platform, as presented by Rauber and Rünger [66], can be written as a function of the scaling factor S, as in EQ 1.19. Moreover, the scaling factor S_1 is the scaling factor which should be the highest because they are proportional to the time values T_i. Therefore, the scaling factors of the others tasks S_i are computed as in EQ 2.2.

$$S_i = S \cdot \frac{T_1}{T_i} = \frac{F_{max}}{F_{new}} \cdot \frac{T_1}{T_i},\ i = 1, 2, \ldots, N \tag{2.2}$$

Rauber and Rünger's scaling factor selection method uses the same energy model.

In their method, the optimal scaling factor is computed by minimizing the derivation of EQ (1.19) which produces EQ (2.3).

$$S_{opt} = \sqrt[3]{\frac{2}{N} \cdot \frac{P_{dyn}}{P_{static}} \cdot \left(1 + \sum_{i=2}^{N} \frac{T_i^3}{T_1^3}\right)} \tag{2.3}$$

This model computes the frequency scaling factor which minimizes the energy consumption of the parallel program.

2.4/ PERFORMANCE EVALUATION OF MPI PROGRAMS

The execution time of a parallel synchronous application with iteration is equal to the execution time of its slowest task as in figure (2.1). If there is no communication in the application and it is not data bounded, the execution time of this parallel application is linearly proportional to the operational frequency. Any DVFS operation for energy reduction increases the execution time of the parallel program. Therefore, the scaling factor S is linearly proportional to the execution time of the application. However, in most MPI applications the processes exchange data. During these communications the processors involved remain idle during a synchronous communication. For that reason, any change in the frequency has no impact on the time of communication [34]. The communication time for a task is the summation of periods of time that begin with an MPI call for sending or receiving a message until the message is synchronously sent or received. To be able to predict the execution time of MPI program, the communication time and the computation time for the slowest task must be measured before scaling. These times are used to predict the execution time for any MPI program as a function of the new scaling factor as in EQ (2.4).

$$T_{new} = T_{Max\ Comp\ Old} \cdot S + T_{Min\ Comm\ Old} \tag{2.4}$$

In this chapter, this prediction method is used to select the best scaling factor for each processor as presented in the next section.

2.5/ PERFORMANCE AND ENERGY REDUCTION TRADE-OFF

This section presents our method for choosing the scaling factor that gives the best trade-off between energy reduction and performance. This method takes into account the execution times for both computation and communication to compute the scaling factor. Since the energy consumption and the performance are not measured using the same metric, a normalized value of both measurements can be used to compare them. The normalized energy is the ratio between the consumed energy with scaled frequency and the consumed energy without scaled frequency:

$$E_{Norm} = \frac{E_{Reduced}}{E_{Original}} = \frac{P_{dyn} \cdot S_1^{-2} \cdot \left(T_1 + \sum_{i=2}^{N} \frac{T_i^3}{T_1^2}\right) + P_{static} \cdot T_1 \cdot S_1 \cdot N}{P_{dyn} \cdot \left(T_1 + \sum_{i=2}^{N} \frac{T_i^3}{T_1^2}\right) + P_{static} \cdot T_1 \cdot N} \tag{2.5}$$

In the same way we can normalize the performance as follows:

$$T_{Norm} = \frac{T_{New}}{T_{Old}} = \frac{T_{Max\ Comp\ Old} \cdot S + T_{Min\ Comm\ Old}}{T_{Max\ Comp\ Old} + T_{Min\ Comm\ Old}} \tag{2.6}$$

The relation between the execution time and the consumed energy of a program is nonlinear and complex. In consequences, the relation between the consumed energy and the scaling factor is also nonlinear, for more details refer to [34]. The resulting normalized energy consumption curve and execution time curve, for different scaling factors, do not have the same direction see Figure 2.2(b). To tackle this problem and optimize both terms, we inverse the equation of the normalized execution time which gives the normalized performance and is computed as follows:

$$P_{Norm} = \frac{T_{old}}{T_{new}} = \frac{T_{max\ Comp\ Old} + T_{min\ Comm\ Old}}{T_{max\ Comp\ Old} \cdot S + T_{min\ Comm\ Old}} \tag{2.7}$$

Then, we can model our objective function as finding the maximum distance between the energy curve EQ 2.5 and the performance curve EQ 2.7 over all available scaling factors. This represents the minimum energy consumption with minimum execution time (better performance) at the same time, see Figure 2.2(a). Then our objective function has the following form:

$$MaxDist = \max_{j=1,2,\ldots,F} (\overbrace{P_{norm}(S_j)}^{\text{Maximize}} - \overbrace{E_{norm}(S_j)}^{\text{Minimize}}) \tag{2.8}$$

where F is the number of available frequencies. Then we can select the optimal scaling factor that satisfies EQ 2.8. Our objective function can work with any energy model or static power values stored in a data file. Moreover, this function works in optimal way when the energy curve has a convex form over the available frequency scaling factors as shown in [52, 66, 74].

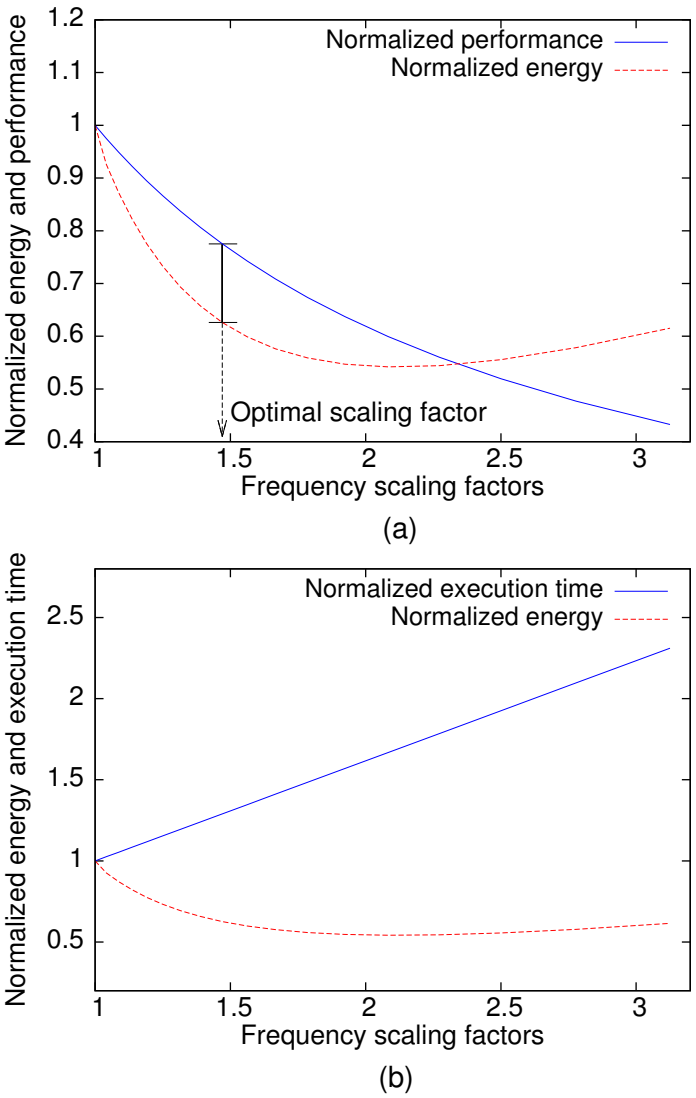

Figure 2.2: The energy and performance relation (a) Converted relation and (b) Real relation

2.6/ OPTIMAL SCALING FACTOR FOR PERFORMANCE AND ENERGY

Algorithm 3 computes the optimal scaling factor according to the objective function described above. The proposed algorithm works online during the execution time of the MPI program. It selects the optimal scaling factor after gathering the computation and communication times from the program after one iteration. Then the program changes the new frequencies of the CPUs according to the computed scaling factors. The experiments conducted over a homogeneous cluster and described in Section 2.7, showed that this algorithm has a small execution time. It takes on average 1.52 μs for 4 nodes and 6.65 μs for 32 nodes. The algorithm complexity is $O(F \cdot N)$, where F is the number of available frequencies and N is the number of computing nodes. The algorithm is called just once during the execution of the program. The DVFS algorithm 4 shows where and when the algorithm 3 is called in the MPI program.

After obtaining the optimal scaling factor, the program calculates the new frequency

Algorithm 3 Scaling factor selection algorithm for a homogeneous cluster

Initialize the variable $Dist = 0$
Set dynamic and static power values.
Set P_{states} to the number of available frequencies.
Set the variable F_{new} to max. frequency, $F_{new} = F_{max}$
Set the variable F_{diff} to the difference between two successive frequencies.
for $j := 1$ to P_{states} **do**
    $F_{new} = F_{new} - F_{diff}$
    $S = \frac{F_{max}}{F_{new}}$
    $S_i = S \cdot \frac{T_1}{T_i} = \frac{F_{max}}{F_{new}} \cdot \frac{T_1}{T_i}$ for $i = 1, \ldots, N$
    $E_{Norm} = \frac{P_{dyn} \cdot S_1^{-2} \cdot \left(T_1 + \sum_{i=2}^{N} \frac{T_i^3}{T_1^2} \right) + P_{static} \cdot T_1 \cdot S_1 \cdot N}{P_{dyn} \cdot \left(T_1 + \sum_{i=2}^{N} \frac{T_i^3}{T_1^2} \right) + P_{static} \cdot T_1 \cdot N}$
    $P_{Norm} = \frac{T_{old}}{T_{new}}$
    if $(P_{Norm} - E_{Norm} > Dist)$ **then**
        $S_{opt} = S$
        $Dist = P_{Norm} - E_{Norm}$
    end if
end for
Return S_{opt}

Algorithm 4 DVFS algorithm of homogeneous cluster

for $k := 1$ to *some iterations* **do**
    Computations section.
    Communications section.
    if $(k = 1)$ **then**
        Gather all times of computation and communication from each node.
        Call algorithm 3 with these times.
        Compute the new frequency from the returned optimal scaling factor.
        Set the new frequency to the CPU.
    end if
end for

F_i for each task proportionally to its execution time, T_i. By substitution of EQ (1.11) in EQ (2.2), we can calculate the new frequency F_i as follows:

$$F_i = \frac{F_{max} \cdot T_i}{S_{opt} \cdot T_{max}} \tag{2.9}$$

According to this equation all the nodes may have the same frequency value if they have balanced workloads, otherwise, they take different frequencies when having imbalanced workloads. Thus, EQ (2.9) adapts the frequency of the CPU to the nodes' workloads to maintain the performance of the program.

Table 2.1: Platform file parameters

Max Freq.	Min Freq.	Backbone Bandwidth	Backbone Latency	Link Bandwidth	Link Latency	Sharing Policy
2.5 GHz	800 MHz	2.25 GBps	0.5 μs	1 GBps	50 μs	Full Duplex

2.7/ EXPERIMENTAL RESULTS

Our experiments are executed on the simulator SimGrid/SMPI v3.10. We configure the simulator to use a homogeneous cluster with one core per node. The detailed characteristics of our platform file are shown in table (2.1). Each node in the cluster has 18 frequency values from 2.5 GHz to 800 MHz with 100 MHz difference between each two successive frequencies. The simulated network link is 1 GB Ethernet (TCP/IP). The backbone of the cluster simulates a high performance switch.

2.7.1/ PERFORMANCE PREDICTION VERIFICATION

In this section, the precision of the proposed performance prediction method based on EQ (2.4) is evaluated by applying it to the NAS benchmarks. The NAS programs are executed with the class B option to compare the real execution time with the predicted execution time. Each program runs offline with all available scaling factors on 8 or 9 nodes (depending on the benchmark) to produce real execution time values. These scaling factors are computed by dividing the maximum frequency by the new one see EQ (1.11). In our cluster there are 18 available frequency states for each processor. This leads to 18 run states for each program. Seven MPI programs of the NAS parallel benchmarks were used: CG, MG, EP, FT, BT, LU and SP. Table 2.2 shows the description of these seven benchmarks. Some of these benchmarks are considered MPI parallel applications with synchronous iterations or iterative applications that repeat the same block of instructions until convergence. However, the proposed method can be applied to any application that executes the same block of instructions many times and it is not limited to iterative methods. Figure (2.3) presents plots of the real execution times compared to the simulated ones. The maximum normalized error between these two execution times varies between 0.0073 to 0.031 depending on the executed benchmark. The smallest prediction error was for CG and the worst one was for LU.

2.7.2/ THE EXPERIMENTAL RESULTS FOR THE SCALING ALGORITHM

The proposed algorithm was applied to seven MPI programs of the NAS benchmarks (EP, CG, MG, FT, BT, LU and SP) which were run with three classes (A, B and C). For each instance, the benchmarks were executed on a number of processors proportional to the size of the class. Each class represents the problem size in ascending order from class A to C. The classes A, B and C were executed on 4, 8 or 9 and 16 nodes respectively. The energy consumption for all the NAS MPI programs was measured while assuming that the dynamic power with the highest frequency is equal to 20 W and the static power is equal to 4 W for all the experiments. These power values were also used by Rauber and Rünger in [66]. The results showed that the algorithm selected different scaling

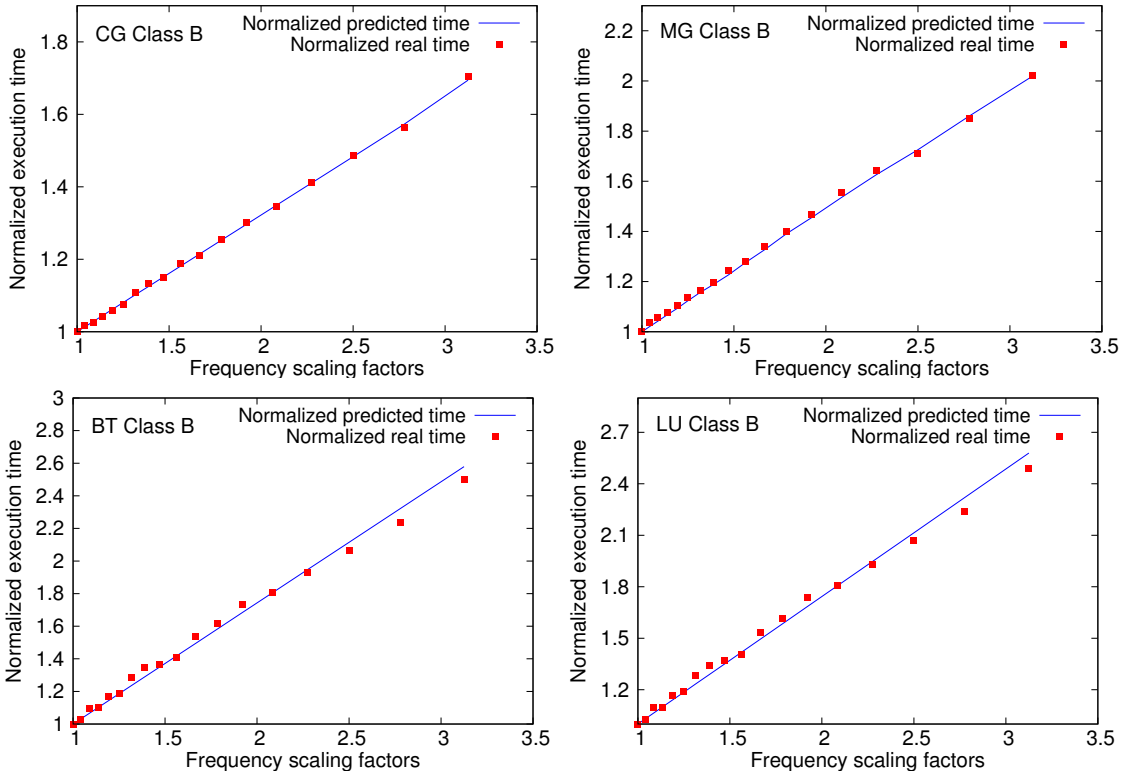

Figure 2.3: Comparing predicted to real execution time

Table 2.2: NAS Benchmarks description

Benchmark	Full Name	Description
CG	Conjugate Gradiant	It solves a system of linear equations by estimating the smallest eigenvalue of a large sparse matrix
MG	MultiGrid	It uses the multigrid method to approximate the solution of a three-dimensional discrete Poisson equation
EP	Embarrassingly Parallel	It applies the Marsaglia polar method to randomly generates independent Gaussian variates
FT	Fast Fourier Transform	It uses the fast Fourier transform to solve a three-dimensional partial differential equation
BT	Block Tridiagonal	They solve nonlinear partial differential equations
LU	Lower-Upper symmetric Gauss-Seidel	
SP	Scalar Pentadiagonal	

factors for each program depending on the communication features of the program as in the plots (2.4). These plots illustrate that there are different distances between the normalized energy and the normalized performance curves, because there are different communication features for each benchmark. When there are little or no communications, the performance curve is very close to the energy curve. Then the distance between the two curves is very small. This leads to small energy savings. The opposite happens when there are a lot of communication, the distance between the two curves is big. This leads to more energy savings (e.g. CG and FT), see table (2.3). All the discovered frequency

scaling factors optimize both the energy and the performance simultaneously for all the NAS benchmarks. In table (2.3), the optimal scaling factors results for each benchmark running class C are presented. These scaling factors give the maximum energy saving percentage and the minimum performance degradation percentage at the same time from all available scaling factors.

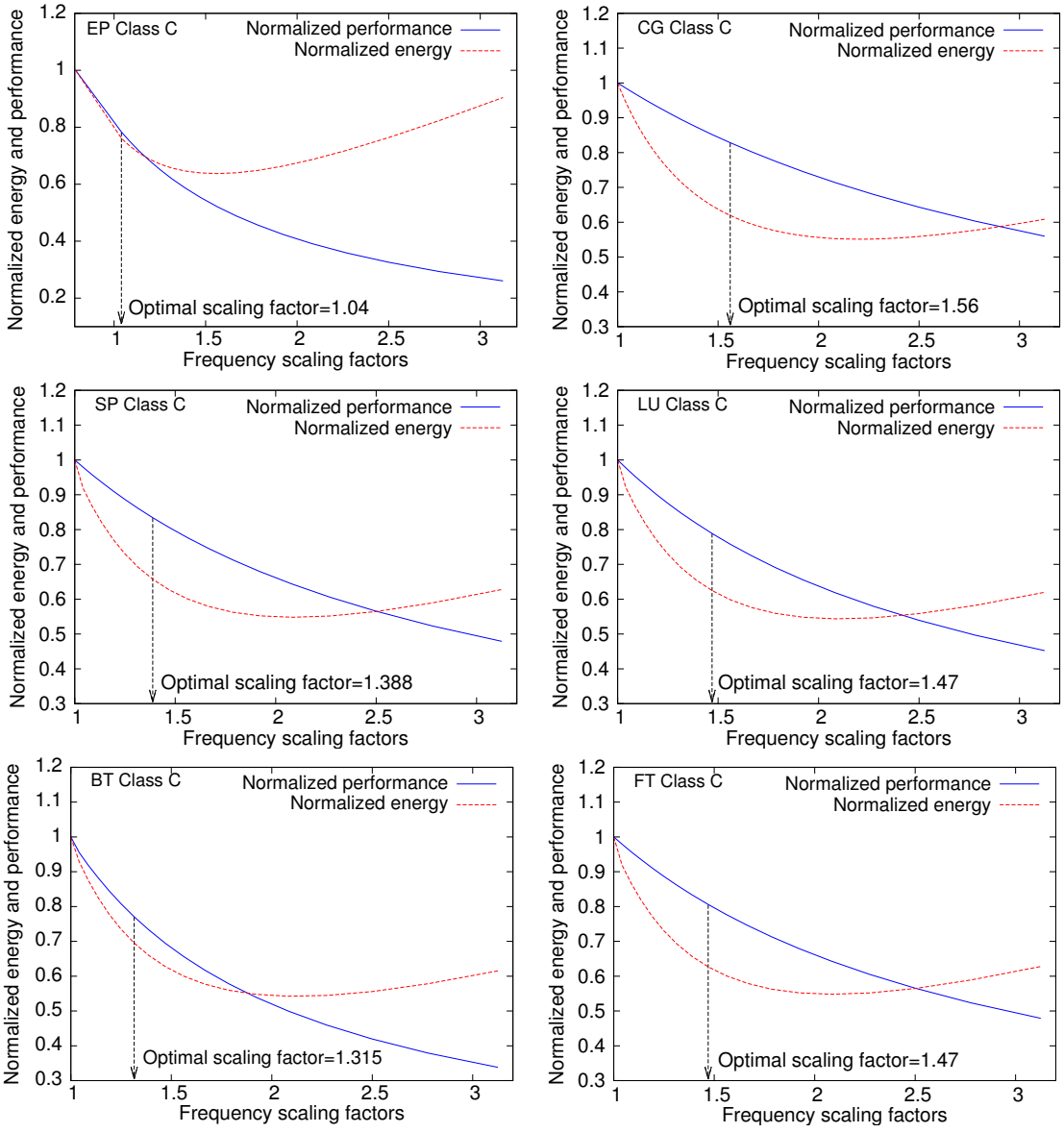

Figure 2.4: Optimal scaling factors for the predicted energy and performance of NAS benchmarks

As shown in table (2.3), when the optimal scaling factor has a big value we can gain more energy savings as in CG and FT benchmarks. The opposite happens when the optimal scaling factor has a small value as in BT and EP benchmarks. Our algorithm selects a big scaling factor value when the communication and other slacks times are big. In EP there are no communication inside the iterations, which leads our algorithm to select smaller scaling factors (inducing smaller energy savings).

Table 2.3: The scaling factors results

Program Name	Optimal Scaling Factor	Energy Saving %	Performance Degradation %	Energy-Perf. Distance
CG	1.56	39.23	14.88	24.35
MG	1.47	34.97	21.70	13.27
EP	1.04	22.14	20.73	1.41
LU	1.38	35.83	22.49	13.34
BT	1.31	29.60	21.28	8.32
SP	1.38	33.48	21.36	12.12
FT	1.47	34.72	19.00	15.72

2.7.3/ RESULTS COMPARISON

In this section, we compare our scaling factor selection method with the Rauber and Rünger's method [66]. They had two scenarios, the first is to reduce energy to the optimal level without considering the performance as in EQ (2.3). We refer to this scenario as R_E. The second scenario is similar to the first except setting the slower task to the maximum frequency (the scale $S = 1$) to keep the performance from degradation as mush as possible. We refer to this scenario as R_{E-P} and to our algorithm as EPSA (Energy to Performance Scaling Algorithm). The comparison is made in table 2.4. This table shows the results of our method and the Rauber and Rünger's scenarios for all the NAS benchmarks programs for class C.

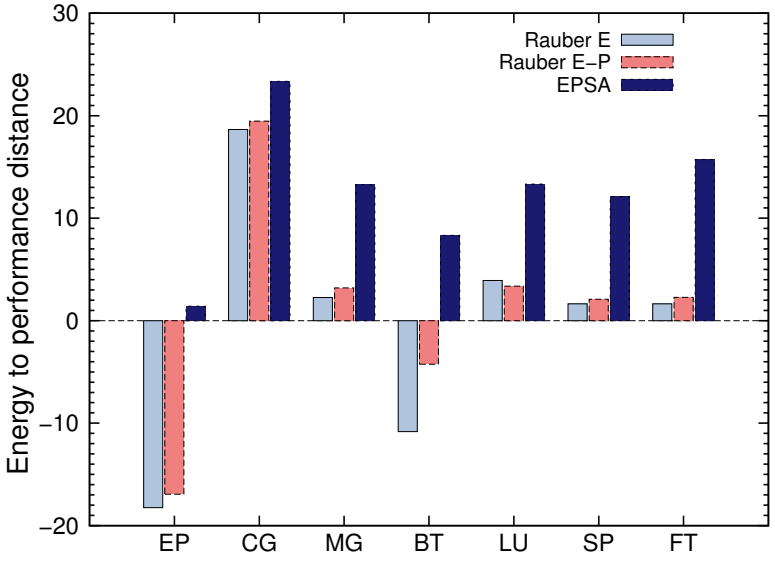

Figure 2.5: Comparing our method to Rauber and Rünger's methods

As shown in the table 2.4, the (R_{E-P}) method outperforms the (R_E) method in terms of performance and energy reduction. The (R_{E-P}) method also gives better energy savings than our method. However, although our scaling factor is not optimal for energy reduction, the results in these tables prove that our algorithm returns the best scaling factor that satisfy our objective method: the largest distance between energy reduction and performance degradation. Figure 2.5 illustrates even better the distance between the energy reduction and performance degradation. The negative values mean that one of

Table 2.4: Comparing results for the NAS class C

Method Name	Program Name	Factor Value	Energy Saving %	Performance Degradation %	Energy-Perf. Distance
$EPSA$	CG	1.56	39.23	14.88	24.35
R_{E-P}	CG	2.15	45.36	25.89	19.47
R_E	CG	2.15	45.36	26.70	18.66
$EPSA$	MG	1.47	34.97	21.69	13.27
R_{E-P}	MG	2.15	43.65	40.45	3.20
R_E	MG	2.15	43.64	41.38	2.26
$EPSA$	EP	1.04	22.14	20.73	1.41
R_{E-P}	EP	1.92	39.40	56.33	-16.93
R_E	EP	1.92	38.10	56.35	-18.25
$EPSA$	LU	1.38	35.83	22.49	13.34
R_{E-P}	LU	2.15	44.97	41.00	3.97
R_E	LU	2.15	44.97	41.80	3.17
$EPSA$	BT	1.31	29.60	21.28	8.32
R_{E-P}	BT	2.13	45.60	49.84	-4.24
R_E	BT	2.13	44.90	55.16	-10.26
$EPSA$	SP	1.38	33.48	21.35	12.12
R_{E-P}	SP	2.10	45.69	43.60	2.09
R_E	SP	2.10	45.75	44.10	1.65
$EPSA$	FT	1.47	34.72	19.00	15.72
R_{E-P}	FT	2.04	39.40	37.10	2.30
R_E	FT	2.04	39.35	37.70	1.65

the two objectives (energy or performance) has been degraded more than the other. The positive trade-offs with the highest values lead to maximum energy savings while keeping the performance degradation as low as possible. Our algorithm always gives the highest positive energy to performance trade-offs while the Rauber and Rünger's method, (R_{E-P}), gives sometimes negative trade-offs such as for BT and EP.

2.8/ THE NEW ENERGY MODEL FOR A HOMOGENEOUS CLUSTER

As mentioned in chapter 1 section 1.3, the power consumed by a processor is divided into two power metrics: the static and the dynamic power. The first power metric is consumed as long as the computing unit is on, while the other one is consumed when the processor is doing the computations. Consequentially, the energy consumed by an individual processor to execute a given program can be computed as follows:

$$E_{ind} = P_{dyn} \cdot T_{Comp} + P_{static} \cdot T \tag{2.10}$$

where T is the execution time of the program, T_{Comp} is the computation time and $T_{Comp} \leq T$. T_{Comp} may be equal to T if there is no communication, no slack time and no synchronization.

Applying a DVFS operation leads to a new frequency state which is represented by

the frequency scaling factor S, computed as in the equation 1.11. According to Rauber and Rünger's energy model 1.19, the dynamic energy is consumed during the overall program's execution time. This assumption is not precise because the CPU only consumes the dynamic power during computation time. Moreover, the CPU involved remains idle during the communication times and only consumes the static power, see [34]. We have also conducted some experiments over a real homogeneous cluster where some MPI programs of the NAS benchmarks were executed while varying the CPUs frequencies at each execution. The results prove that changing the frequency does not effect on the communication times of these programs. Therefore, the frequency scaling factor S can increase the computation times proportionally to its value, and does not effect the communication times. This assumption consort with the used performance prediction model 2.4. This model is evaluated and its prediction accuracy is showed in section 2.7.1. Therefore, the new dynamic energy is the dynamic power multiplied by the new time of computation and is given by the following equation:

$$Ed_{New} = Pd_{Old} \cdot S^{-3} \cdot (T_{comp} \cdot S) = S^{-2} \cdot Pd_{Old} \cdot T_{comp} \tag{2.11}$$

The static power is related to the power leakage of the CPU and is consumed during computation and even when idle. As in [66, 88], the static power of a processor is considered as constant during idle and computation periods, and for all its available frequencies. The static energy is the static power multiplied by the execution time of the program. According to the execution time model in (2.4), the execution time of the program is the sum of the computation and the communication times. The computation time is linearly related to the frequency scaling factor, while this scaling factor does not affect the communication time. Then, the static energy of a processor after scaling its frequency is computed as follows:

$$Es = P_{static} \cdot (T_{comp} \cdot S + T_{comm}) \tag{2.12}$$

In particular, in a homogeneous cluster all the computing nodes have the same specification and thus their CPUs have similar frequencies gears. The execution time of the MPI application is the execution time of the slowest task as shown in section 2.3.1. Therefore, the frequency scaling factor S of the slowest task can be used to modelize the energy consumption of the parallel application. The dynamic energy consumed by N parallel tasks is the summation of all the dynamic energies of all tasks during the computation time T_{comp_i} of each task. The static energy of each task is the static power consumed during the execution time of the slower task because all the tasks are synchronised and have the same execution time. Therefore, the energy consumption model of N parallel task executed synchronously over a homogeneous platforms can be represented as in 2.13.

$$E_{new} = \sum_{i=1}^{N} (S^{-2} \cdot Pd \cdot T_{comp_i}) + (Ps \cdot (T_{\textit{Max Comp Old}} \cdot S + T_{\textit{Min Comm Old}})) \cdot N \tag{2.13}$$

According to this model, the frequency scaling factor S reduces the energy consumption of the homogeneous architecture by a factor of S^{-2} and increases the execution time by a factor of S. This model can be used to predict the energy consumption of the message passing applications with synchronous iterations after gathering the computation and communication times of the first iteration. Furthermore, it can be used to measure

the energy consumption of the parallel application with iterations by multiplying the energy consumed of all tasks in one iteration by the number of the iterations.

This model is used by the algorithm 3 to predict the energy consumption and to select the optimal frequency scaling factor. The new frequency F_i can be computed as in 2.9 while using the new selected frequency scaling factor. In the next section, algorithm 3 is re-evaluated while using this new energy model and the new results are presented.

2.9/ THE EXPERIMENTAL RESULTS USING THE NEW ENERGY MODEL

This section presents the results of applying the frequency selection algorithm 3 using the new proposed energy model 2.13 to NAS parallel benchmarks. The class C of the benchmarks was executed on a homogeneous architecture composed of 16 nodes and simulated by SimGrid. The same static and dynamic power values were used as in section 2.7.2. Figure 2.6 presents the energy consumption of the NAS benchmarks class C using the new energy model and the Rauber and Rünger's model. The energy consumptions of both models are computed using similar parameters: frequency scaling factors, dynamic and static powers values. As shown in this figure, the majority of the benchmarks consumes less energy using the new model than when using the Rauber and Rünger's model. Two reasons explain these differences in the energy consumptions: the first one is related to the dynamic power consumption, where the new energy model ensures that this power metric is only consumed during the computation time, while the other model assumes that the dynamic power is consumed during both computation and communication times and thus increasing the dynamic energy consumption. The second reason is related to the execution time. In the new model only the computation times are increased when the frequency of a processor is scaled down, while Rauber and Rünger's model indicates that both the computation and communication times are increased according to the scaling factor and hence more static energy is consumed. Therefore, the MPI programs that have big communication times, have bigger energy consumption values using Rauber and Rünger's model when compared to the new model as for the CG, SP, LU and FT benchmarks. Whereas, if the MPI programs have very small communication times, their computed energy values have very small differences using both models such as for the MG and BT benchmarks, or they are identical such as for the EP benchmark where there is no communication and no idle times.

Method Name	Rauber Energy Model Results			New Energy Model Results		
	Scaling Factors	Energy Saving%	Performance Degradation%	Scaling Factors	Energy Saving%	Performance Degradation%
CG	1.56	39.23	14.88	1.47	30.20	13.56
MG	1.47	34.97	21.69	1.38	30.04	16.48
EP	1.04	22.14	20.73	1.04	22.14	20.73
LU	1.38	35.83	22.49	1.31	29.15	18.03
BT	1.31	29.60	21.53	1.31	28.75	21.55
SP	1.38	33.48	21.35	1.31	28.93	14.83
FT	1.47	34.72	19.00	1.38	29.94	17.43

Table 2.5: The Results of NAS Parallel Benchmarks running on 16 nodes

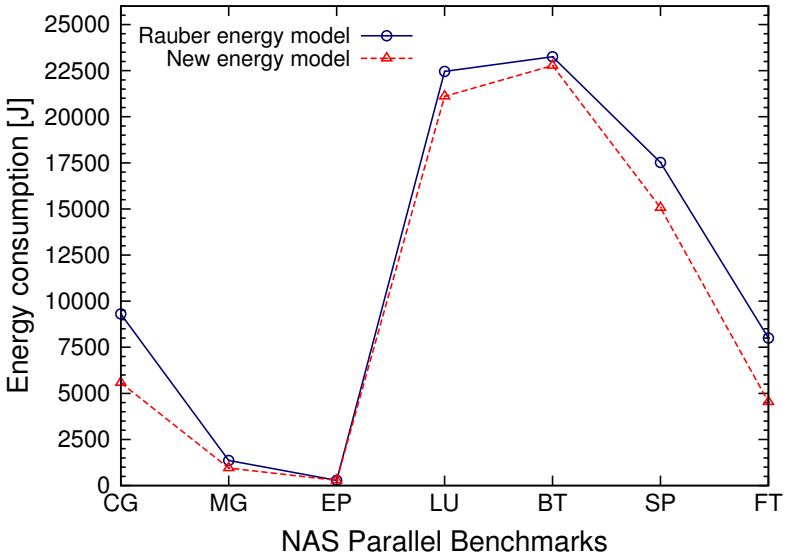

Figure 2.6: Comparing the energy consumptions estimated using Rauber energy model and our own

Table 2.5 shows the energy saving and performance degradation percentages when applying the frequency selecting algorithm using the new proposed energy model. It also presents the new selected frequency scaling factors and compares them to the ones used by the Rauber and Rünger's model. It shows that the new selected frequency scaling factors are smaller than those selected using the other model because the predicted energies by the new energy model are smaller. Consequently, less energy savings and performance degradation percentages are produced according to these smaller frequency scaling factors such as for the CG, MG, LU, SP and FT benchmarks. While in the BT and EP benchmarks where there are very small or no communication times, similar scaling factors are selected because the predicted energies by the two models are approximately equivalent.

Therefore, the new proposed energy model is more accurate than Rauber and Rünger's energy model, because it takes into consideration both the communication and idle times in addition to the computation times of message passing programs running over homogeneous clusters. The scaling factor selection algorithm can work with any energy model and it selects the scaling factor values according to the predicted energy values.

2.10/ CONCLUSION

In this chapter, a new online scaling factor selection method that optimizes simultaneously the energy and performance of a distributed application running on a homogeneous cluster have been presented . It uses the computation and communication times measured at the first iteration to predict the energy consumption and the performance of the parallel application at every available frequency. Then, it selects the scaling factor that gives the best trade-off between energy reduction and performance which is the maximum distance between the energy and the performance curves. To evaluate this method, we have applied it to the NAS benchmarks and it was compared to the Rauber and Rünger's method

while being executed on the SimGrid simulator. The results showed that our method, outperforms the Rauber and Rünger's method in terms of energy-performance ratio. Finally, this chapter presents a new energy consumption model for parallel applications with synchronous iterations running on homogeneous clusters. This model takes into consideration both the computation and communication times and their relation with the frequency scaling factor. The results obtained using the new energy model have shown that different frequency scaling factors were selected which gave new experimental results that are more accurate and realistic.

3

Energy Optimization of Heterogeneous Platforms

3.1/ Introduction

Computing platforms are consuming more and more energy due to the increasing number of nodes composing them. In a heterogeneous computing platform composed of multiple computing nodes, nodes may differ in the computing power from each others. Accordingly, the fast nodes have to wait for the slow ones to finish their works. The resulting waiting times are called idle times which are increased proportionally to the increase in the heterogeneity between the computing nodes. This leads to a big waste in the computing power and thus the energy consumed by fast nodes. To minimize the operating costs of these platforms many techniques have been used. Dynamic voltage and frequency scaling (DVFS) is one of them. It reduces the frequency of a CPU to lower its energy consumption. However, lowering the frequency of a CPU may increase the execution time of an application running on that processor. Therefore, the frequency that gives the best trade-off between the energy consumption and the performance of an application must be selected.

In this chapter, two new online frequency selecting algorithms for heterogeneous local clusters (heterogeneous CPUs) and grid platforms are presented. They select the frequencies that try to give the best trade-off between energy saving and performance degradation, for each node computing the synchronous message passing application with iterations. These algorithms have a small overhead and work without training or profiling. They use new energy models for message passing synchronous applications with iterations running on both the heterogeneous local cluster and the grid platform. The first proposed algorithm for a heterogeneous local cluster was evaluated on the SimGrid simulator while running the class C of the NAS parallel benchmarks. The experiments conducted over 8 heterogeneous nodes show that it reduces on average the energy consumption by 29.8% while limiting the performance degradation to 3.8%. The second proposed algorithm for a grid platform was evaluated on the Grid5000 testbed platform while running the class D of the NAS parallel benchmarks. The experiments were run on 16 nodes, distributed on three clusters, and show that the algorithm reduces on average the energy consumption by 30% while the performance is on average only degraded by 3.2%. Finally, both algorithms were compared to the EDP method. The comparison results show that they outperform the latter in the energy reduction and performance trade-off.

This chapter is organized as follows: Section 3.2 presents some related works from other authors. Section 3.3 presents the performance and energy models of synchronous message passing programs running over a heterogeneous local cluster. It also describes the proposed frequency selecting algorithm then the precision of the proposed algorithm is verified. Section 3.4 presents the simulation results of applying the algorithm on the NAS parallel benchmarks class C and executing them on a heterogeneous local cluster. It shows the results of running three different power scenarios and comparing them. Moreover, it also shows the comparison results between the proposed method and an existing method. Section 3.5 shows the energy and performance models in addition to the frequencies selecting algorithm of synchronous message passing programs running over a grid platform. Section 3.6 presents the results of applying the algorithm on the NAS parallel benchmarks (class D) and executing them on the Grid'5000 testbed. The algorithm is also evaluated over multi-core architectures and over three different power scenarios. Moreover, Section 3.6, shows the comparison results between the proposed method and the EDP method. Finally, in Section 3.7 the chapter ends with a summary.

3.2/ RELATED WORKS

The process of selecting the appropriate frequency for a processor to satisfy some objectives, while taking into account all the constraints, is not a trivial operation. Many researchers used different strategies to tackle this problem. Some of them developed online methods that compute the new frequency while executing the application, such as [74, 75]. Others used offline methods that may need to run the application and profile it before selecting the new frequency, such as [71, 40]. The methods could be heuristics, exact or brute force methods that satisfy varied objectives such as energy reduction or performance. They also could be adapted to the execution's environment and the type of the application such as sequential, parallel or distributed architecture, homogeneous or heterogeneous platform, synchronous or asynchronous application, ...

In this chapter, we are interested in reducing the energy consumption when running a message passing synchronous applications with iterations over a heterogeneous platform. Some works have already been done for such platforms which can be classified into two types of heterogeneous platforms:

- the platform is composed of homogeneous GPUs and homogeneous CPUs.
- the platform is only composed of heterogeneous CPUs.

For the first type of platform, the computing intensive parallel tasks are executed on the GPUs and the rest are executed on the CPUs. Luley et al. [50], proposed a heterogeneous cluster composed of Intel Xeon CPUs and NVIDIA GPUs. Their main goal was to maximize the energy efficiency of the platform during computation by maximizing the number of FLOPS per watt generated. In [51], Kai Ma et al. developed a scheduling algorithm that distributes workloads proportional to the computing power of the nodes which could be a GPU or a CPU. All the tasks must be completed at the same time. In [37], Rong et al. showed that a heterogeneous (GPUs and CPUs) cluster that enables DVFS operations gave better energy and performance efficiency than other clusters only composed of CPUs.

The work presented in this chapter concerns the second type of platform, with heterogeneous CPUs. Many methods were conceived to reduce the energy consumption of this type of platform. Naveen et al. [55] developed a method that minimizes the value of $energy \times delay^2$ (the delay is the sum of slack times that happen during synchronous communications) by dynamically assigning new frequencies to the CPUs of the heterogeneous cluster. Lizhe et al. [81] proposed an algorithm that divides the executed tasks into two types: the critical and non critical tasks. The algorithm scales down the frequency of non critical tasks proportionally to their slack and communication times while limiting the performance degradation percentage to less than 10%. In [43], they developed a heterogeneous cluster composed of two types of Intel and AMD processors. They use a gradient method to predict the impact of DVFS operations on performance. In [73] and [48], the best frequencies for a specified heterogeneous cluster are selected offline using on heuristic. Chen et al. [21] used a greedy dynamic programming approach to minimize the power consumption of heterogeneous servers while respecting the given time constraint. This approach had considerable overhead. In contrast to the above described works, the work of this chapter presents the following contributions:

1. two new energy and two performance models for message passing synchronous applications with iterations running over a heterogeneous local cluster and a grid platform. All the models take into account the communications and the slack times. The models can predict the energy consumption and the execution time of the application.

2. two new online frequencies selecting algorithms for a heterogeneous local cluster and a grid platform. The algorithms have a very small overhead and do not need any training or profiling. They use a new optimization function which simultaneously maximizes the performance and minimizes the energy consumption of a message passing synchronous application with iterations.

3.3/ THE ENERGY OPTIMIZATION OF PARALLEL APPLICATIONS WITH ITERATIONS RUNNING OVER LOCAL HETEROGENEOUS CLUSTERS

3.3.1/ THE EXECUTION TIME OF MESSAGE PASSING DISTRIBUTED APPLICATIONS WITH ITERATIONS ON A HETEROGENEOUS LOCAL CLUSTER

In this section, we are interested in reducing the energy consumption of message passing distributed synchronous applications with iterations running over heterogeneous local clusters. In this work, a heterogeneous local cluster is defined as a collection of heterogeneous computing nodes interconnected via a high speed homogeneous network. Therefore, the nodes may have different characteristics such as computing power (FLOPS), energy consumption, CPU's frequency range, ... but they all have the same network bandwidth and latency.

The overall execution time of a distributed synchronous application with iterations over a heterogeneous local cluster consists of the sum of the computation time and the communication time for every iteration on a node. However, due to the heterogeneous computation power of the computing nodes, slack times may occur when fast nodes have

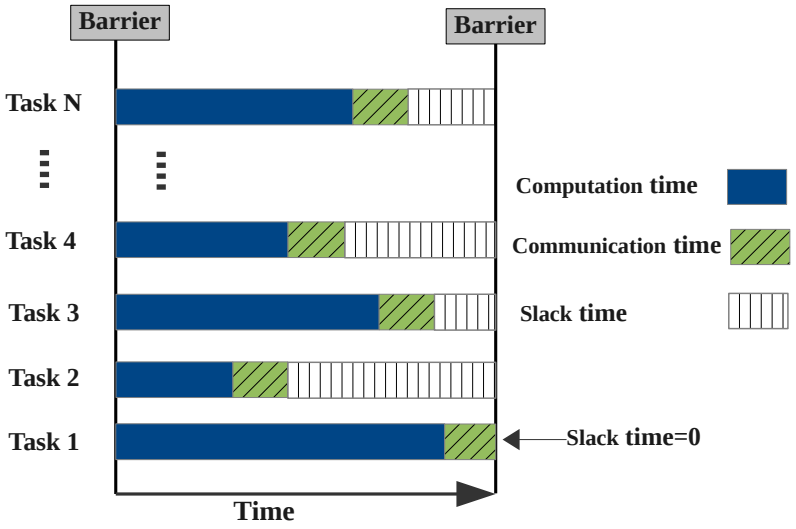

Figure 3.1: Parallel tasks on a heterogeneous platform

to wait, during synchronous communications, for the slower nodes to finish their computations (see Figure 3.1). Therefore, the overall execution time of the program is the execution time of the slowest task which has the highest computation time and no slack time.

Reducing the frequency of a processor by applying DVFS operation can be expressed by the scaling factor S which is the ratio between the maximum frequency and the new frequency of a CPU as in (1.11). The execution time of a compute bound sequential program is linearly proportional to the frequency scaling factor S. On the other hand, message passing distributed applications consist of two parts: computation and communication. The execution time of the computation part is linearly proportional to the frequency scaling factor S but the communication time is not affected by the scaling factor because the processors involved remain idle during the communications [34]. The communication time for a task is the summation of periods of time that begin with an MPI call for sending or receiving a message until the message is synchronously sent or received.

Since in a heterogeneous cluster the nodes may have different characteristics, especially different frequency gears, when applying DVFS operations on these nodes, they may get different scaling factors represented by a scaling vector: $(S_1, S_2, \ldots, S_N)$ where S_i is the scaling factor of processor i. To be able to predict the execution time of message passing synchronous applications with iterations running over a heterogeneous local cluster, for different vectors of scaling factors, the communication time and the computation time for all the tasks must be measured during the first iteration before applying any DVFS operation. Then the execution time for one iteration of the application with any vector of scaling factors can be predicted using (3.1).

$$T_{New} = \max_{i=1,2,\ldots,N}(T_{cpOld_i} \cdot S_i) + \min_{i=1,2,\ldots,N}(T_{cm_i}) \tag{3.1}$$

where T_{cpOld_i} is the computation time of processor i during the first iteration. The model computes the maximum computation time with scaling factor from each node added to the communication time of the slowest node. It means only the communication time without

any slack time is taken into account. Therefore, the execution time of the application with iterations is equal to the execution time of one iteration as in (3.1) multiplied by the number of iterations of that application.

This prediction model is improved from the model that predicts the execution time of message passing distributed applications for homogeneous architectures presented in Chapter 2 Section 2.4. The execution time prediction model is used in the method that optimizes both the energy consumption and the performance of parallel application with iterations, which is presented in the following sections.

3.3.2/ ENERGY MODEL FOR HETEROGENEOUS LOCAL CLUSTER

In Chapter 2, the dynamic and the static energy consumption of a processor is computed according to Equations 2.11 and 2.12 respectively. Then, the total energy consumption of a processor is the sum of these two metrics. Therefore, the overall energy consumption for the parallel tasks over a parallel cluster is the summation of the energies consumed by all the processors.

In the considered heterogeneous platform, each processor i may have different dynamic and static powers, noted as P_{d_i} and P_{s_i} respectively. Therefore, even if the distributed message passing application with iterations is load balanced, the computation time of each CPU i noted T_{cp_i} may be different and different frequency scaling factors may be computed in order to decrease the overall energy consumption of the application and reduce the slack times. The communication time of a processor i is noted as T_{cm_i} and could contain slack times when communicating with slower nodes, see Figure 3.1. Therefore, all the nodes do not have equal communication times. While the dynamic energy is computed according to the frequency scaling factor and the dynamic power of each node as in (2.11), the static energy is computed as the sum of the execution time of one iteration as in 3.1 multiplied by the static power of each processor. The overall energy consumption of a message passing distributed application executed over a heterogeneous cluster during one iteration is the summation of the dynamic and static energies for all the processors. It is computed as follows:

$$E = \sum_{i=1}^{N} (S_i^{-2} \cdot P_{d_i} \cdot T_{cp_i}) + \sum_{i=1}^{N} (P_{s_i} \cdot (\max_{i=1,2,\ldots,N} (T_{cp_i} \cdot S_i) + \min_{i=1,2,\ldots,N} (T_{cm_i}))) \tag{3.2}$$

Reducing the frequencies of the processors according to the vector of scaling factors $(S_1, S_2, \ldots, S_N)$ may degrade the performance of the application and thus, increase the consumed static energy because the execution time is increased [44]. The overall energy consumption for an application with iterations can be measured by measuring the energy consumption for one iteration as in (3.2) multiplied by the number of iterations of that application.

3.3.3/ OPTIMIZATION OF BOTH ENERGY CONSUMPTION AND PERFORMANCE

Using the lowest frequency for each processor does not necessarily give the most energy efficient execution of an application. Indeed, even though the dynamic power is reduced while scaling down the frequency of a processor, its computation power is proportionally decreased. Hence, the execution time might be drastically increased and during that

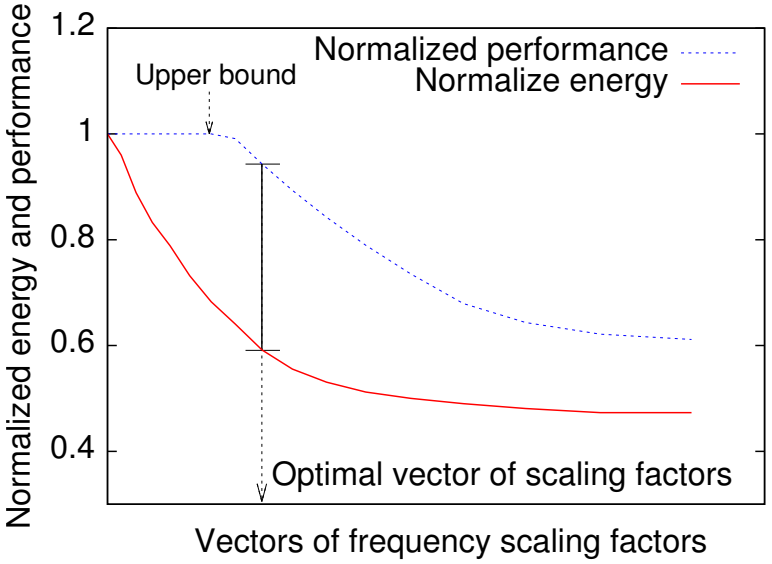

Figure 3.2: The energy and performance relation in heterogeneous cluster

time, dynamic and static powers are being consumed. Therefore, it might cancel any gains achieved by scaling down the frequency of all nodes to the minimum and the overall energy consumption of the application might not be the optimal one. It is not trivial to select the appropriate frequency scaling factor for each processor while considering the characteristics of each processor (computation power, range of frequencies, dynamic and static powers) and the task it is executing (computation/communication ratio). In Chapter 2, we proposed a method that selects the optimal frequency scaling factor for a homogeneous cluster executing a message passing synchronous application with iterations while giving the best trade-off between the energy consumption and the performance for such applications. In this section, this optimization method is improved while considering a heterogeneous clusters.

As described before, the relation between the energy consumption and the execution time for an application is complex and nonlinear. Thus, to find the trade-off relation between the energy consumption computed in Equation 3.2 and the performance with Equation 3.1 for the message passing applications with iterations, first we need to normalize both terms as follows:

$$E_{Norm} = \frac{E_{Reduced}}{E_{Original}} = \frac{\sum_{i=1}^{N} (S_i^{-2} \cdot P_{d_i} \cdot T_{cp_i}) + \sum_{i=1}^{N} (P_{s_i} \cdot T_{New})}{\sum_{i=1}^{N} (P_{d_i} \cdot T_{cp_i}) + \sum_{i=1}^{N} (P_{s_i} \cdot T_{Old})} \tag{3.3}$$

$$P_{Norm} = \frac{T_{Old}}{T_{New}} = \frac{\max_{i=1,2,...,N} (T_{cp_i} + T_{cm_i})}{\max_{i=1,2,...,N}(T_{cp_i} \cdot S_i) + \min_{i=1,2,...,N}(T_{cm_i})} \tag{3.4}$$

Then, the objective function can be modeled in order to find the maximum distance between the energy curve (3.3) and the performance curve (3.4) over all available sets of scaling factors for the processors of the heterogeneous cluster. This represents the minimum energy consumption with minimum execution time (maximum performance) at

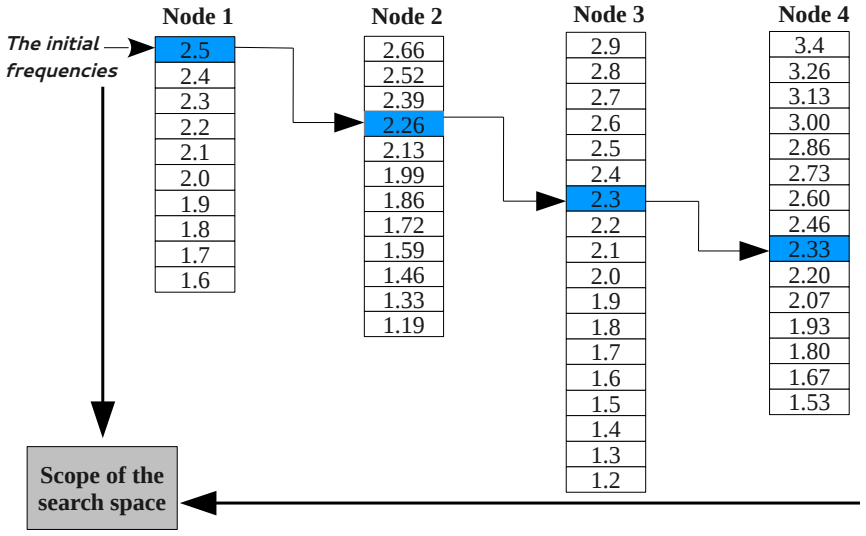

Figure 3.3: Selecting the initial frequencies in heterogeneous cluster

the same time, see Figure 3.2. Then the objective function has the following form:

$$MaxDist = \max_{\substack{i=1,\dots N \\ j=1,\dots,F_i}} (\overbrace{P_{Norm}(S_{ij})}^{\text{Maximize}} - \overbrace{E_{Norm}(S_{ij})}^{\text{Minimize}}) \tag{3.5}$$

where N is the number of nodes and F_i is the number of available frequencies for the node i. Then, the set of scaling factors that maximizes the objective function (3.5) should be selected.

3.3.4/ THE SCALING FACTORS SELECTION ALGORITHM FOR HETEROGENEOUS CLUSTER

In this section, Algorithm 5 is presented. It selects the frequency scaling factors vector that gives the best trade-off between minimizing the energy consumption and maximizing the performance of a message passing synchronous application with iterations executed on a heterogeneous local cluster. It works online during the execution time of the message passing program with iterations. It uses information gathered during the first iteration such as the computation time and the communication time in one iteration for each node. The algorithm is executed after the first iteration and returns a vector of optimal frequency scaling factors that satisfies the objective function (3.5). The program applies DVFS operations to change the frequencies of the CPUs according to the computed scaling factors. This algorithm is called just once during the execution of the program. Algorithm 6 shows where and when the proposed scaling algorithm is called in the MPI program with iterations.

The nodes in a heterogeneous cluster may have different computing powers. The algorithm takes into account this problem and tries to reduce these slack times when selecting the frequency scaling factors vector. At first, it selects initial frequency scaling factors that increase the execution times of fast nodes and minimize the differences between the computation times of the fast nodes and the slow ones. The value of the initial

Algorithm 5 Scaling factors selection algorithm for heterogeneous cluster

Require:

T_{cp_i} array of all computation times for all nodes during one iteration and with highest frequency.

T_{cm_i} array of all communication times for all nodes during one iteration and with highest frequency.

F_{max_i} array of the maximum frequencies for all nodes.

P_{d_i} array of the dynamic powers for all nodes.

P_{s_i} array of the static powers for all nodes.

F_{diff_i} array of the differences between two successive frequencies for all nodes.

Ensure: $S_{opt_1}, S_{opt_2} \ldots, S_{opt_N}$ is a vector of optimal scaling factors

1: $S_{cp_i} \leftarrow \frac{\max_{i=1,2,\ldots,N}(T_{cp_i})}{T_{cp_i}}$
2: $F_i \leftarrow \frac{F_{max_i}}{S_{cp_i}},\ i = 1, 2, \cdots, N$
3: Round the computed initial frequencies F_i to the closest one available in each node.
4: **if** (not the first frequency) **then**
5: $F_i \leftarrow F_i + F_{diff_i},\ i = 1, \ldots, N.$
6: **end if**
7: $T_{Old} \leftarrow \max_{i=1,\ldots,N}(T_{cp_i} + T_{cm_i})$
8: $E_{Original} \leftarrow \sum_{i=1}^{N} (P_{d_i} \cdot T_{cp_i} + P_{s_i} \cdot T_{Old})$
9: $S_{opt_i} \leftarrow 1,\ i = 1, \ldots, N.$
10: $Dist \leftarrow 0$
11: **while** (all nodes not reach their minimum frequency) **do**
12: **if** (not the last freq. **and** not the slowest node) **then**
13: $F_i \leftarrow F_i - F_{diff_i},\ i = 1, \ldots, N.$
14: $S_i \leftarrow \frac{F_{max_i}}{F_i},\ i = 1, \ldots, N.$
15: **end if**
16: $T_{New} \leftarrow \max_{i=1,2,\ldots,N}(T_{cpOld_i} \cdot S_i) + \min_{i=1,2,\ldots,N}(T_{cm_i})$
17: $E_{Reduced} \leftarrow \sum_{i=1}^{N} (S_i^{-2} \cdot P_{d_i} \cdot T_{cp_i}) + \sum_{i=1}^{N} (P_{s_i} \cdot (\max_{i=1,2,\ldots,N}(T_{cp_i} \cdot S_i) + \min_{i=1,2,\ldots,N}(T_{cm_i})))$
18: $P_{Norm} \leftarrow \frac{T_{Old}}{T_{New}}$
19: $E_{Norm} \leftarrow \frac{E_{Reduced}}{E_{Original}}$
20: **if** $(P_{Norm} - E_{Norm} > Dist)$ **then**
21: $S_{opt_i} \leftarrow S_i,\ i = 1, \ldots, N.$
22: $Dist \leftarrow P_{Norm} - E_{Norm}$
23: **end if**
24: **end while**
25: Return $S_{opt_1}, S_{opt_2}, \ldots, S_{opt_N}$

frequency scaling factor for each node is inversely proportional to its computation time that was gathered from the first iteration. These initial frequency scaling factors are computed as a ratio between the computation time of the slowest node and the computation time of the node i as follows:

$$S_{cp_i} = \frac{\max_{i=1,2,\ldots,N}(T_{cp_i})}{T_{cp_i}} \tag{3.6}$$

Using the initial frequency scaling factors computed in (3.6), the algorithm computes the initial frequencies for all nodes as a ratio between the maximum frequency of node i and

Algorithm 6 DVFS algorithm of heterogeneous platform

```
1: for k = 1 to some iterations do
2:     Computations section.
3:     Communications section.
4:     if (k = 1) then
5:         Gather all times of computation and communication from each node.
6:         Call Algorithm 5.
7:         Compute the new frequencies from the returned optimal scaling factors.
8:         Set the new frequencies to nodes.
9:     end if
10: end for
```

the computed scaling factor S_{cp_i} as follows:

$$F_i = \frac{F_{max_i}}{S_{cp_i}},\ i = 1, 2, \ldots, N \tag{3.7}$$

If the computed initial frequency for a node is not available in the gears of that node, it is replaced by the nearest available frequency. In Figure 3.3, the nodes are sorted by their computing power in ascending order and the frequencies of the faster nodes are scaled down according to the computed initial frequency scaling factors. The resulting new frequencies are highlighted in Figure 3.3. This set of frequencies can be considered as a higher bound for the search space of the optimal vector of frequencies because selecting scaling factors higher than the higher bound will not improve the performance of the application and it will increase its overall energy consumption. Therefore the algorithm that selects the frequency scaling factors starts the search method from these initial frequencies and takes a downward search direction toward lower frequencies. The algorithm iterates on all remaining frequencies, from the higher bound until all nodes reach their minimum frequencies, to compute their overall energy consumption and performance, and select the optimal frequency scaling factors vector. At each iteration the algorithm determines the slowest node according to Equation (3.1) and keeps its frequency unchanged, while it lowers the frequency of all other nodes by one gear. The new overall energy consumption and execution time are computed according to the new scaling factors. The optimal set of frequency scaling factors is the set that gives the highest distance according to the objective function (3.5).

Figure 3.2 illustrates the normalized performance and consumed energy for an application running on a heterogeneous cluster while increasing the scaling factors. It can be noticed that in a homogeneous cluster, as in the figure 2.2 (a), the search for the optimal scaling factor should start from the maximum frequency because the performance and the consumed energy decrease from the beginning of the plot. On the other hand, in the heterogeneous cluster the performance is maintained at the beginning of the plot even if the frequencies of the faster nodes decrease until the computing power of scaled down nodes are lower than the slowest node. In other words, until they reach the higher bound. It can also be noticed that the higher the difference between the faster nodes and the slower nodes is, the bigger the maximum distance between the energy curve and the performance curve is which results in bigger energy savings.

3.3.5/ THE EVALUATION OF THE PROPOSED ALGORITHM

The accuracy of the proposed algorithm mainly depends on the execution time prediction model defined in (3.1) and the energy model computed by Equation (3.2). The energy model is also significantly dependent on the execution time model because the static energy is linearly related to the execution time and the dynamic energy is related to the computation time. So, all the works presented in this chapter are based on the execution time model. To verify this model, the predicted execution time was compared to the real execution time over SimGrid/SMPI simulator, v3.10 [18], for all the NAS parallel benchmarks NPB v3.3 [57], running class B on 8 or 9 nodes. The comparison showed that the proposed execution time model is very accurate, the maximum normalized difference between the predicted execution time and the real execution time is equal to 0.03 for all the NAS benchmarks.

Since the proposed algorithm is not an exact method, it does not test all the possible solutions (vectors of scaling factors) in the search space. To prove its efficiency, it was compared on small instances to a brute force search algorithm that tests all the possible solutions. The brute force algorithm was applied to different NAS benchmarks classes with different number of nodes. The solutions returned by the brute force algorithm and the proposed algorithm were identical and the proposed algorithm was on average 10 times faster than the brute force algorithm. It has a small execution time: for a heterogeneous cluster composed of four different types of nodes having the characteristics presented in Table 3.1, it takes on average 0.04 *ms* for 4 nodes and 0.15 *ms* on average for 144 nodes to compute the best scaling factors vector. The algorithm complexity is $O(F_i \cdot N)$, where F_i is the maximum number of available frequencies in the node i, and N is the number of computing nodes. The algorithm needs from 12 to 20 iterations to select the best vector of frequency scaling factors that gives the results of the next sections.

Table 3.1: Heterogeneous nodes characteristics

Node type	Simulated GFLOPS	Max Freq. GHz	Min Freq. GHz	Diff. Freq. GHz	Dynamic power	Static power
1	40	2.50	1.20	0.100	20 W	4 W
2	50	2.66	1.60	0.133	25 W	5 W
3	60	2.90	1.20	0.100	30 W	6 W
4	70	3.40	1.60	0.133	35 W	7 W

3.4/ EXPERIMENTAL RESULTS OVER A HETEROGENEOUS LOCAL CLUSTER

To evaluate the efficiency and the overall energy consumption reduction of Algorithm 5, it was applied to the NAS parallel benchmarks NPB v3.3 which is composed of synchronous message passing applications. The experiments were executed on the simulator SimGrid/SMPI which offers easy tools to create a heterogeneous local cluster and run message passing applications over it. The heterogeneous cluster that was used in the experiments, had one core per node because just one process was executed per node. The heterogeneous cluster was composed of four types of nodes. Each type of

nodes had different characteristics such as the maximum CPU frequency, the number of available frequencies and the computational power, see Table 3.1. The characteristics of these different types of nodes are inspired from the specifications of real Intel processors. The heterogeneous cluster had up to 144 nodes and had nodes from the four types in equal proportions, for example if a benchmark was executed on 8 nodes, 2 nodes from each type were used. Since the constructors of CPUs do not specify the dynamic and the static power of their CPUs, for each type of node they were chosen proportionally to their computing powers (FLOPS). The dynamic power corresponds to 80% of the overall power consumption while executing at the higher frequency and the remaining 20% is the static power. The same assumption was made in Chapter 2 and [67]. Finally, These nodes were connected via an Ethernet network with 1 *Gbit/s* bandwidth.

3.4.1/ THE EXPERIMENTAL RESULTS OF THE SCALING ALGORITHM

The proposed algorithm was applied to the seven parallel NAS benchmarks (EP, CG, MG, FT, BT, LU and SP). The benchmarks were executed with class C while being run on different number of nodes, ranging from 8 to 128 or 144 nodes depending on the benchmark being executed. Indeed, the benchmarks CG, MG, LU, EP and FT had to be executed on 1, 2, 4, 8, 16, 32, 64, or 128 nodes. The other benchmarks such as BT and SP had to be executed on 1, 4, 9, 16, 36, 64, or 144 nodes.

Table 3.2: Running NAS benchmarks on 8 and 9 nodes

Program name	Execution time/s	Energy consumption/J	Energy saving%	Performance degradation%	Distance
CG	36.11	3263.49	31.25	7.12	24.13
MG	8.99	953.39	33.78	6.41	27.37
EP	40.39	5652.81	27.04	0.49	26.55
LU	218.79	36149.77	28.23	0.01	28.22
BT	166.89	23207.42	32.32	7.89	24.43
SP	104.73	18414.62	24.73	2.78	21.95
FT	51.10	4913.26	31.02	2.54	28.48

Table 3.3: Running NAS benchmarks on 16 nodes

Program name	Execution time/s	Energy consumption/J	Energy saving%	Performance degradation%	Distance
CG	31.74	4373.90	26.29	9.57	16.72
MG	5.71	1076.19	32.49	6.05	26.44
EP	20.11	5638.49	26.85	0.56	26.29
LU	144.13	42529.06	28.80	6.56	22.24
BT	97.29	22813.86	34.95	5.80	29.15
SP	66.49	20821.67	22.49	3.82	18.67
FT	37.01	5505.60	31.59	6.48	25.11

The overall energy consumption was computed for each instance according to the energy consumption model (3.2), with and without applying the algorithm. The execution

Table 3.4: Running NAS benchmarks on 32 and 36 nodes

Program name	Execution time/s	Energy consumption/J	Energy saving%	Performance degradation%	Distance
CG	32.35	6704.21	16.15	5.30	10.85
MG	4.30	1355.58	28.93	8.85	20.08
EP	9.96	5519.68	26.98	0.02	26.96
LU	99.93	67463.43	23.60	2.45	21.15
BT	48.61	23796.97	34.62	5.83	28.79
SP	46.01	27007.43	22.72	3.45	19.27
FT	28.06	7142.69	23.09	2.90	20.19

Table 3.5: Running NAS benchmarks on 64 nodes

Program name	Execution time/s	Energy consumption/J	Energy saving%	Performance degradation%	Distance
CG	46.65	17521.83	8.13	1.68	6.45
MG	3.27	1534.70	29.27	14.35	14.92
EP	5.05	5471.11	27.12	3.11	24.01
LU	73.92	101339.16	21.96	3.67	18.29
BT	39.99	27166.71	32.02	12.28	19.74
SP	52.00	49099.28	24.84	0.03	24.81
FT	25.97	10416.82	20.15	4.87	15.28

Table 3.6: Running NAS benchmarks on 128 and 144 nodes

Program name	Execution time/s	Energy consumption/J	Energy saving%	Performance degradation%	Distance
CG	56.92	41163.36	4.00	1.10	2.90
MG	3.55	2843.33	18.77	10.38	8.39
EP	2.67	5669.66	27.09	0.03	27.06
LU	51.23	144471.90	16.67	2.36	14.31
BT	37.96	44243.82	23.18	1.28	21.90
SP	64.53	115409.71	26.72	0.05	26.67
FT	25.51	18808.72	12.85	2.84	10.01

time was also measured for all these experiments. Then, the energy saving and performance degradation percentages were computed for each instance. The results are presented in Tables 3.2, 3.3, 3.4, 3.5 and 3.6. All these results are the average values obtained from many experiments for energy savings and performance degradation. The tables show the experimental results for running the NAS parallel benchmarks on different numbers of nodes. The experiments show that the algorithm significantly reduces the energy consumption (up to 34%) and tries to limit the performance degradation. They also show that the energy saving percentage decreases when the number of computing nodes increases. This reduction is due to the increase of the communication times compared to the execution times when the benchmarks are run over a higher number of nodes. Indeed, the benchmarks with the same class, C, are executed on different numbers of nodes, so the computation required for each iteration is divided by the number of computing nodes. On the other hand, more communications are required when

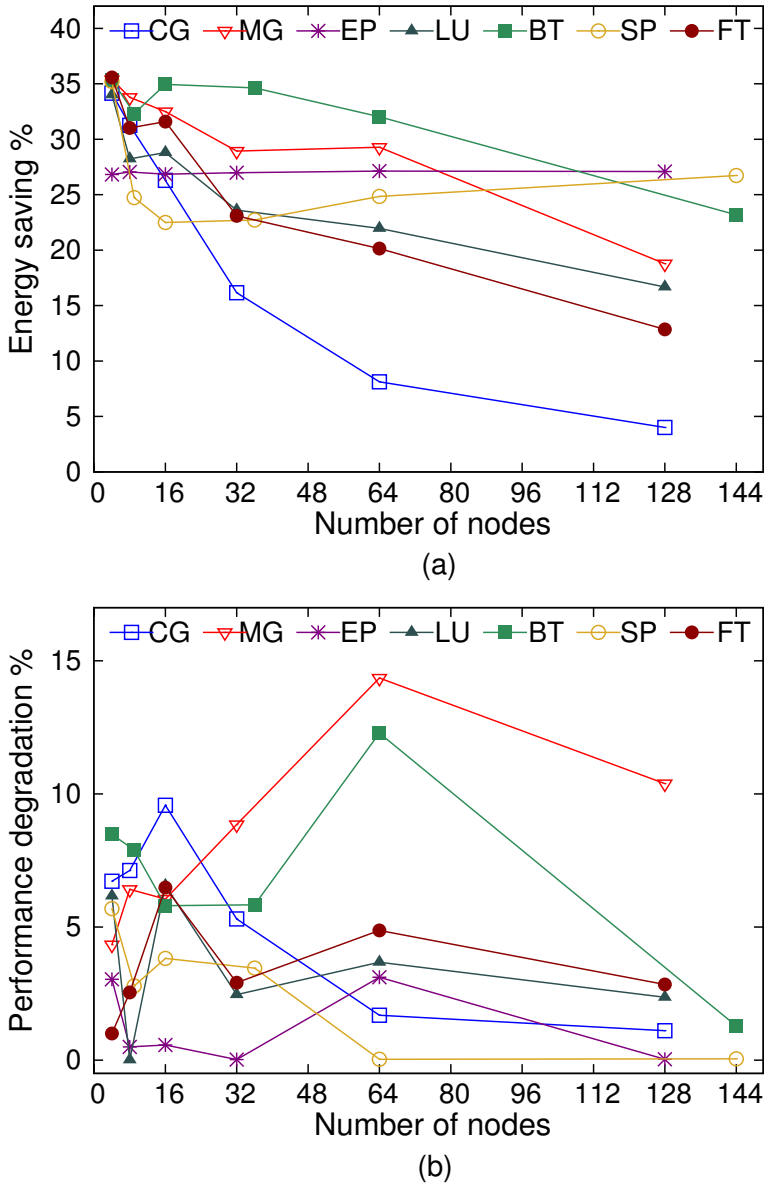

Figure 3.4: NAS benchmarks running with a different number of nodes (a) the energy saving and (b) the performance degradation

increasing the number of nodes so the static energy increases linearly according to the communication time and the dynamic power is less relevant in the overall energy consumption. Therefore, reducing the frequency with Algorithm 5 is less effective in reducing the overall energy savings. It can also be noticed that for the benchmarks EP and SP that contain little or no communications, the energy savings are not significantly affected by the high number of nodes. No experiments were conducted using bigger classes than D, because they require a lot of memory (more than 64 *GB*) when being executed by the simulator on one machine. The maximum distance between the normalized energy curve and the normalized performance for each instance is also shown in the result tables. It decreases in the same way as the energy saving percentage. The tables also show that the performance degradation percentage is not significantly increased when the number of computing nodes is increased because the computation times are small

when compared to the communication times.

Figure 3.4 (a) and (b) present the energy saving and performance degradation respectively for all the benchmarks according to the number of used nodes. As shown in the first plot, the energy saving percentages of the benchmarks MG, LU, BT and FT decrease linearly when the number of nodes increase. While for the EP and SP benchmarks, the energy saving percentage is not affected by the increase of the number of computing nodes, because in these benchmarks there are little or no communications. Finally, the energy saving of the CG benchmark significantly decreases when the number of nodes increase because this benchmark has more communications than the others. The second plot shows that the performance degradation percentages of most of the benchmarks decrease when they run on a big number of nodes because they spend more time communicating than computing, thus, scaling down the frequencies of some nodes has less effect on the performance.

3.4.2/ THE RESULTS FOR DIFFERENT POWER CONSUMPTION SCENARIOS

The results of the previous section were obtained while using processors that consume during computation an overall power which is 80% composed of dynamic power and 20% of static power. In this section, these ratios are changed and two new power scenarios are considered in order to evaluate how the proposed algorithm adapts itself according to the static and dynamic power values. The two new power scenarios are the following:

- 70% of dynamic power and 30% of static power
- 90% of dynamic power and 10% of static power

The NAS parallel benchmarks were executed again over processors that follow the new power scenarios. The class C of each benchmark was run over 8 or 9 nodes and the results are presented in Tables 3.7 and 3.8. These tables show that the energy saving percentage of the 70%-30% scenario is smaller for all benchmarks compared to the energy saving of the 90%-10% scenario. Indeed, in the latter more dynamic power is consumed when nodes are running on their maximum frequencies, thus, scaling down the frequency of the nodes results in higher energy savings than in the 70%-30% scenario. On the other hand, the performance degradation percentage is smaller in the 70%-30% scenario compared to the 90%-10% scenario. This is due to the higher static power percentage in the first scenario which makes it more relevant in the overall consumed energy. Indeed, the static energy is related to the execution time and if the performance is degraded the amount of consumed static energy directly increases. Therefore, the proposed algorithm does not significantly scale down the frequencies of the nodes in order to limit the increase of the execution time and thus limiting the effect of the consumed static energy.

Both new power scenarios are compared to the old one in Figure 3.5 (a). It shows the average of the performance degradation, the energy saving and the distances for all the NAS benchmarks running class C on 8 or 9 nodes. The comparison shows that the energy saving ratio is proportional to the dynamic power ratio: it is increased when applying the 90%-10% scenario because at maximum frequency the dynamic energy is the most relevant in the overall consumed energy and can be reduced by lowering the frequency of some processors. On the other hand, the energy saving decreases

Table 3.7: The results of the 70%-30% power scenario

Program name	Energy consumption/J	Energy saving%	Performance degradation%	Distance
CG	4144.21	22.42	7.72	14.70
MG	1133.23	24.50	5.34	19.16
EP	6170.30	16.19	0.02	16.17
LU	39477.28	20.43	0.07	20.36
BT	26169.55	25.34	6.62	18.71
SP	19620.09	19.32	3.66	15.66
FT	6094.07	23.17	0.36	22.81

Table 3.8: The results of the 90%-10% power scenario

Program name	Energy consumption/J	Energy saving%	Performance degradation%	Distance
CG	2812.38	36.36	6.80	29.56
MG	825.43	38.35	6.41	31.94
EP	5281.62	35.02	2.68	32.34
LU	31611.28	39.15	3.51	35.64
BT	21296.46	36.70	6.60	30.10
SP	15183.42	35.19	11.76	23.43
FT	3856.54	40.80	5.67	35.13

Table 3.9: Comparing the MaxDist algorithm to the EDP method

Program name	Energy saving %		Perf. degradation %		Distance	
	EDP	MaxDist	EDP	MaxDist	EDP	MaxDist
CG	27.58	31.25	5.82	7.12	21.76	24.13
MG	29.49	33.78	3.74	6.41	25.75	27.37
LU	19.55	28.33	0.00	0.01	19.55	28.22
EP	28.40	27.04	4.29	0.49	24.11	26.55
BT	27.68	32.32	6.45	7.87	21.23	24.43
SP	20.52	24.73	5.21	2.78	15.31	21.95
FT	27.03	31.02	2.75	2.54	24.28	28.48

when the 70%-30% scenario is used because the dynamic energy is less relevant in the overall consumed energy and lowering the frequency does not return big energy savings. Moreover, the average of the performance degradation is decreased when using a higher ratio for the static power (e.g. 70%-30% scenario and 80%-20% scenario). Since the proposed algorithm optimizes the energy consumption when using a higher ratio for the dynamic power, the algorithm selects bigger frequency scaling factors that results in more energy saving but degrade the performance, for example see Figure 3.5 (b). The opposite happens when using a higher ratio for the static power, the algorithm proportionally selects smaller scaling values which results in less energy saving but also less performance degradation.

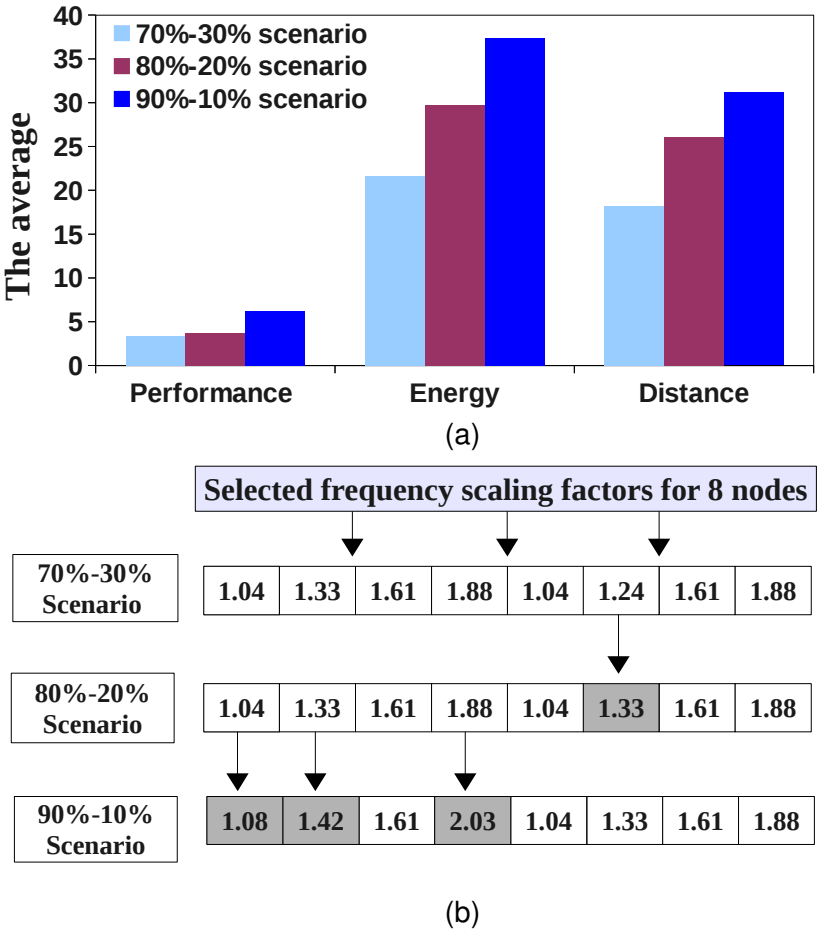

Figure 3.5: (a) Comparison the results of the three power scenarios and (b) Comparison the selected frequency scaling factors of MG benchmark class C running on 8 nodes

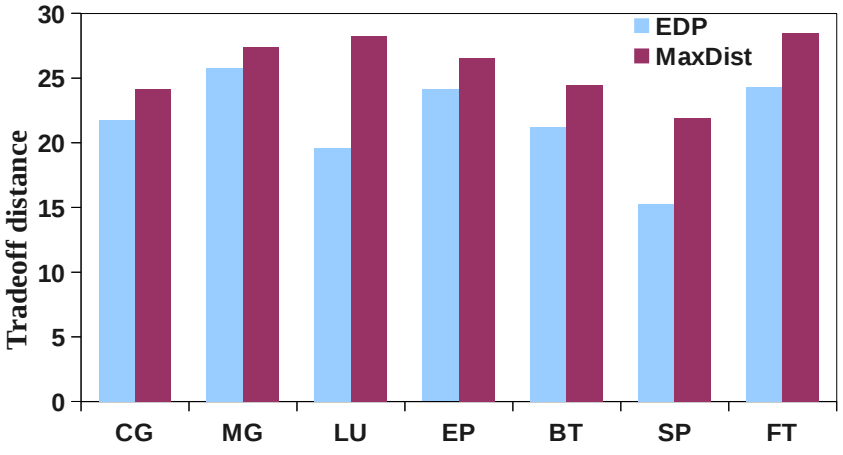

Figure 3.6: Trade-off comparison for NAS benchmarks class C

3.4.3/ COMPARISON BETWEEN THE PROPOSED SCALING ALGORITHM AND THE EDP METHOD

In this section, the scaling factors selection algorithm, called MaxDist, is compared to [75], EDP method. They developed a green governor that regularly applies an online frequency selecting algorithm to reduce the energy consumed by a multi-core architecture without degrading much its performance. The algorithm selects the frequencies that minimize the energy and delay product, $EDP = energy \times delay$, using the predicted overall energy consumption and execution time delay for each frequency. To fairly compare both algorithms, the same energy and execution time models, Equations (3.2) and (3.1), were used for both algorithms to predict the energy consumption and the execution times. Spiliopoulos et al. algorithm was adapted to start the search from the initial frequencies computed using Equation (3.7). The resulting algorithm is an exhaustive search algorithm that minimizes the EDP and has the initial frequencies values as an upper bound.

Both algorithms were applied to the parallel NAS benchmarks to compare their efficiency. Table 3.9 presents the execution times and the energy consumption for both versions of the NAS benchmarks while running the class C of each benchmark over 8 or 9 heterogeneous nodes. The results show that our algorithm provides better energy savings than Spiliopoulos et al. algorithm, on average it results in 29.76% energy saving while their algorithm saves just 25.75%. The average of performance degradation percentage is approximately the same for both algorithms, about 4%.

For all benchmarks, our algorithm outperforms Spiliopoulos et al. algorithm in the energy reduction to performance trade-off, see Figure 3.6, because it maximizes the distance between the energy saving and the performance degradation values while giving the same weight for both metrics.

3.5/ THE ENERGY OPTIMIZATION OF PARALLEL APPLICATIONS WITH ITERATIONS RUNNING OVER GRIDS

3.5.1/ THE ENERGY AND PERFORMANCE MODELS OF GRID PLATFORM

In this section, we are interested in reducing the energy consumption of message passing applications with synchronous iterations running over heterogeneous grid platforms. A heterogeneous grid platform could be defined as a collection of heterogeneous computing clusters interconnected via a long distance network which has a lower bandwidth and a higher latency than the local networks of the clusters. Each computing cluster in the grid is composed of homogeneous nodes that are connected together via a high speed network. However, nodes from distinct clusters may have different characteristics such as computing power (FLOPS), energy consumption, CPU's frequency range, network bandwidth and latency.

Since in a heterogeneous grid each cluster has different characteristics, when applying DVFS operations on the nodes of these clusters, they may get different scaling factors represented by a scaling vector: $(S_{11}, S_{12}, \ldots, S_{NM})$ where S_{ij} is the scaling factor of processor j in cluster i. To be able to predict the execution time of message passing applications with synchronous iterations running over a heterogeneous grid, for different vectors of scaling factors, the communication time and the computation time for all the

tasks must be measured during the first iteration before applying any DVFS operation. Then the execution time for one iteration of the application with any vector of scaling factors can be predicted using Equation (3.8).

$$T_{New} = \max_{\substack{i=1,...N \\ j=1,...,M_i}} (T_{cpOld_{ij}} \cdot S_{ij}) + \min_{j=1,...,M_i} (T_{cm_{hj}}) \tag{3.8}$$

where N is the number of clusters in the grid, M_i is the number of nodes in cluster i, $T_{cpOld_{ij}}$ is the computation time of processor j in the cluster i and $T_{cm_{hj}}$ is the communication time of processor j in the cluster h during the first iteration. The execution time for one iteration is equal to the sum of the maximum computation time for all nodes with the new scaling factors and the slowest communication time without slack time during one iteration. The latter is equal to the communication time of the slowest node in the slowest cluster h. It means that only the communication time without any slack time is taken into account. Therefore, the execution time of the parallel application with iterations is equal to the execution time of one iteration as in Equation (3.8) multiplied by the number of iterations of that application.

In the considered heterogeneous grid platform, each node j in cluster i may have different dynamic and static powers from the nodes of the other clusters, noted as $P_{d_{ij}}$ and $P_{s_{ij}}$ respectively. Therefore, even if the distributed message passing application with iterations is load balanced, the computation time of each CPU j in cluster i noted $T_{cp_{ij}}$ may be different and different frequency scaling factors may be computed in order to decrease the overall energy consumption of the application and reduce slack times. The communication time of a processor j in cluster i is noted as $T_{cm_{ij}}$ and could contain slack times when communicating with slower nodes, see Figure 3.1. Therefore, all nodes do not have equal communication times. While the dynamic energy is computed according to the frequency scaling factor and the dynamic power of each node as in Equation (1.16), the static energy is computed as the sum of the execution time of one iteration multiplied by the static power of each processor. The overall energy consumption of a message passing distributed application executed over a heterogeneous grid platform during one iteration is the summation of all dynamic and static energies for M_i processors in N clusters. It is computed as follows:

$$E = \sum_{i=1}^{N} \sum_{i=1}^{M_i} (S_{ij}^{-2} \cdot P_{d_{ij}} \cdot T_{cp_{ij}}) + \sum_{i=1}^{N} \sum_{j=1}^{M_i} (P_{s_{ij}} \cdot (\max_{\substack{i=1,...N \\ j=1,...,M_i}} (T_{cp_{ij}} \cdot S_{ij}) + \min_{j=1,...M_i} (T_{cm_{hj}}))) \tag{3.9}$$

To optimize both of the energy consumption model computed by 3.9 and the performance model computed by 3.8, they must be normalized as in Equation 3.3 and Equation 3.4 respectively. While the original energy consumption is the consumed energy with the maximum frequency for all the nodes computed as follows:

$$E_{Original} = \sum_{i=1}^{N} \sum_{j=1}^{M_i} (P_{d_{ij}} \cdot T_{cp_{ij}}) + \sum_{i=1}^{N} \sum_{j=1}^{M_i} (P_{s_{ij}} \cdot T_{Old}) \tag{3.10}$$

By the same way, the old execution time with the maximum frequency for all the nodes is computed as follows:

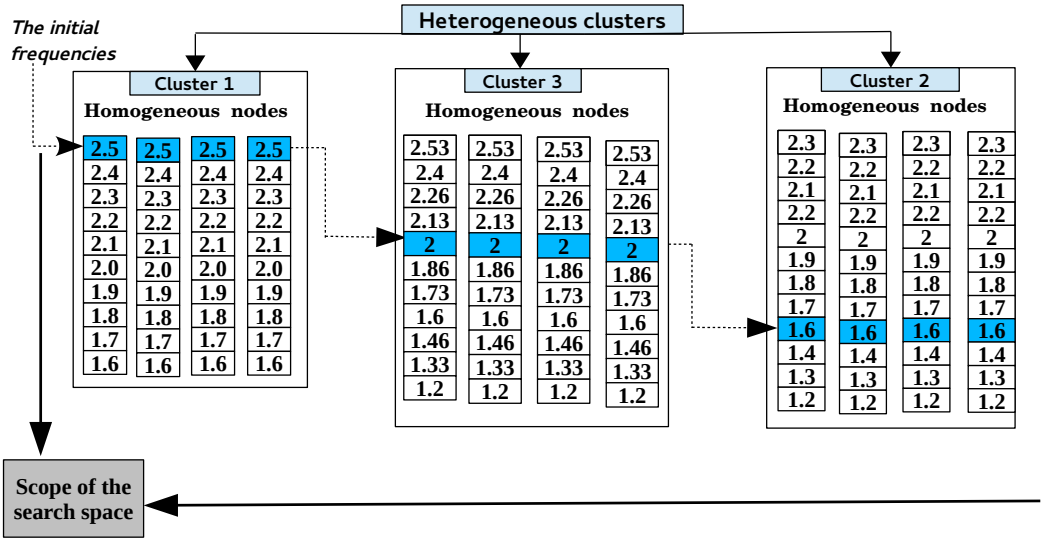

Figure 3.7: Selecting the initial frequencies in the grid architecture

$$T_{Old} = \max_{\substack{i=1,...N \\ j=1,...,M_i}} (T_{cp_{ij}}) + \min_{j=1,...,M_i} (T_{cm_{hj}}) \tag{3.11}$$

Therefore, the objective function can be modelled in order to find the maximum distance between the normalized energy curve and the normalized performance curve over all possible sets of scaling factors as follows:

$$MaxDist = \max_{\substack{i=1,...N \\ j=1,...,M_i \\ k=1,...,F_j}} (\overbrace{P_{Norm}(S_{ijk})}^{\text{Maximize}} - \overbrace{E_{Norm}(S_{ijk})}^{\text{Minimize}}) \tag{3.12}$$

where N is the number of clusters, M_i is the number of nodes in each cluster and F_j is the number of available frequencies for the node j. Then, the optimal set of scaling factors that satisfies (3.12) can be selected.

3.5.2/ THE SCALING FACTORS SELECTION ALGORITHM FOR A GRID ARCHITECTURE

In this section, the scaling factors selection algorithm for a grid, Algorithm 7, is presented. It selects the vector of frequency scaling factors that gives the best trade-off between minimizing the energy consumption and maximizing the performance of a message passing application with synchronous iterations executed on a grid. It is similar to the frequency selection algorithm for heterogeneous local clusters presented in Section 3.3.4.

The value of the initial frequency scaling factor for each node is inversely proportional to its computation time that was gathered in the first iteration. The initial frequency scaling factor for a node i is computed as a ratio between the computation time of the slowest

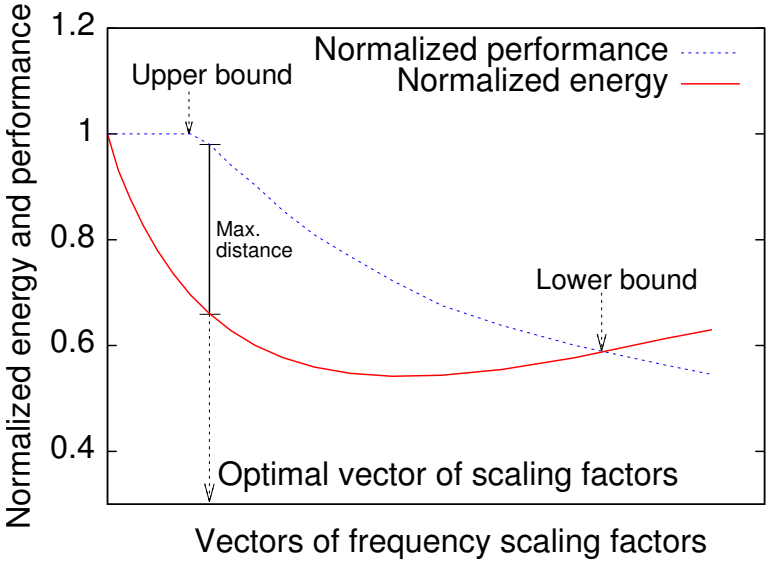

Figure 3.8: The energy and performance relation in grid

node and the computation time of the node i as follows:

$$S_{cp_{ij}} = \frac{\max\limits_{\substack{i=1,\ldots,N \\ j=1,\ldots,M_i}} (T_{cp_{ij}})}{T_{cp_{ij}}} \tag{3.13}$$

Using the initial frequency scaling factors computed in (3.13), the algorithm computes the initial frequencies for all nodes as a ratio between the maximum frequency of the node and its computed scaling factor, as follows:

$$F_{ij} = \frac{F_{max_{ij}}}{S_{cp_{ij}}},\ i = 1, 2, \ldots, N,\ j = 1, \ldots, M_i \tag{3.14}$$

Figure 3.7 shows the selected initial frequencies for a grid composed of three clusters. In contrast to algorithm 5, algorithm 7 replaces the computed initial frequency for a node by the nearest available frequency if not available in the gears of that node. The frequency scaling algorithm of the grid stops its iteration if it reaches the lower bound, which is the frequency that gives a negative distance between the energy and performance. A negative distance means that the performance degradation ratio is higher than the energy saving ratio as in figure 3.8. In this situation, the algorithm must stop the downward search because it has reached the lower bound and it is useless to test the lower frequencies. Indeed, they will all give worse distances. Therefore, the algorithm iterates on all the remaining frequencies, from the higher bound until all nodes reach their minimum frequencies or their lower bounds, to compute the overall energy consumption and execution time. Then, it selects the vector of frequency scaling factors that give the maximum distance (MaxDist). Algorithm 6 is also used to call the Algorithm 7 in the MPI program executed over the grid platform.

Algorithm 7 Scaling factors selection algorithm for grid

Require:

N number of clusters in the grid.

M_i number of nodes in each cluster.

$T_{cp_{ij}}$ array of all computation times for all nodes during one iteration and with the highest frequency.

$T_{cm_{ij}}$ array of all communication times for all nodes during one iteration and with the highest frequency.

$F_{max_{ij}}$ array of the maximum frequencies for all nodes.

$P_{d_{ij}}$ array of the dynamic powers for all nodes.

$P_{s_{ij}}$ array of the static powers for all nodes.

$F_{diff_{ij}}$ array of the differences between two successive frequencies for all nodes.

Ensure: $S_{opt_{11}}, S_{opt_{12}} \ldots, S_{opt_{NM_i}}$, a vector of scaling factors that gives the optimal trade-off between energy consumption and execution time

1: $S_{cp_{ij}} \leftarrow \frac{\max\limits_{\substack{i=1,\ldots N \\ j=1,\ldots,M_i}} (T_{cp_{ij}})}{T_{cp_{ij}}}$

2: $F_{ij} \leftarrow \frac{F_{max_{ij}}}{S_{cp_i}},\ i = 1, 2, \cdots, N,\ j = 1, 2, \ldots, M_i.$

3: Round the computed initial frequencies F_i to the closest available frequency for each node.

4: **if** (not the first frequency) **then**

5: $\quad F_{ij} \leftarrow F_{ij} + F_{diff_{ij}},\ i = 1, \ldots, N,\ j = 1, \ldots, M_i.$

6: **end if**

7: $T_{Old} \leftarrow \max\limits_{\substack{i=1,\ldots N \\ j=1,\ldots,M_i}} (T_{cp_{ij}}) + \min\limits_{j=1,\ldots,M_i} (T_{cm_{hj}})$

8: $E_{Original} \leftarrow \sum_{i=1}^{N} \sum_{j=1}^{M_i} (P_{d_{ij}} \cdot T_{cp_{ij}}) + \sum_{i=1}^{N} \sum_{j=1}^{M_i} (P_{s_{ij}} \cdot T_{Old})$

9: $S_{opt_{ij}} \leftarrow 1,\ i = 1, \ldots, N,\ j = 1, \ldots, M_i.$

10: $Dist \leftarrow 0$

11: **while** (all nodes have not reached their minimum frequency **or** $P_{Norm} - E_{Norm} < 0$) **do**

12: $\quad$ **if** (not the last freq. **and** not the slowest node) **then**

13: $\quad\quad F_{ij} \leftarrow F_{ij} - F_{diff_{ij}},\ i = 1, \ldots, N,\ j = 1, \ldots, M_i.$

14: $\quad\quad S_{ij} \leftarrow \frac{F_{max_{ij}}}{F_{ij}},\ i = 1, \ldots, N,\ j = 1, \ldots, M_i.$

15: $\quad$ **end if**

16: $\quad T_{New} \leftarrow \max\limits_{\substack{i=1,\ldots N \\ j=1,\ldots,M_i}} (T_{cpOld_{ij}} \cdot S_{ij}) + \min\limits_{j=1,\ldots,M_i} (T_{cm_{hj}})$

17: $\quad E_{Reduced} \leftarrow \sum\limits_{i=1}^{N} \sum\limits_{i=1}^{M_i} (S_{ij}^{-2} \cdot P_{d_{ij}} \cdot T_{cp_{ij}}) + \sum\limits_{i=1}^{N} \sum\limits_{j=1}^{M_i} (P_{s_{ij}} \cdot (\max\limits_{\substack{i=1,\ldots N \\ j=1,\ldots,M_i}} (T_{cp_{ij}} \cdot S_{ij}) + \min\limits_{j=1,\ldots,M_i} (T_{cm_{hj}})))$

18: $\quad P_{Norm} \leftarrow \frac{T_{Old}}{T_{New}}$

19: $\quad E_{Norm} \leftarrow \frac{E_{Reduced}}{E_{Original}}$

20: $\quad$ **if** $(P_{Norm} - E_{Norm} > Dist)$ **then**

21: $\quad\quad S_{opt_{ij}} \leftarrow S_{ij},\ i = 1, \ldots, N,\ j = 1, \ldots, M_i$

22: $\quad\quad Dist \leftarrow P_{Norm} - E_{Norm}$

23: $\quad$ **end if**

24: **end while**

25: Return $S_{opt_{11}}, S_{opt_{12}}, \ldots, S_{opt_{NM_i}}$

3.6/ EXPERIMENTAL RESULTS OVER THE GRID5000 PLATFORM

In this section, real experiments were conducted over the Grid'5000 platform. Grid'5000 [3] is a large-scale testbed that consists of ten sites distributed all over metropolitan France and Luxembourg. These sites are: Grenoble, Lille, Luxembourg, Lyon, Nancy, Reims, Rennes , Sophia, Toulouse and Bordeaux. Figure 3.9 shows the geographical distribution of grid'5000 sites over France and Luxembourg. All the sites are connected together via a special long distance network called RENATER, which is the abbreviation of the French National Telecommunication Network for Technology. Each site in the grid is composed of a few heterogeneous computing clusters and each cluster contains many homogeneous nodes. In total, Grid'5000 has about one thousand heterogeneous nodes and eight thousand cores. In each site, the clusters and their nodes are connected via high speed local area networks. Two types of local networks are used, Ethernet or Infiniband networks, which have different characteristics in terms of bandwidth and latency. Grid'5000 is dedicated for research experiments and users can book nodes from different sites to conduct their experiments. It also gives the opportunity to the users to deploy their customized operating system over the reserved nodes. Indeed, many software tools are available for users in order to control and manage the reservation and deployment processes remotely. For example, OAR [5] is a batch scheduler that is used to manage the heterogeneous resources of the grid'5000.

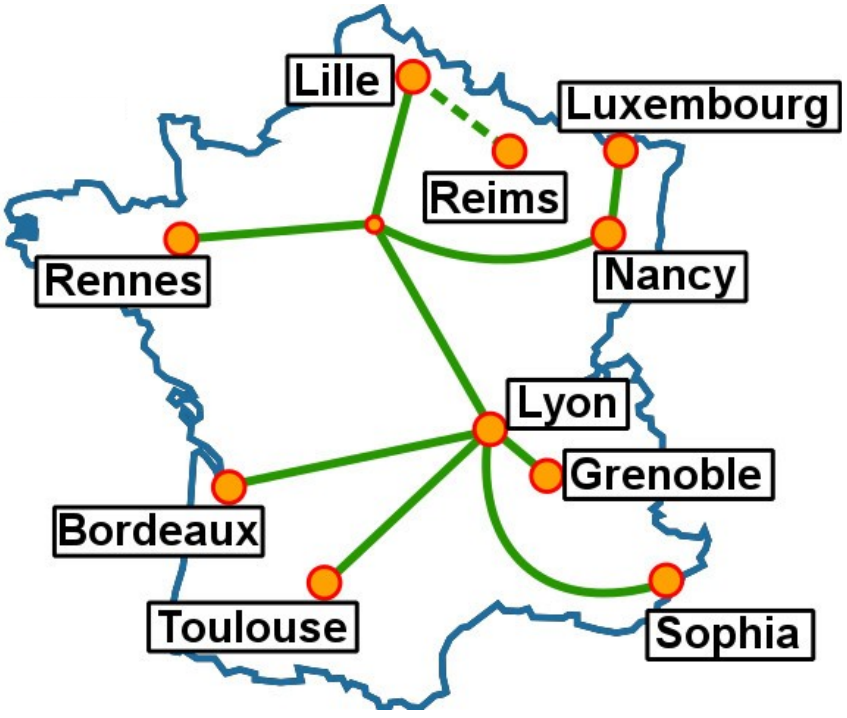

Figure 3.9: Grid5000's sites distribution in France and Luxembourg

Moreover, the Grid'5000 testbed provides at some sites a power measurement tool to capture the power consumption for each node in those sites. The measured power is the overall consumed power by all the components of a node at a given instant. For more details refer to [68]. In order to correctly measure the CPU power of one core in a node j,

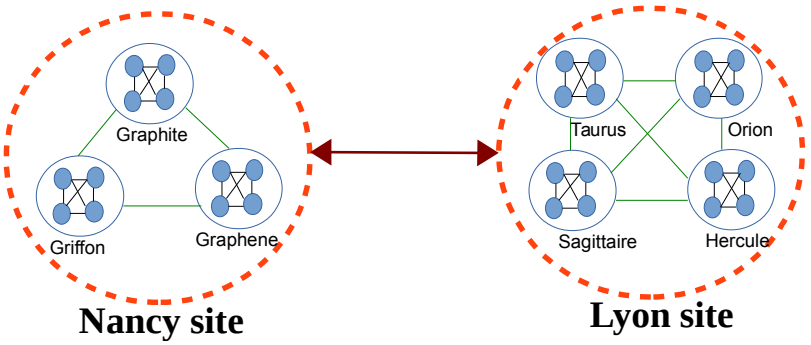

Figure 3.10: The selected two sites of Grid'5000

firstly, the power consumed by the node while being idle at instant y, noted as $P_{idle_{jy}}$, was measured. Then, the power was measured while running a single thread benchmark with no communication (no idle time) over the same node with its CPU scaled to the maximum available frequency. The latter power measured at time x with maximum frequency for one core of node j is noted $P_{max_{jx}}$. The difference between the two measured power consumptions represents the dynamic power consumption of that core with the maximum frequency, see Figure 3.11.

The dynamic power P_{d_j} is computed as in Equation 3.15

$$P_{d_j} = \max_{x=\beta_1,\ldots\beta_2}(P_{max_{jx}}) - \min_{y=\Theta_1,\ldots\Theta_2}(P_{idle_{jy}}) \tag{3.15}$$

where P_{d_j} is the dynamic power consumption for one core of node j, $\{\beta_1,\beta_2\}$ is the time interval for the measured maximum power values, $\{\Theta_1,\Theta_2\}$ is the time interval for the measured idle power values. Therefore, the dynamic power of one core is computed as the difference between the maximum measured value in maximum powers vector and the minimum measured value in the idle powers vector.

On the other hand, the static power consumption by one core is a part of the measured idle power consumption of the node. Since in Grid'5000 there is no way to measure precisely the consumed static power and it was assumed, as in Sections 3.4 and 2.7, that the static power represents a ratio of the dynamic power, the value of the static power is assumed to be equal to 20% of the dynamic power consumption of the core.

In the experiments presented in the following sections, two sites of Grid'5000 were used, Lyon and Nancy sites. These two sites have in total seven different clusters as shown in Figure 3.10.

Four clusters from the two sites were selected in the experiments: one cluster from Lyon's site, Taurus, and three clusters from Nancy's site, Graphene, Griffon and Graphite. Each one of these clusters composed of homogeneous nodes, while nodes from different clusters are heterogeneous in many aspects such as: computing power, power consumption, available frequency ranges and local network features: the bandwidth and the latency. Table 3.10 shows the detailed characteristics of these four clusters. Moreover, the dynamic powers were computed using Equation 3.15 for all the nodes in the selected clusters and are presented in Table 3.10.

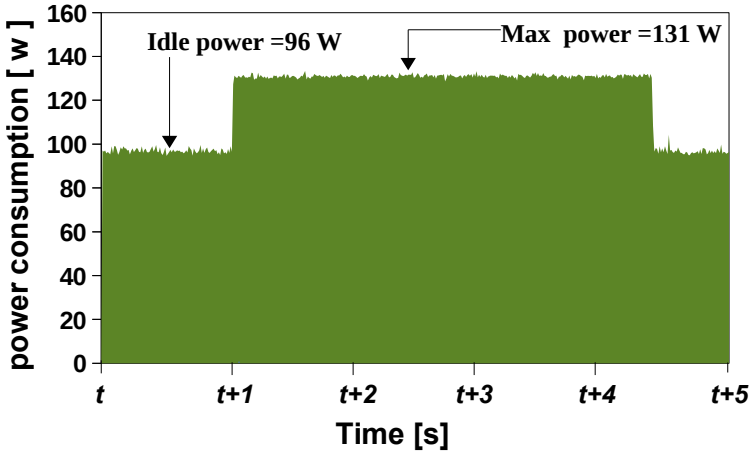

Figure 3.11: The power consumed by one core from the Taurus cluster

Table 3.10: The characteristics of the CPUs in the selected clusters

Cluster Name	CPU model	Max Freq. GHz	Min Freq. GHz	Diff. Freq. GHz	Cores per CPU	Dynamic power of one core
Taurus	Intel Xeon E5-2630	2.3	1.2	0.1	6	35 W
Graphene	Intel Xeon X3440	2.53	1.2	0.133	4	23 W
Griffon	Intel Xeon L5420	2.5	2	0.5	4	46 W
Graphite	Intel Xeon E5-2650	2	1.2	0.1	8	35 W

The energy model and the scaling factors selection algorithm were applied to the NAS parallel benchmarks v3.3 [57] and evaluated over Grid'5000. The benchmark suite contains seven applications: CG, MG, EP, LU, BT, SP and FT. These applications have different computations and communications ratios and strategies which make them good testbed applications to evaluate the proposed algorithm and energy model. The benchmarks have seven different classes, S, W, A, B, C, D and E, that represent the size of the problem that the method solves. In the next sections, the class D was used for all the benchmarks in all the experiments.

3.6.1/ THE EXPERIMENTAL RESULTS OF THE SCALING ALGORITHM ON A GRID

In this section, the results of applying the scaling factors selection algorithm to the NAS parallel benchmarks are presented. As mentioned previously, the experiments were conducted over two sites of Grid'5000, Lyon and Nancy sites. Two scenarios were considered

while selecting the clusters from these two sites :

- In the first scenario, nodes from two sites and three heterogeneous clusters were selected. The two sites are connected via a long distance network.
- In the second scenario nodes from three clusters located in one site, Nancy's site, were selected.

The main reason for using these two scenarios is to evaluate the influence of long distance communications (higher latency) on the performance of the scaling factors selection algorithm. Indeed, in the first scenario the computations to communications ratio is very low due to the higher communication times which reduces the effect of the DVFS operations.

The NAS parallel benchmarks are executed over 16 and 32 nodes for each scenario. The number of participating computing nodes from each cluster is different because all the selected clusters do not have the same available number of nodes and all benchmarks do not require the same number of computing nodes. Table 3.11 shows the number of nodes used from each cluster for each scenario.

Table 3.11: The different grid scenarios

Scenario name	The participating clusters		
	Cluster	Site	Nodes per cluster
Two sites / 16 nodes	Taurus	Lyon	5
	Graphene	Nancy	5
	Griffon	Nancy	6
Two sites / 32 nodes	Taurus	Lyon	10
	Graphene	Nancy	10
	Griffon	Nancy	12
One site / 16 nodes	Graphite	Nancy	4
	Graphene	Nancy	6
	Griffon	Nancy	6
One site / 32 nodes	Graphite	Nancy	4
	Graphene	Nancy	14
	Griffon	Nancy	14

The NAS parallel benchmarks are executed over these two platforms with different number of nodes, as in Table 3.11. The overall energy consumption of all the benchmarks solving the class D instance and using the proposed frequency selection algorithm is measured using Equation 3.9.

The energy consumptions and the execution times for all the benchmarks are presented in Figures 3.12 (a) and (b) respectively. For the majority of the benchmarks, the energy consumed while executing the NAS benchmarks over one site on 16 and 32 nodes is lower than the energy consumed while using two sites. The long distance communications between the two distributed sites increase the idle times, which lead to more static energy consumption.

The execution times of these benchmarks over one site with 16 and 32 nodes are also lower than those of the two sites scenario. Moreover, most of the benchmarks running over the one site scenario have their execution times approximately halved when the

number of computing nodes is doubled from 16 to 32 nodes (linear speed up according to the number of the nodes).

However, the execution times and the energy consumptions of the EP and MG benchmarks, which have no or small communications, are not significantly affected in both scenarios, even when the number of nodes is doubled. On the other hand, the communication times of the rest of the benchmarks increase when using long distance communications between two sites or when increasing the number of computing nodes.

The energy saving percentage is computed as the ratio between the reduced energy consumption, Equation 3.9, and the original energy consumption, Equation 3.10, for all the benchmarks as in Figure 3.13. This figure shows that the energy saving percentages of the one site scenario for 16 and 32 nodes are bigger than those of the two sites scenario which is due to the higher computations to communications ratio in the first scenario than in the second one. Moreover, the frequency selecting algorithm selects smaller frequencies when the computation times are bigger than the communication times which results in a lower energy consumption. Indeed, the dynamic consumed power is exponentially related to the CPU's frequency value. On the other hand, the increase in the number of computing nodes can increase the communication times and thus produces less energy saving depending on the benchmarks being executed. The results of the benchmarks CG, MG, BT and FT show more energy saving percentage in the one site scenario when executed over 16 nodes than on 32 nodes. LU and SP consume more energy with 16 nodes than with 32 nodes on one site because their computations to communications ratio is not affected by the increase of the number of local communications.

The energy saving percentage is reduced for all the benchmarks because of the long distance communications in the two sites scenario, except for the EP benchmark which has no communication. Therefore, the energy saving percentage of this benchmark is dependent on the maximum difference between the computing powers of the heterogeneous computing nodes, for example in the one site scenario, the graphite cluster is selected but in the two sites scenario this cluster is replaced with the Taurus cluster which is more powerful. Therefore, the energy savings of the EP benchmark are bigger in the two sites scenario due to the higher maximum difference between the computing powers of the nodes.

In fact, high differences between the nodes' computing powers make the proposed frequencies selecting algorithm select smaller frequencies for the powerful nodes which produces less energy consumption and thus more energy saving. The best energy saving percentage was obtained in the one site scenario with 16 nodes, the energy consumption was on average reduced up to 30%.

Figure 3.14 presents the performance degradation percentages for all the benchmarks over the two scenarios. The performance degradation percentage for the benchmarks running on two sites with 16 and 32 nodes is on average equal to 8.3% and 4.7% respectively. For this scenario, the proposed scaling algorithm selects smaller frequencies for the executions with 32 nodes without significantly degrading their performance because the communication times are high with 32 nodes which results in smaller computations to communications ratio. On the other hand, the performance degradation percentage for the benchmarks running on one site with 16 and 32 nodes is on average equal to 3.2% and 10.6% respectively. In contrary to the two sites scenario, when the number of computing nodes is increased in the one site scenario, the performance degradation

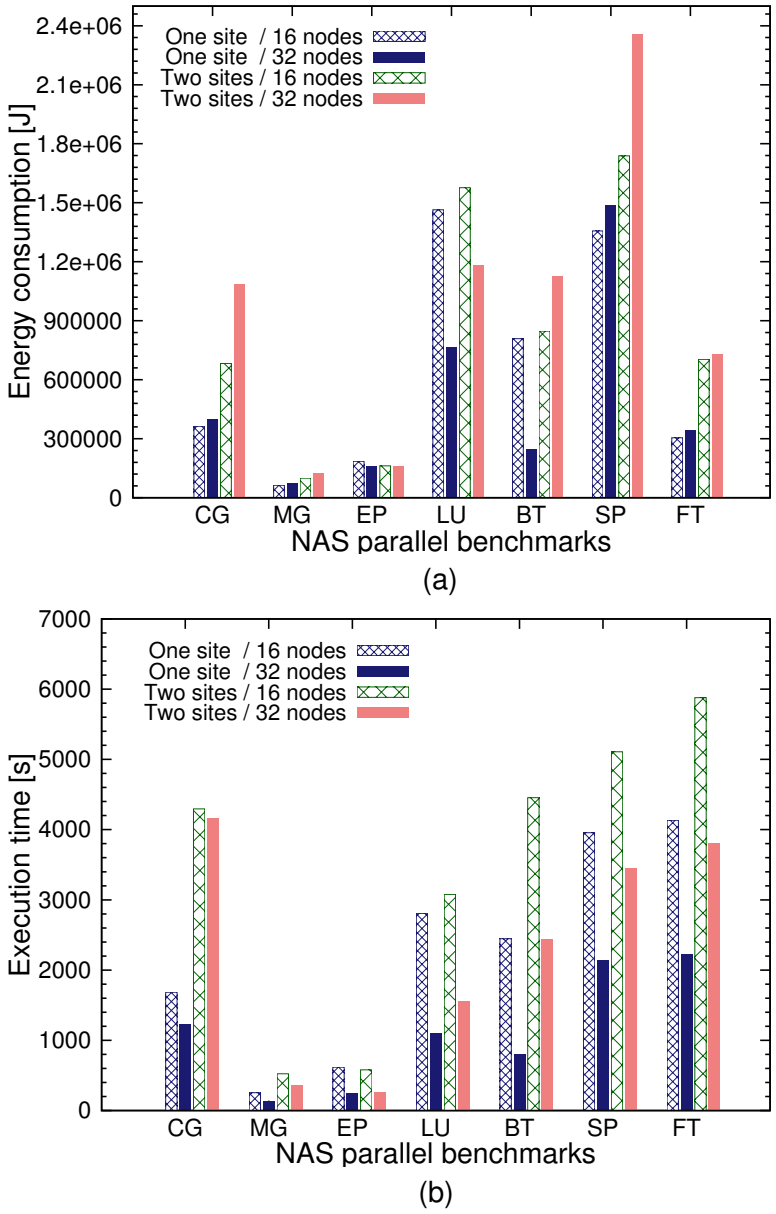

Figure 3.12: (a) energy consumption and (b) execution time of NAS Benchmarks over different scenarios

percentage is increased. Therefore, doubling the number of computing nodes when the communications occur in high speed network does not decrease the computations to communication ratio.

The performance degradation percentage of the EP benchmark after applying the scaling factors selection algorithm is the highest in comparison to the other benchmarks. Indeed, in the EP benchmark, there are no communication and slack times and its performance degradation percentage only depends on the frequencies values selected by the algorithm for the computing nodes. The rest of the benchmarks showed different performance degradation percentages which decrease when the communication times increase and vice versa.

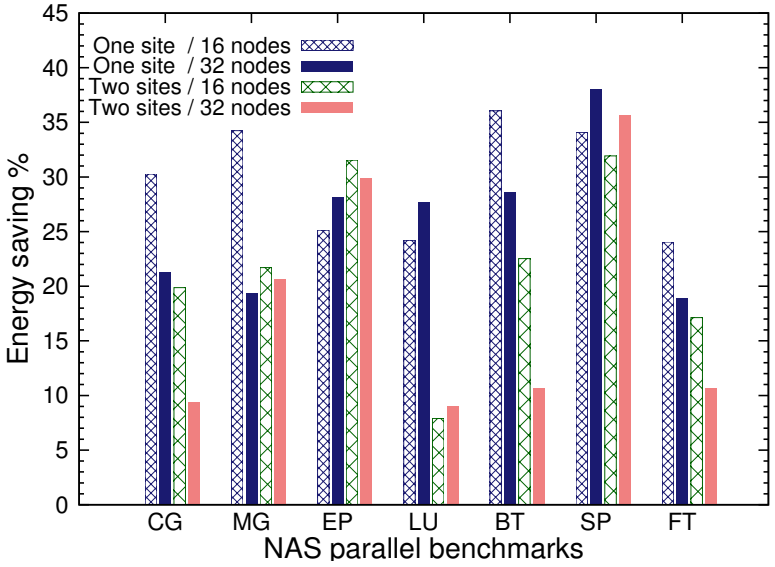

Figure 3.13: The energy reduction percentages while executing the NAS benchmarks over different scenarios

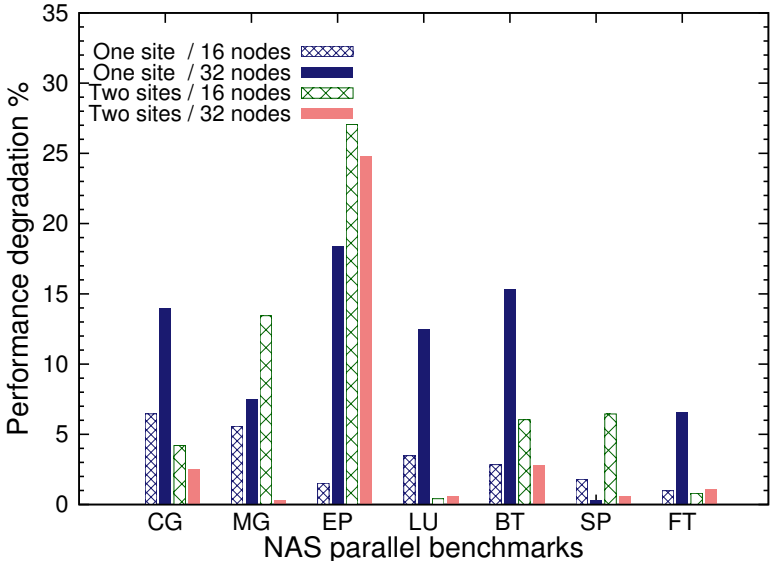

Figure 3.14: The performance degradation percentages of the NAS benchmarks over different scenarios

Figure 3.15 presents the distance percentage between the energy saving and the performance degradation for each benchmark over both scenarios. The trade-off distance percentage can be computed as in Equation 3.12. The one site scenario with 16 nodes gives the best energy and performance trade-off, on average it is equal to 26.8%. The one site scenario using both 16 and 32 nodes had better energy and performance trade-off comparing to the two sites scenario because the former has high speed local communications which increase the computations to communications ratio and the latter uses long distance communications which decrease this ratio.

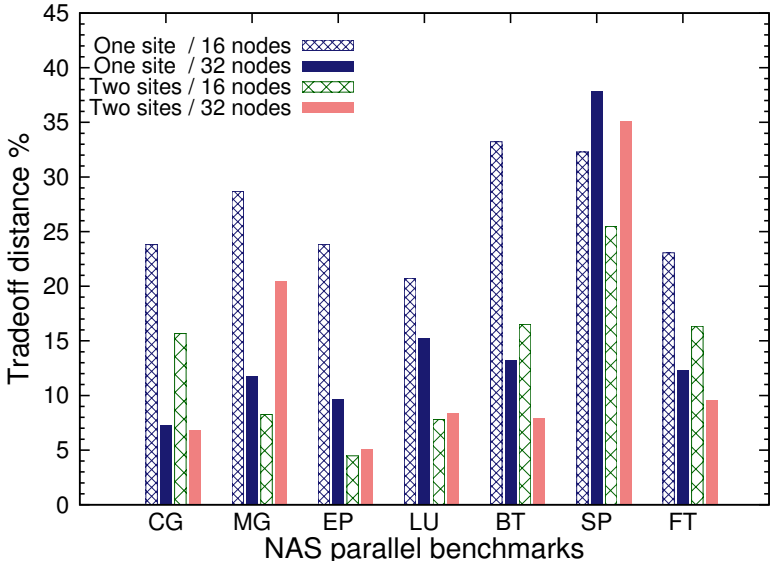

Figure 3.15: The trade-off distance percentages between the energy reduction and the performance of the NAS benchmarks over different scenarios

Finally, the best energy and performance trade-off depends on all of the following: 1) the computations to communications ratio when there are communications and slack times, 2) the heterogeneity of the computing powers of the nodes and 3) the heterogeneity of the consumed static and dynamic powers of the nodes.

3.6.2/ THE EXPERIMENTAL RESULTS OVER MULTI-CORE CLUSTERS

The clusters of Grid'5000 have different number of cores embedded in their nodes as shown in Table 3.10. In this section, the proposed scaling algorithm is evaluated over the Grid'5000 platform while using multi-core nodes selected according to the one site scenario described in Section 3.6.1.

Table 3.12: The multi-core scenarios

Scenario name	Cluster name	Nodes per cluster	Cores per node
One core per node	Graphite	4	1
	Graphene	14	1
	Griffon	14	1
Multi-core per node	Graphite	1	4
	Graphene	4	3 or 4
	Griffon	4	3 or 4

The one site scenario uses 32 cores from multi-core nodes instead of 32 distinct nodes. For example if the participating number of cores from a certain cluster is equal to 14, in the multi-core 4 nodes are selected and 3 or 4 cores from each node are used. The platforms with one core per node and multi-core nodes are shown in Table 3.12. The energy consumptions and execution times of running class D of the NAS parallel benchmarks over these two different platforms are presented in Figures 3.17 and 3.16 respectively.

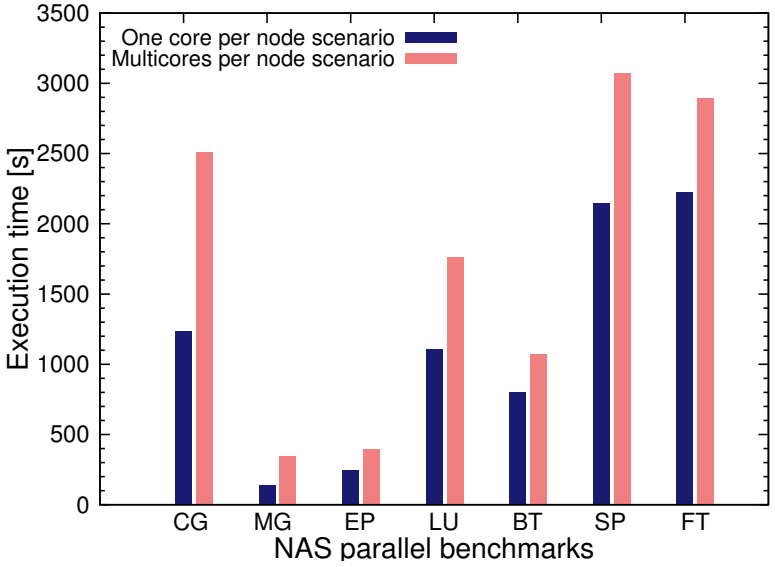

Figure 3.16: The execution times of the NAS benchmarks running over the one core and the multi-core scenarios

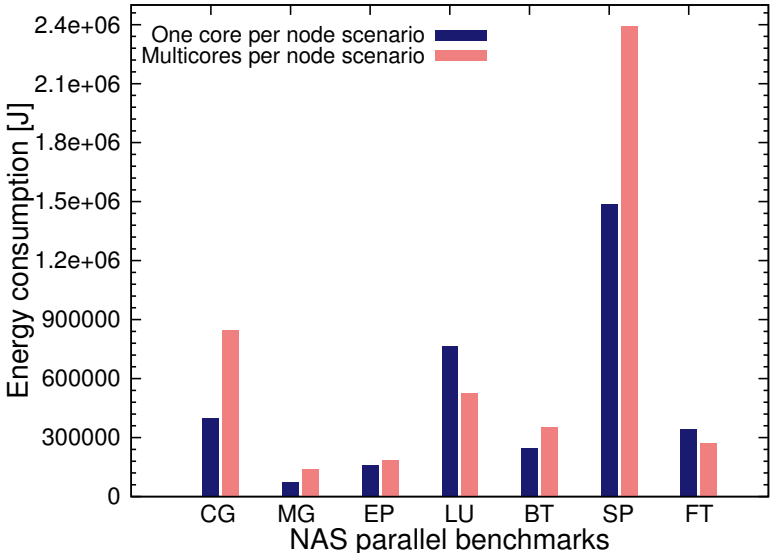

Figure 3.17: The energy consumptions and execution times of the NAS benchmarks over one core and multi-core per node architectures

The execution times for most of the NAS benchmarks are higher over the multi-core per node scenario than over the single core per node scenario. Indeed, the communication times are higher in the one site multi-core scenario than in the latter scenario because all the cores of a node share the same node network link which can be saturated when running communication bound applications. Moreover, the cores of a node share the memory bus which can be also saturated and might become a bottleneck. Moreover, the energy consumptions of the NAS benchmarks are lower over the one core scenario than over the multi-core scenario because the first scenario had less execution time than the latter which results in less static energy being consumed. The computations to communications ratios of the NAS benchmarks are higher over the one site one core scenario

when compared to the ratio of the multi-core scenario. More energy reduction can be gained when this ratio is big because it pushes the proposed scaling algorithm to select smaller frequencies that decrease the dynamic power consumption. These experiments also showed that the energy consumption and the execution times of the EP and MG benchmarks do not change significantly over these two scenarios because there are no or small communications. Contrary to EP and MG, the energy consumptions and the execution times of the rest of the benchmarks vary according to the communication times that are different from one scenario to the other.

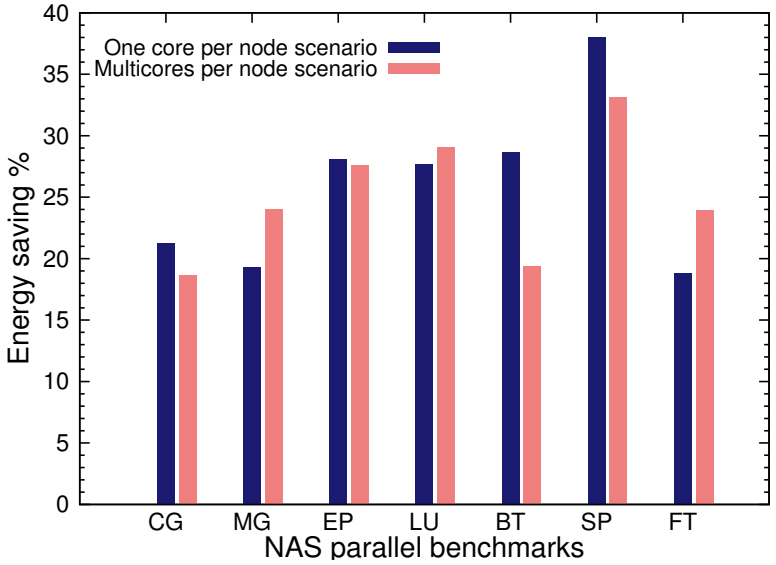

Figure 3.18: The energy saving percentages of running NAS benchmarks over one core and multi-core scenarios

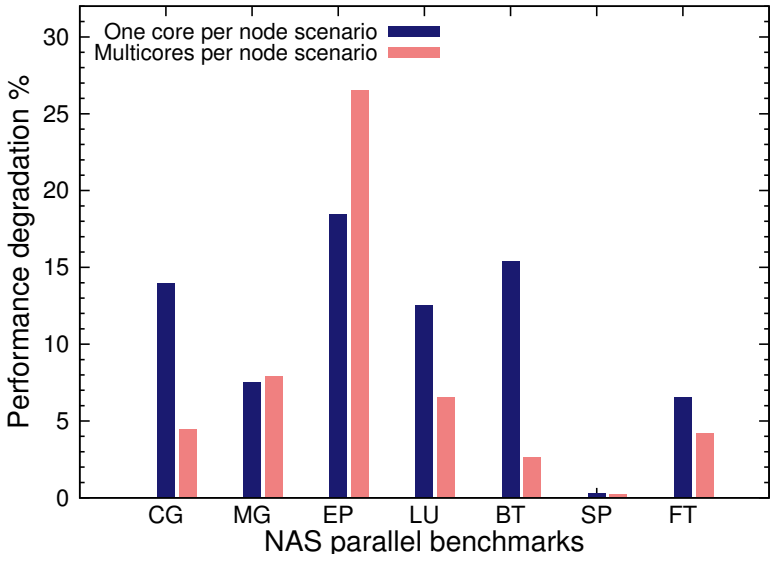

Figure 3.19: The performance degradation percentages of running NAS benchmarks over one core and multi-core scenarios

The energy saving percentages of all the NAS benchmarks running over these two

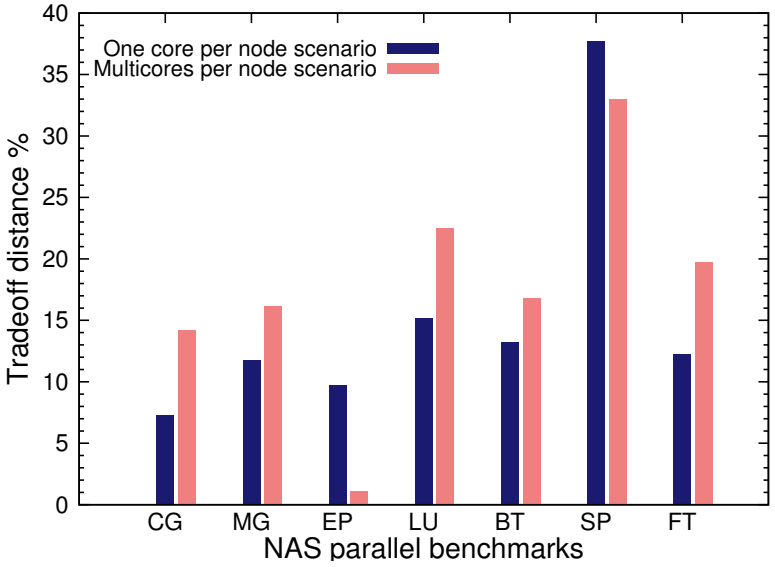

Figure 3.20: The trade-off distance percentages of running NAS benchmarks over one core and multi-core scenarios

scenarios are presented in Figure 3.18. It shows that the energy saving percentages in the one core and the multi-core scenarios are approximately equivalent, on average they are equal to 25.9% and 25.1% respectively. The energy consumption is reduced at the same rate in the two scenarios when compared to the energy consumption of the executions without DVFS.

The performance degradation percentages of the NAS benchmarks are presented in Figure 3.19. It shows that the performance degradation percentages are higher for the NAS benchmarks executed over the one core per node scenario (on average equal to 10.6%) than over the multi-core scenario (on average equal to 7.5%). The performance degradation percentages over the multi-core scenario are lower because the computations to communications ratios are smaller than the ratios of the other scenario.

The trade-off distances percentages of the NAS benchmarks over both scenarios are presented in Figure 3.20. These trade-off distances between energy consumption reduction and performance are used to verify which scenario is the best in both terms at the same time. The figure shows that the trade-off distance percentages are on average bigger over the multi-core scenario (17.6%) than over the one core per node scenario (15.3%).

3.6.3/ EXPERIMENTS WITH DIFFERENT STATIC POWER SCENARIOS

In Section 3.6, since it was not possible to measure the static power consumed by a CPU, the static power was assumed to be equal to 20% of the measured dynamic power. This power is consumed during the whole execution time, during computation and communication times. Therefore, when the DVFS operations are applied by the scaling algorithm and the CPUs' frequencies lowered, the execution time might increase and consequently the consumed static energy will be increased too.

The aim of this section is to evaluate the scaling algorithm while assuming different

values of static powers. In addition to the previously used percentage of static power, two new static power ratios, 10% and 30% of the measured dynamic power of the core, are used in this section. The experiments have been executed with these two new static power scenarios over the one site one core per node scenario. In these experiments, the class D of the NAS parallel benchmarks were executed over the Nancy site. 16 computing nodes from the three clusters, Graphite, Graphene and Griffon, were used in this experiment.

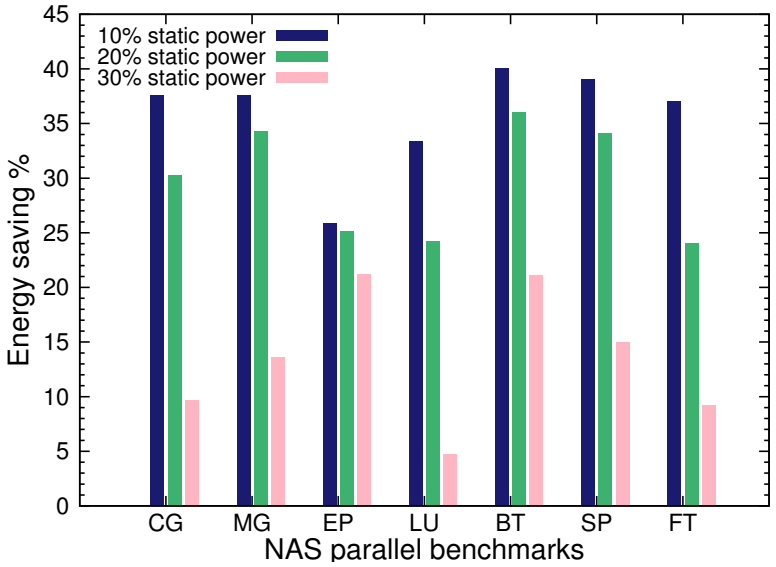

Figure 3.21: The energy saving percentages for the nodes executing the NAS benchmarks over the three power scenarios

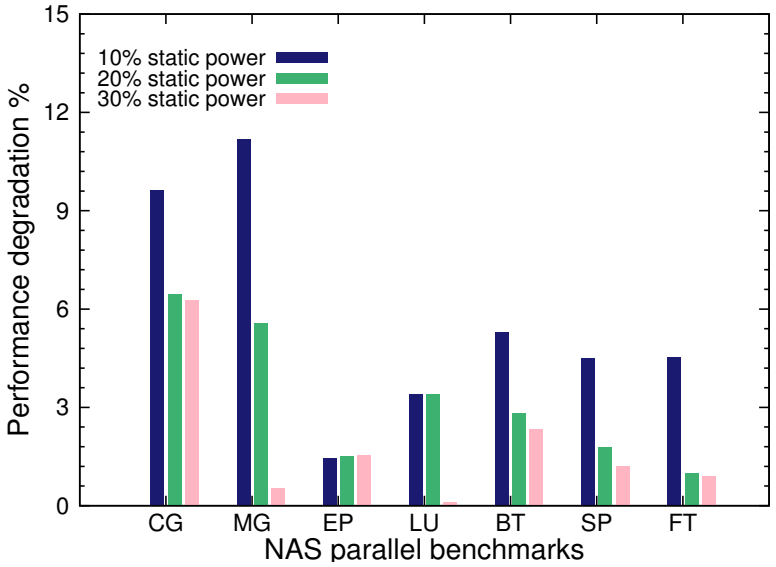

Figure 3.22: The performance degradation percentages for the NAS benchmarks over the three power scenarios

The energy saving percentages of the NAS benchmarks with the three static power scenarios are presented in Figure 3.21. This figure shows that the 10% of static power

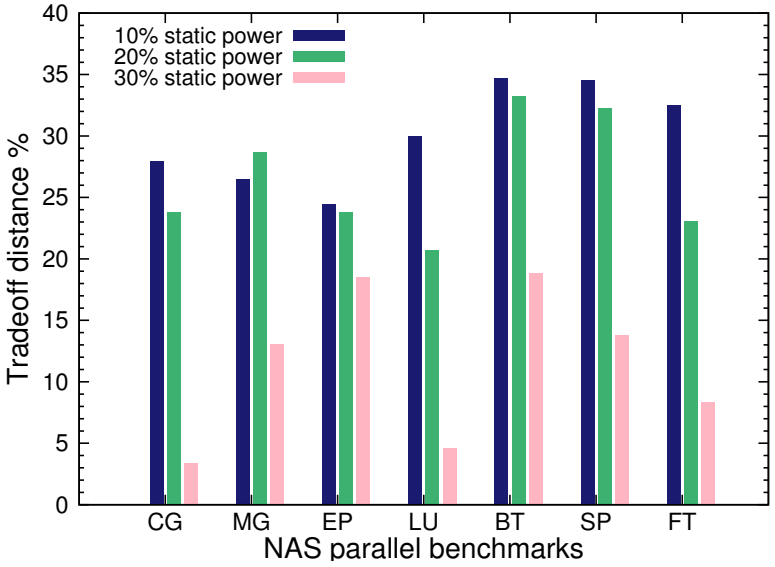

Figure 3.23: The trade-off distance percentages between the energy reduction and the performance of the NAS benchmarks over the three power scenarios

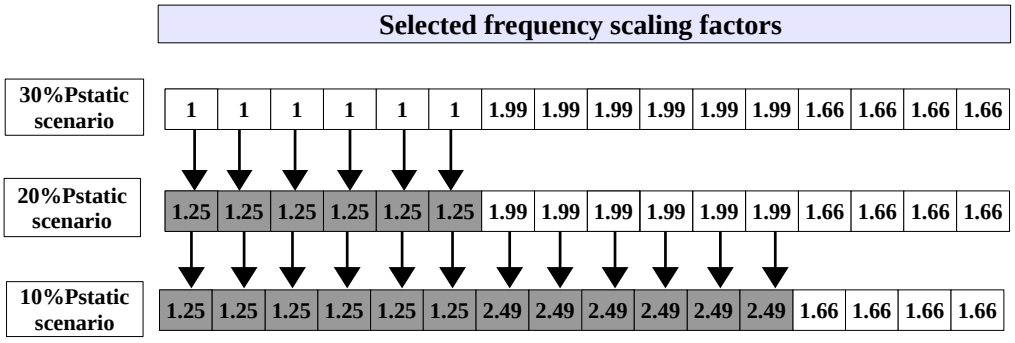

Figure 3.24: Comparing the selected frequency scaling factors for the MG benchmark over the three static power scenarios

scenario gives the biggest energy saving percentages in comparison to the 20% and 30% static power scenarios. The small value of the static power consumption makes the proposed scaling algorithm select smaller frequencies for the CPUs. These smaller frequencies reduce the dynamic energy consumption more than increasing the consumed static energy which gives less overall energy consumption. The energy saving percentages of the 30% static power scenario is the smallest between the other scenarios, because the scaling algorithm selects bigger frequencies for the CPUs which increases the energy consumption. Figure 3.24 demonstrates that the proposed scaling algorithm selects the best frequency scaling factors according to the static power consumption ratio being used.

The performance degradation percentages are presented in Figure 3.22. The 30% static power scenario had less performance degradation percentage because the scaling algorithm had selected big frequencies for the CPUs. While, the inverse happens in the 10% and 20% scenarios because the scaling algorithm had selected CPUs' frequencies smaller than those of the 30% scenario. The trade-off distance percentage for the

NAS benchmarks with these three static power scenarios are presented in Figure 3.23. It shows that the best trade-off distance percentage is obtained with the 10% static power scenario and this percentage is decreased for the other two scenarios because the scaling algorithm had selected different frequencies according to the static power values.

In the EP benchmark, the energy saving, performance degradation and trade-off distance percentages for these static power scenarios are not significantly different because there is no communication in this benchmark. Therefore, the static power is only consumed during computation and the proposed scaling algorithm selects similar frequencies for the three scenarios. On the other hand, for the rest of the benchmarks, the scaling algorithm selects the values of the frequencies according to the communication times of each benchmark because the static energy consumption increases proportionally to the communication times.

3.6.4/ COMPARISON BETWEEN THE PROPOSED FREQUENCIES SELECTING ALGORITHM AND THE EDP METHOD

Finding the frequencies that give the best trade-off between the energy consumption and the performance for a parallel application is not a trivial task. Many algorithms have been proposed to tackle this problem. In this section, the proposed frequencies selecting algorithm is compared to a method that uses the well known energy and delay product objective function, $EDP = energy \times delay$, that has been used by many researchers [72, 20, 56]. This objective function was also used by Spiliopoulos et al. algorithm [75] where they select the frequencies that minimize the EDP product and apply them with DVFS operations to the multi-core architecture. Their online algorithm predicts the energy consumption and execution time of a processor before using the EDP method.

To fairly compare the proposed frequencies scaling algorithm to Spiliopoulos et al. algorithm, called Maxdist and EDP respectively, both algorithms use the same energy model, Equation 3.9 and execution time model, Equation 3.8, to predict the energy consumption and the execution time for each computing node. Moreover, both algorithms start the search space from the upper bound computed as in Equation 3.7. Finally, the resulting EDP algorithm is an exhaustive search algorithm that tests all the possible frequencies, starting from the initial frequencies (upper bound), and selects the vector of frequencies that minimize the EDP product. Both algorithms were applied to the class D of the NAS benchmarks running over 16 nodes. The participating computing nodes are distributed according to the two scenarios described in Section 3.6.1. The experimental results, the energy saving, performance degradation and trade-off distance percentages, are presented in Figures 3.25, 3.26 and 3.27 respectively.

As shown in these figures, the proposed frequencies selection algorithm, Maxdist, outperforms the EDP algorithm in terms of energy consumption reduction and performance for all of the benchmarks executed over the two scenarios. The proposed algorithm gives better results than the EDP method because it maximizes the energy saving and the performance at the same time. Moreover, the proposed scaling algorithm gives the same weight for these two metrics. Whereas, the EDP algorithm gives sometimes negative trade-off values for some benchmarks in the two sites scenarios. These negative trade-off values mean that the performance degradation percentage is higher than the energy saving percentage. The high positive values of the trade-off distance percentage mean that the energy saving percentage is much higher than the performance degradation per-

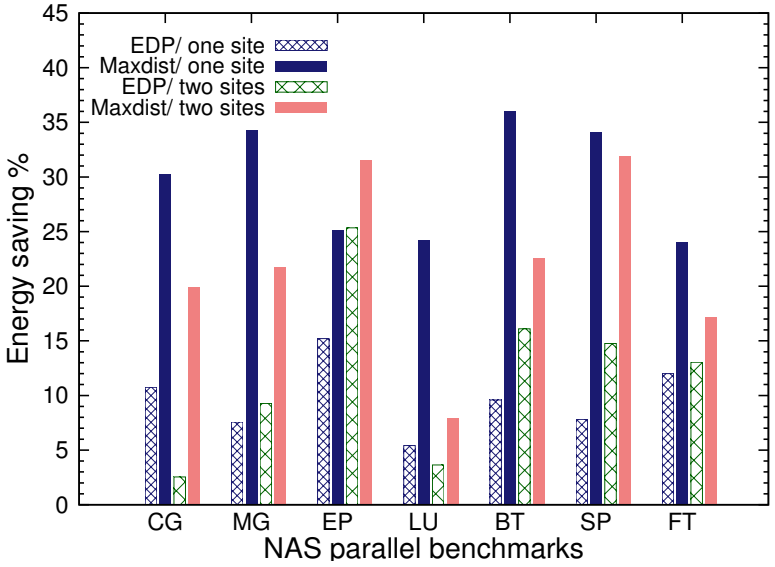

Figure 3.25: The energy reduction percentages induced by the Maxdist method and the EDP method

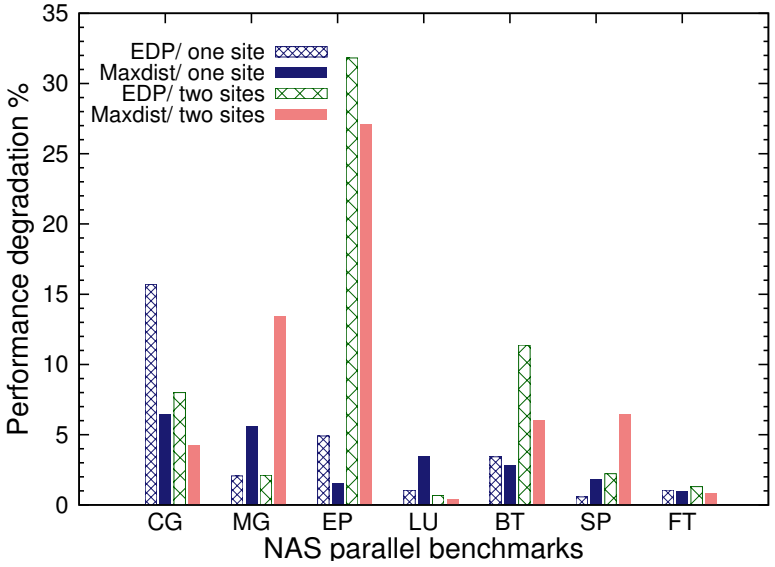

Figure 3.26: The performance degradation percentages induced by the Maxdist method and the EDP method

centage. The complexity of both algorithms, Maxdist and EDP, are of order $O(N \cdot M_i \cdot F_j)$ and $O(N \cdot M_i \cdot F_j^2)$ respectively, where N is the number of the clusters, M_i is the number of nodes and F_j is the maximum number of available frequencies of node j. When Maxdist is applied to a benchmark that is being executed over 32 nodes distributed between Nancy and Lyon sites, it takes on average $0.01\ ms$ to compute the best frequencies while the EDP method is on average ten times slower over the same architecture.

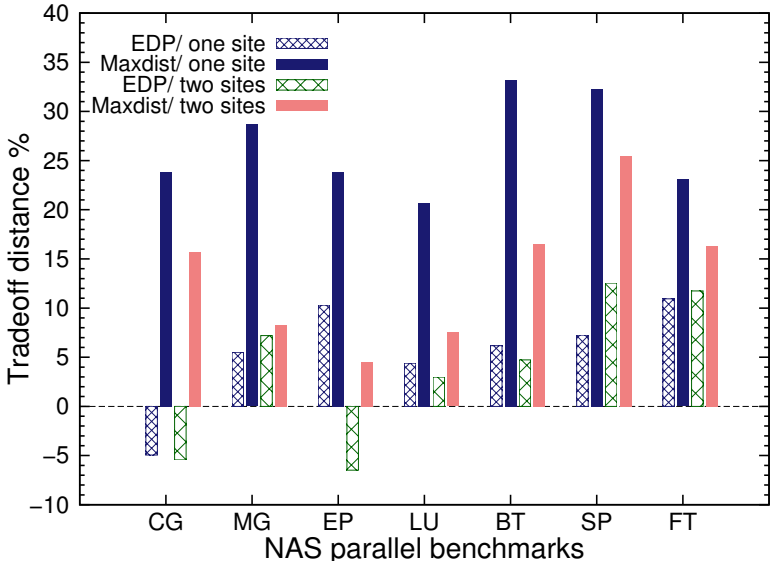

Figure 3.27: The trade-off distance percentages between the energy consumption reduction and the performance for the Maxdist method and the EDP method

3.7/ CONCLUSION

In this chapter, two new online frequency scaling factors selecting algorithms have been presented. They select the best possible vectors of frequency scaling factors that give the maximum distance (optimal trade-off) between the predicted energy and the predicted performance curves for a heterogeneous cluster and grid. Both algorithms use a new energy models for measuring and predicting the energy consumption of message passing applications with iterations running over a heterogeneous local cluster and a grid platform. Firstly, the proposed scaling factors selection algorithm for a heterogeneous local cluster is applied to the class C of the NAS parallel benchmarks and simulated by SimGrid. The results of the simulations showed that the algorithm on average reduces by 29.8% the energy consumption of the NAS benchmarks executed over 8 nodes while limiting the degradation of the performance by 3.8%. The algorithm also selects different scaling factors according to the percentage of the computing and communication times, and according to the values of the static and dynamic powers of the CPUs. Secondly, the proposed scaling factors selection algorithm for a grid is applied to the class D of the NAS parallel benchmarks and executed over the Grid5000 testbed platform. The experiments executed on 16 nodes distributed over three clusters, showed that the algorithm on average reduces by 30% the energy consumption of all the NAS benchmarks while on average only degrading by 3.2% their performance. The algorithm was also evaluated in different scenarios that vary in the distribution of the computing nodes between different clusters' sites or use multi-core per node architecture or consume different static power values. The algorithm selects different vectors of frequencies according to the computations and communication times ratios, and the values of the static and measured dynamic powers of the CPUs. Thus, the simulation and the real results are comparable in term of energy saving and performance degradation percentages. Finally, both algorithms were compared to a method that uses the well known energy and delay product as an objective function. The comparison results showed that the proposed algorithms outperform the latter by selecting vectors of frequencies that give a better trade-off results.

4

Energy Optimization of Asynchronous Applications

4.1/ Introduction

A grid is composed of heterogeneous clusters: CPUs from distinct clusters might have different computing power, energy consumption or frequency range. Running synchronous parallel applications on grids results in long slack times where the fast nodes have to wait for the slower ones to finish their computations before synchronously exchanging data with them. Therefore, it is widely accepted that asynchronous parallel methods are more suitable than synchronous ones for such architectures because there is no slack time and the asynchronous communications are overlapped by computations. However, they usually execute more iterations than the synchronous ones and thus consume more energy. In order to make the asynchronous method a good alternative to the synchronous one, it should not be just competitive in performance but also in energy consumption. To reduce the energy consumption of a CPU executing the asynchronous iterative method, the Dynamic voltage and frequency scaling (DVFS) technique can be used. Modern operating systems automatically adjust the frequency of the processor according to their needs using DVFS operations. However, the user can scale down the frequency of the CPU using the on-demand governor [82]. It lowers the frequency of a CPU to reduce its energy consumption, but it also decreases its computing power and thus it might increase the execution time of an application running on that processor. Therefore, the frequency that gives the best trade-off between energy consumption and performance must be selected. For parallel asynchronous methods running over a grid, a different frequency might be selected for each CPU in the grid depending on its characteristics. In chapters 2 and 3, three frequency selecting algorithms were proposed to reduce the energy consumption of synchronous message passing applications with iterations running over homogeneous or heterogeneous platforms. In this chapter, a new frequency selecting algorithm for asynchronous iterative message passing applications running over grids is presented. An adaptation for hybrid methods, with synchronous and asynchronous communications, is also proposed. The algorithm and its adaptation select the vector of frequencies which simultaneously offers a maximum energy reduction and minimum performance degradation ratio. The algorithm has a very small overhead and works online without needing any training nor any profiling.

This chapter is organized as follows: Section 4.2 presents some related works from other authors. Models for predicting the performance and the energy consumption of

both synchronous and asynchronous iterative message passing programs executed over grids are explained in Section 4.3. It also presents the objective function that maximizes the reduction of energy consumption while minimizing the degradation of the program's performance, used to select the frequencies. Section 4.4 details the proposed frequency selecting algorithm. Section 4.5 presents the iterative multi-splitting application which is a hybrid method and was used as a benchmark to evaluate the efficiency of the proposed algorithm. Section 4.6 presents the simulation results of applying the algorithm on the multi-splitting application and executing it on different grid scenarios. It also shows the results of running three different power scenarios and comparing them. Moreover, in the last subsection, the proposed algorithm is compared to the energy and delay product (EDP) method. Section 4.7 presents the results of real experiments executed over the Grid'5000 platform and compared to the EDP method. Finally, the chapter ends with a summary.

4.2/ RELATED WORKS

A message passing application is in general composed of two types of sections, which are the computations and the communications sections. The communications can be done synchronously or asynchronously. In a synchronous message passing application, when a process synchronously sends a message to another node, it is blocked until the latter receives the message. During that time, there is no computation on both nodes and that period is called slack time. On the contrary, in an asynchronous message passing application, the asynchronous communications are overlapped by computations, thus, there is no slack time. Many techniques have been used to reduce the energy consumption of message passing applications, such as scheduling, heuristics and DVFS. For example, different scheduling techniques, to switch off the idle nodes to save their energy consumption, were presented in [77, 28, 59] and [29]. In [23] and [14], an heuristic to manage the workloads between the computing resources of the cluster and reduce their energy, was published. However, the dynamic voltage and frequency scaling (DVFS) is the most popular technique to reduce the energy consumption of computing processors.

As shown in the related works of chapter 2, most of the works in this field targeted the synchronous message passing applications because they are more common than the asynchronous ones and easier to work on. Some researchers tried to reduce slack times in synchronous applications running over homogeneous clusters. These slack times can happen on such architectures if the distributed workloads over the computing nodes are imbalanced. Other works focused on reducing the energy consumption of synchronous applications running over heterogeneous architectures such as heterogeneous clusters or grids. When executing synchronous message passing applications on these architectures, slack times are generated when fast nodes have to communicate with slower ones. Indeed, the fast nodes have to wait for the slower ones to finish their computations to be able to communicate with them. In this case, some energy was saved as in the work of chapter 3 and its related works by reducing the frequencies of the fast nodes with DVFS operations while minimizing the slack times.

Whereas, no work has been conducted to optimize the energy consumption of asynchronous message passing applications. Some works use asynchronous communications when applying DVFS operations on synchronous applications. For example, Hsu et al. [41] proposed an online adaptive algorithm that divides the synchronous message

passing application into several time periods and selects the suitable frequency for each one. The algorithm asynchronously applies the new computed frequencies to overlap the multiple DVFS switching times with computation. Similarly to this work, Zhu et al. [87] studied the difference between applying synchronously or asynchronously the frequency changing algorithm during the execution time of the program. The results of the proposed asynchronous scheduler were more energy efficient than synchronous one. In [79], Vishnu et al. presented an energy efficient asynchronous agent that reduces the slack times in a parallel program to reduce the energy consumption. They used asynchronous communications in the proposed algorithm, which calls the DVFS algorithm many times during the execution time of the program. The three previous presented works were applied on applications running over homogeneous platforms.

In [10], the energy consumption of an asynchronous iterative linear solver running over a heterogeneous platform, is evaluated. The results showed that the asynchronous version of the application had less execution time than the synchronous one. Therefore, according to their energy model the asynchronous method consumes less energy. However, in their model they do not consider that during synchronous communications only static power which is significantly lower than dynamic power, is consumed.

This chapter presents the following contributions:

1. new model to predict the energy consumption and the execution time of asynchronous iterative message passing applications running over a grid platform.

2. a new online algorithm that selects a vector of frequencies which gives the best trade-off between energy consumption and performance for asynchronous iterative message passing applications running over a grid platform. The algorithm has a very small overhead and does not need any training or profiling. The new algorithm can be applied synchronously and asynchronously on an iterative message passing application.

4.3/ THE PERFORMANCE AND THE ENERGY CONSUMPTION MEASUREMENT MODELS

4.3.1/ THE EXECUTION TIME OF ITERATIVE ASYNCHRONOUS MESSAGE PASSING APPLICATIONS

In this chapter, we are interested in running asynchronous iterative message passing distributed applications over a grid while reducing the energy consumption of the CPUs during the execution. Figure 4.1 is an example of a grid with four different clusters. Inside each cluster, all the nodes are homogeneous, have the same specifications, but are different from the nodes of the other clusters. To reduce the energy consumption of these applications while running on a grid, the heterogeneity of the clusters' nodes, such as nodes' computing powers (FLOPS), energy consumptions and CPU's frequency ranges, must be taken into account. To reduce the complexity of the experiments and focus on the heterogeneity of the nodes, the local networks of all the clusters are assumed to be identical, with the same latency and bandwidth. The networks connecting the clusters are also assumed to be homogeneous but they are slower than the local networks.

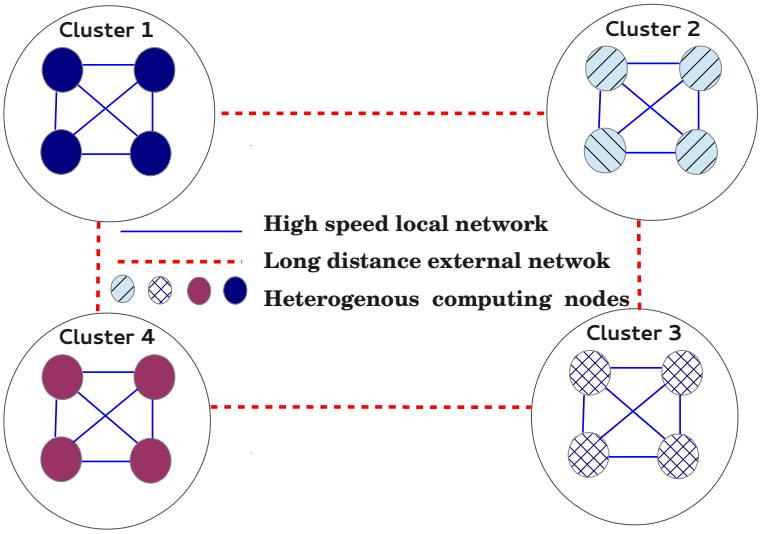

Figure 4.1: A grid platform composed of heterogeneous clusters

An iterative application consists of a block of instructions that is repeatedly executed until convergence. A distributed iterative application with interdependent tasks requires, at each iteration, exchanging data between nodes to compute the distributed tasks. The communications between the nodes can be done synchronously or asynchronously. In the synchronous model, each node has to wait to receive data from all its neighbours to compute its iteration, see figures 1.14 and 1.15. Since the tasks are synchronized, all the nodes execute the same number of iterations. Then, The overall execution time of an iterative synchronous message passing application with balanced tasks, running on the grid described above, is equal to the execution time of the slowest node in the slowest cluster running a task as presented in 3.1.

Whereas, in the asynchronous model, the fast nodes do not have to wait for the slower nodes to finish their computations to exchange data, see Figure 1.16. Therefore, there are no idle times between successive iterations, the node executes the computations with the last received data from its neighbours and the communications are overlapped by computations. Since there are no synchronizations between nodes, all nodes do not have the same number of iterations. The difference in the number of executed iterations between the nodes depends on the heterogeneity of the computing powers of the nodes. The execution time of an asynchronous iterative message passing application is not equal to the execution time of the slowest node like in the synchronous mode because each node executes a different number of iterations. Moreover, the overall execution time is directly dependent on the method used to detect the global convergence of the asynchronous iterative application. The global convergence detection method might be synchronous or asynchronous and centralized or distributed.

In a grid, the nodes in each cluster have different characteristics, especially different frequency gears. Therefore, when applying DVFS operations on these nodes, they may get different scaling factors represented by a scaling vector: $(S_{11}, S_{12}, \ldots, S_{NM_i})$ where S_{ij} is the scaling factor of processor j in the cluster i. To be able to predict the execution time of asynchronous iterative message passing applications running over a grid, for different vectors of scaling factors, the communication times and the computation times for all the tasks must be measured during the first iteration before applying any DVFS operation.

Then, the execution time of one iteration of an asynchronous iterative message passing application, running on a grid after applying a vector of scaling factors, is equal to the execution time of the synchronous application but without its communication times. The communication times are overlapped by computations and the execution time can be evaluated for all the application as the average of the execution time of all the parallel tasks. This is presented in Equation 4.1.

$$T_{New} = \frac{\sum_{i=1}^{N} \sum_{j=1}^{M_i} (T_{cpOld_{ij}} \cdot S_{ij})}{N \cdot M_i} \tag{4.1}$$

In this work, a hybrid (synchronous/asynchronous) message passing application [65] is being used. It is composed of two loops:

1. In the inner loop, at each iteration, the nodes in a cluster synchronously exchange data between them. There is no communication between nodes from different clusters.
2. In the outer loop, at each iteration, the nodes from different clusters asynchronously exchange their data between them because the network interconnecting the clusters has a high latency.

Therefore, the execution time of one outer iteration of such a hybrid application can be evaluated by computing the average of the execution time of the slowest node in each cluster. The overall execution time of the asynchronous iterative applications can be evaluated as follows:

$$T_{New} = \frac{\sum_{i=1}^{N} (\max_{j=1,\ldots,M_i} (T_{cpOld_{ij}} \cdot S_{ij}) + \min_{j=1,\ldots,M_i} (L_{tcm_{ij}}))}{N} \tag{4.2}$$

In Equation (4.2), the communication times $L_{tcm_{ij}}$ are only the communications between the local nodes because the communications between the clusters are asynchronous and overlapped by computations.

4.3.2/ THE ENERGY MODEL AND TRADE-OFF OPTIMIZATION

The energy consumption of an asynchronous application running over a heterogeneous grid is the summation of the dynamic and static power of each node multiplied by the computation time of that node as in Equation (4.3). The computation time of each node is equal to the overall execution time of the node because the asynchronous communications are overlapped by computations.

$$E = \sum_{i=1}^{N} \sum_{j=1}^{M_i} (S_{ij}^{-2} \cdot T_{cp_{ij}} \cdot (P_{d_{ij}} + P_{s_{ij}})) \tag{4.3}$$

It is common for distributed algorithms running over grids to have asynchronous external communications between clusters and synchronous ones between the nodes of the same cluster. In this hybrid communication scheme, the dynamic energy consumption can be computed in the same way as for the synchronous application with Equation

(2.11). However, since the nodes of different clusters are not synchronized and do not have the same execution time as in the synchronous application, the static energy consumption is different between them. The cluster execution time is equal to the execution time of the slowest task in that cluster. The energy consumption of the asynchronous iterative message passing application running on a heterogeneous grid platform during one iteration can be computed as follows:

$$E = \sum_{i=1}^{N}\sum_{j=1}^{M_i}(S_{ij}^{-2}\cdot P_{d_{ij}}\cdot T_{cp_{ij}}) + \sum_{i=1}^{N}\sum_{j=1}^{M_i}(P_{s_{ij}}\cdot(\max_{j=1,\ldots,M_i}(T_{cp_{ij}}\cdot S_{ij}) + \min_{j=1,\ldots,M_i}(L_{tcm_{ij}}))) \quad (4.4)$$

Where $L_{tcm_{ij}}$ is the local communication time of the cluster i of node j. Reducing the frequencies of the processors according to the vector of scaling factors $(S_{11}, S_{12}, \ldots, S_{NM_i})$ may degrade the performance of the application and thus, increase the static energy consumed because the execution time is increased [44]. The overall energy consumption for the asynchronous application can be computed by multiplying the energy consumption from one iteration of each cluster by the number of the iterations of that cluster, N_{iter_i}, as in Equation (4.5).

$$E = \sum_{i=1}^{N}(\sum_{j=1}^{M_i}(S_{ij}^{-2}\cdot P_{d_{ij}}\cdot T_{cp_{ij}}))\cdot N_{iter_i} + \sum_{i=1}^{N}(\sum_{j=1}^{M_i}(P_{s_{ij}}\cdot (\max_{j=1,\ldots,M_i}(T_{cp_{ij}}\cdot S_{ij}) + \min_{j=1,\ldots,M_i}(L_{tcm_{ij}})))\cdot N_{iter_i} \quad (4.5)$$

In order to optimize the energy consumption and the performance of the asynchronous iterative applications at the same time, the maximum distance between the two metrics can be computed as in the previous chapters. However, both the energy model and performance must be normalized as in the Equations 3.3 and 3.4 respectively. Hence, T_{New} should be computed as in Equation 4.2 and T_{Old} computed as follows:

$$T_{Old} = \frac{\sum_{i=1}^{N}(\max_{j=1,\ldots,M_i}(T_{cpOld_{ij}}) + \min_{j=1,\ldots,M_i}(L_{tcm_{ij}}))}{N} \quad (4.6)$$

The original energy consumption of asynchronous applications, $E_{Original}$ is computed as in (4.7).

$$E_{original} = \sum_{i=1}^{N}\sum_{j=1}^{M_i}(P_{d_{ij}}\cdot T_{cpOld_{ij}}) + \sum_{i=1}^{N}\sum_{j=1}^{M_i}(P_{s_{ij}}\cdot(\max_{j=1,\ldots,M_i}(T_{cpOld_{ij}}) + \min_{j=1,\ldots,M_i}(L_{tcm_{ij}}))) \quad (4.7)$$

Then, the objective function can be modeled as the maximum distance between the normalized energy curve and the normalized performance curve over all available sets of scaling factors and is computed as in the objective function 3.12.

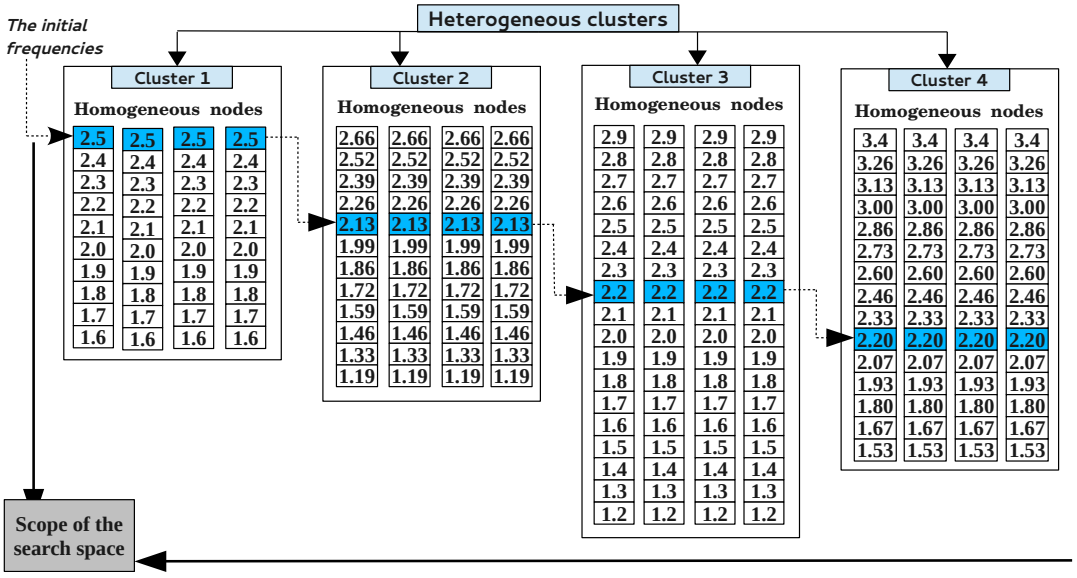

Figure 4.2: Selecting the initial frequencies in a grid composed of four clusters

4.4/ THE SCALING FACTORS SELECTION ALGORITHM OF ASYNCHRONOUS APPLICATIONS OVER GRID

The frequency selection algorithm (8) works online during the first iteration of asynchronous iterative message passing program running over a grid. The algorithm selects the set of frequency scaling factors $S_{opt_{11}}, S_{opt_{12}}, \ldots, S_{opt_{NM_i}}$ which maximizes the distance, the tradeoff function (3.12), between the predicted normalized energy consumption and the normalized performance of the program. The algorithm is called just once in the iterative program and it uses information gathered from the first iteration to approximate the vector of frequency scaling factors that gives the best tradeoff. According to the returned vector of scaling factors, the DVFS algorithm (6) computes the new frequency for each node in the grid. It also shows where and when the proposed scaling algorithm is called in the iterative message passing program.

In contrast to the scaling factors selection algorithm of synchronous applications running on the grid (algorithm 7), this algorithm computed the initial frequencies depending on the Equations 3.13 and 3.14. Figure 4.2 shows the selected initial frequencies of the grid composed of four different types of clusters that are presented in the Figure 4.1. The only difference between the two algorithms is the energy and performance models that are used. Furthermore, this algorithm scales down all frequencies of nodes at each iteration, while other algorithm don't scaled down the frequency of the slowest node. However, the performance of asynchronous application does not depend on the performance of the slower nodes, while it depends on the performance of all nodes.

Algorithm 8 Scaling factors selection algorithm of asynchronous applications over grid

Require:

N number of clusters in the grid.

M number of nodes in each cluster.

$T_{cp_{ij}}$ array of all computation times for all nodes during one iteration and with the highest frequency.

$T_{cm_{ij}}$ array of all communication times for all nodes during one iteration and with the highest frequency.

$F_{max_{ij}}$ array of the maximum frequencies for all nodes.

$P_{d_{ij}}$ array of the dynamic powers for all nodes.

$P_{s_{ij}}$ array of the static powers for all nodes.

$F_{diff_{ij}}$ array of the differences between two successive frequencies for all nodes.

Ensure: $S_{opt_{11}}, S_{opt_{12}} \ldots, S_{opt_{NM_i}}$, a vector of scaling factors that gives the optimal tradeoff between energy consumption and execution time

1: $S_{cp_{ij}} \leftarrow \frac{\max_{i=1,2,\ldots,N}(\max_{j=1,2,\ldots,M_i}(T_{cp_{ij}}))}{T_{cp_{ij}}}$

2: $F_{ij} \leftarrow \frac{F_{max_{ij}}}{S_{cp_i}},\ i = 1, 2, \cdots, N,\ j = 1, 2, \ldots, M_i.$

3: Round the computed initial frequencies F_i to the closest available frequency for each node.

4: **if** (not the first frequency) **then**

5: $\quad F_{ij} \leftarrow F_{ij} + F_{diff_{ij}},\ i = 1, \ldots, N,\ j = 1, \ldots, M_i.$

6: **end if**

7: $T_{Old} \leftarrow \frac{\sum_{i=1}^{N}(\max_{j=1,\ldots,M_i}(T_{cpOld_{ij}}) + \min_{j=1,\ldots,M_i}(L_{tcm_{ij}}))}{N}$

8: $E_{Original} \leftarrow \sum_{i=1}^{N}\sum_{j=1}^{M_i}(P_{d_{ij}} \cdot T_{cpOld_{ij}}) + \sum_{i=1}^{N}\sum_{j=1}^{M_i}(P_{s_{ij}} \cdot (\max_{j=1,\ldots,M_i}(T_{cpOld_{ij}}) + \min_{j=1,\ldots,M_i}(L_{tcm_{ij}})))$

9: $S_{opt_{ij}} \leftarrow 1,\ i = 1, \ldots, N,\ j = 1, \ldots, M_i.$

10: $Dist \leftarrow 0$

11: **while** (all nodes have not reached their minimum frequency **or** $P_{Norm} - E_{Norm} < 0$) **do**

12: $\quad$ **if** (not the last frequency) **then**

13: $\quad\quad F_{ij} \leftarrow F_{ij} - F_{diff_{ij}},\ i = 1, \ldots, N,\ j = 1, \ldots, M_i.$

14: $\quad\quad S_{ij} \leftarrow \frac{F_{max_{ij}}}{F_{ij}},\ i = 1, \ldots, N,\ j = 1, \ldots, M_i.$

15: $\quad$ **end if**

16: $\quad T_{New} \leftarrow \frac{\sum_{i=1}^{N}(\max_{j=1,\ldots,M_i}(T_{cpOld_{ij}} \cdot S_{ij}) + \min_{j=1,\ldots,M_i}(L_{tcm_{ij}}))}{N}$

17: $\quad E_{Reduced} \leftarrow \sum_{i=1}^{N}\sum_{j=1}^{M_i}(S_{ij}^{-2} \cdot P_{d_{ij}} \cdot T_{cp_{ij}}) + \sum_{i=1}^{N}\sum_{j=1}^{M_i}(P_{s_{ij}} \cdot (\max_{j=1,\ldots,M_i}(T_{cp_{ij}} \cdot S_{ij}) + \min_{j=1,\ldots,M_i}(L_{tcm_{ij}})))$

18: $\quad P_{Norm} \leftarrow \frac{T_{Old}}{T_{New}}$

19: $\quad E_{Norm} \leftarrow \frac{E_{Reduced}}{E_{Original}}$

20: $\quad$ **if** $(P_{Norm} - E_{Norm} > Dist)$ **then**

21: $\quad\quad S_{opt_{ij}} \leftarrow S_{ij},\ i = 1, \ldots, N,\ j = 1, \ldots, M_i.$

22: $\quad\quad Dist \leftarrow P_{Norm} - E_{Norm}$

23: $\quad$ **end if**

24: **end while**

25: Return $S_{opt_{11}}, S_{opt_{12}}, \ldots, S_{opt_{NM_i}}$

Table 4.1: The characteristics of the four types of nodes

node type	Simulated GFLOPS of one node	Max Freq. GHz	Min Freq. GHz	Diff. Freq. GHz	Dynamic power	Static power
A	40	2.50	1.20	0.100	20 W	4 W
B	50	2.66	1.60	0.133	25 W	5 W
C	60	2.90	1.20	0.100	30 W	6 W
D	70	3.40	1.60	0.133	35 W	7 W

4.5/ THE ITERATIVE MULTI-SPLITTING METHOD

Multi-splitting algorithms have been initially studied to solve linear systems of equations in parallel [58]. Thereafter, they were used to design non linear iterative algorithms and asynchronous iterative algorithms [12]. The principle of multi-splitting algorithms lies in splitting the system of equations, then solving each sub-system using a direct or an iterative method and then combining the results in order to build a global solution. Since a multi-splitting method is iterative, it requires executing several iterations in order to reach global convergence.

In this chapter, we have used an asynchronous iterative multisplitting method to solve a 3D Poisson problem as described in [65]. The problem is divided into small 3D subproblems and each one is solved by a parallel GMRES method. For more information about multi-splitting algorithms, interested readers are invited to consult the previous references.

4.6/ THE EXPERIMENTAL RESULTS OVER SIMGRID

In this section, the heterogeneous scaling algorithm (HSA), Algorithm (8), is applied to the parallel iterative multi-splitting method. The performance of this algorithm is evaluated by executing the iterative multi-splitting method on the Simgrid/SMPI simulator v3.10 [18]. This simulator offers flexible tools to create a grid architecture and run the iterative application over it. The grid used in these experiments has four different types of nodes. Two types of nodes have different computing powers, frequency ranges, static and dynamic powers. Table 4.1 presents the characteristics of the four types of nodes. The specifications of the simulated nodes are similar to real Intel processors. Many grid configurations have been used in the experiments where the number of clusters and the number of nodes per cluster are equal to 4 or 8. For the grids composed of 8 clusters, two clusters of each type of nodes were used. The number of nodes per cluster is the same for all the clusters in a given grid.

The CPUs' constructors do not specify the amount of static and dynamic powers their CPUs consume. The maximum power consumption for each node's CPU was chosen to be proportional to its computing power (FLOPS). The dynamic power was assumed to represent 80 % of the overall power consumption and the rest (20 %) is the static power. Similar assumptions were made in last two chapters and [66]. The clusters of the grid are connected via a long distance Ethernet network with 1 Gbit/s bandwidth, while inside each cluster the nodes are connected via a high-speed 10 Gbit/s bandwidth local Ethernet

Table 4.2: The different experiment scenarios

Platform scenario	Clusters number	Number of nodes in cluster	Vector size	Total number of nodes in grid
Grid.4*4.400	4	4	400^3	16
Grid.4*8.400	4	8	400^3	32
Grid.8*4.400	8	4	400^3	32
Grid.8*8.400	8	8	400^3	64
Grid.4*4.500	4	4	500^3	16
Grid.4*8.500	4	8	500^3	32
Grid.8*4.500	8	4	500^3	32
Grid.8*8.500	8	8	500^3	64

network. The local networks have ten times less latency than the network connecting the clusters.

4.6.1/ THE ENERGY CONSUMPTION AND THE EXECUTION TIME OF THE MULTI-SPLITTING APPLICATION

The multi-splitting (MS) method solves a three dimensional problem of size $N = N_x \cdot N_y \cdot N_z$. The problem is divided into equal sub-problems which are distributed to the computing nodes of the grid and then solved. The experiments were conducted on problems of size $N = 400^3$ or $N = 500^3$ that require more than 12 and 24 Gigabyte of memory, respectively. Table 4.2 presents the different experiment scenarios with different numbers of clusters, nodes per cluster and problem sizes. A name, consisting in the values of these parameters was given to each scenario.

This section focuses on the execution time and the energy consumed by the MS application while running over the grid platform without using DVFS operations. The energy consumption of the synchronous and asynchronous MS was measured using the energy Equations 3.9 and 4.4 respectively. Figures 4.3 (a) and (b) show the energy consumption and the execution time, respectively, of the multi-splitting application running over a heterogeneous grid with different numbers of clusters and nodes per cluster. The synchronous and the asynchronous versions of the MS application were executed over each scenario in Table 4.2. As shown in Figure 4.3 (a), the asynchronous MS consumes more energy than the synchronous one. Indeed, the asynchronous application overlaps the asynchronous communications with computations and thus it executes more iterations than the synchronous one and has no slack times. More computations result in more dynamic energy consumption by the CPU in the asynchronous MS and since the dynamic power is chosen to be four times higher than the static power, the asynchronous MS method consumes more overall energy than the synchronous one. However, the execution times of the experiments, presented in Figure 4.3 (b), show that the execution times of the asynchronous MS are smaller than the execution times of the synchronous one. Indeed, in the asynchronous application the fast nodes do not have to wait for the slower ones to exchange data. So there are no slack times and more iterations are executed by fast nodes which accelerates the convergence to the final solution.

The synchronous and asynchronous MS scale well. The execution times of both methods decrease linearly with the increase of the number of computing nodes in the grid,

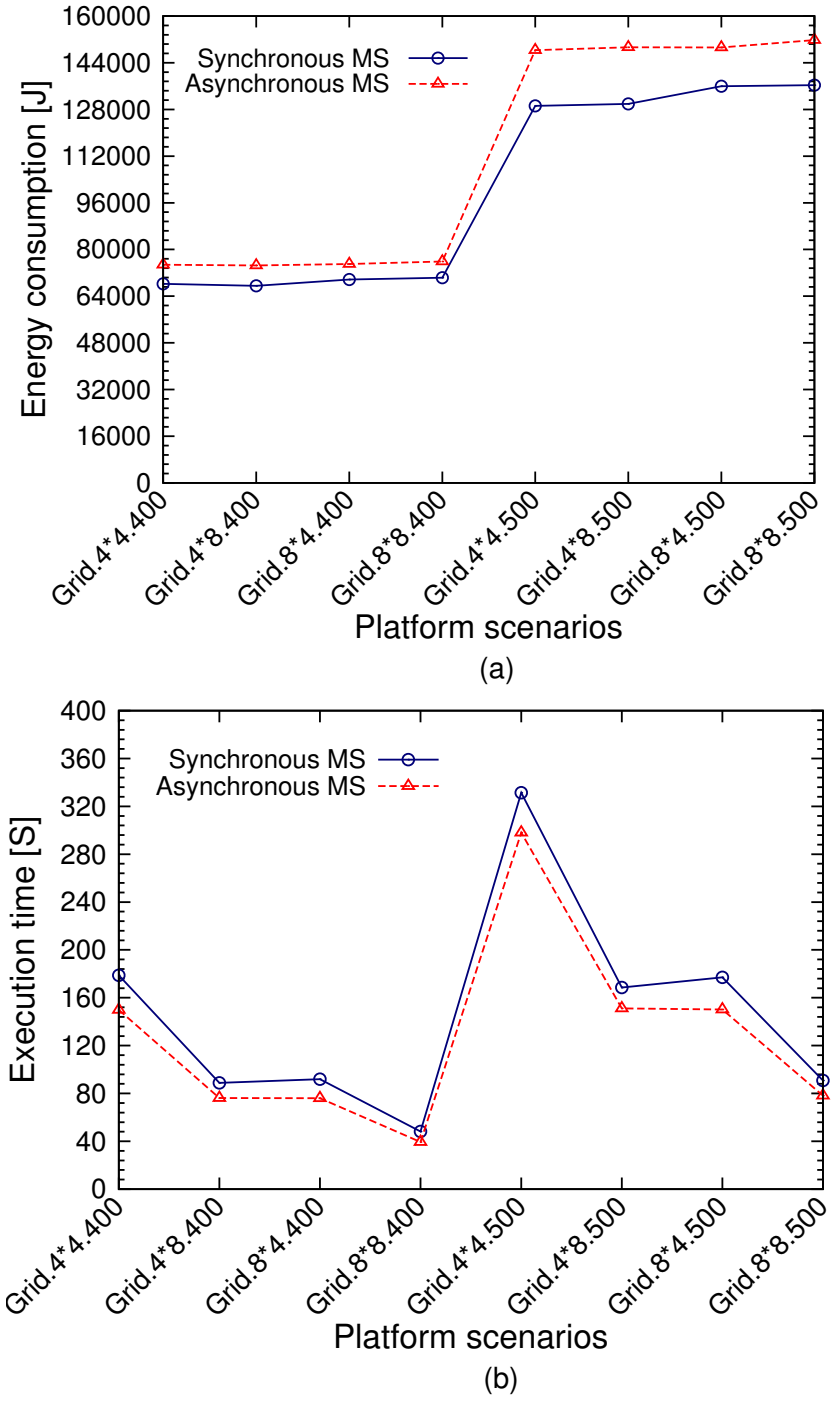

Figure 4.3: (a) energy consumption and (b) execution time of multi-splitting application without applying the HSA algorithm

whereas the energy consumption is approximately the same when the number of computing nodes increases. Therefore, the energy consumption of this application is not directly related to the number of computing nodes.

4.6.2/ THE RESULTS OF THE SCALING FACTOR SELECTION ALGORITHM

The scaling factor selection algorithm 8 was applied to both synchronous and asynchronous MS applications which were executed over the 8 possible scenarios presented in table 4.2. The DVFS algorithm 4 needs to send and receive some information before calling the scaling factor selection algorithm algorithm 8. The communications of the DVFS algorithm can be applied synchronously or asynchronously which results in four different versions of the application: synchronous or asynchronous MS with synchronous or asynchronous DVFS communications. Figures 4.4 (a) and (b) present the energy consumption and the execution time for the four different versions of the application running on all the scenarios in Table 4.2.

Figure 4.4 (a) shows that the energy consumption of all four versions of the method, running over the 8 grid scenarios described in Table 4.2, are not affected by the increase in the number of computing nodes. MS without applying DVFS operations had the same behaviour. On the other hand, Figure 4.4 (b) shows that the execution time of the MS application with DVFS operations decreases in inverse proportion to the number of nodes. Moreover, it can be noticed that the asynchronous MS with synchronous DVFS consumes less energy when compared to the other versions of the method. Two reasons explain this energy consumption reduction:

1. The asynchronous MS with synchronous DVFS version uses synchronous DVFS communications which allow it to apply the new computed frequencies at the beginning of the second iteration. Thus, reducing the consumption of dynamic energy by the application from the second iteration until the end of the application. Whereas in asynchronous DVFS versions where the DVFS communications are asynchronous, the new frequencies cannot be computed at the end of the first iteration and consequently cannot be applied at the beginning of the second iteration. Indeed, since the performance information gathered during the first iteration is not sent synchronously at the end of the first iteration, fast nodes might execute many iterations before receiving the performance information, computing the new frequencies based on this information and applying the new computed frequencies. Therefore, many iterations might be computed by CPUs running on their highest frequency and consuming more dynamic energy than scaled down processors.

2. As shown in Figure 4.3 (b), the execution time of the asynchronous MS version is lower than the execution time of the synchronous MS version because there is no idle time in the asynchronous version and the communications are overlapped by computations. Since the consumption of static energy is proportional to the execution time, the asynchronous MS version consumes less static energy than the synchronous version.

The energy saving percentage is the ratio between the reduced energy consumption after applying the HSA algorithm and the original energy consumption of synchronous MS without DVFS. Whereas, the performance degradation percentage is the ratio between the original execution time of the synchronous MS without DVFS and the new execution time after applying the HSA algorithm. Therefore, in this section, the synchronous MS method without DVFS serves as a reference for comparison with the other methods for the following terms: energy saving, performance degradation and the distance between the two previous terms.

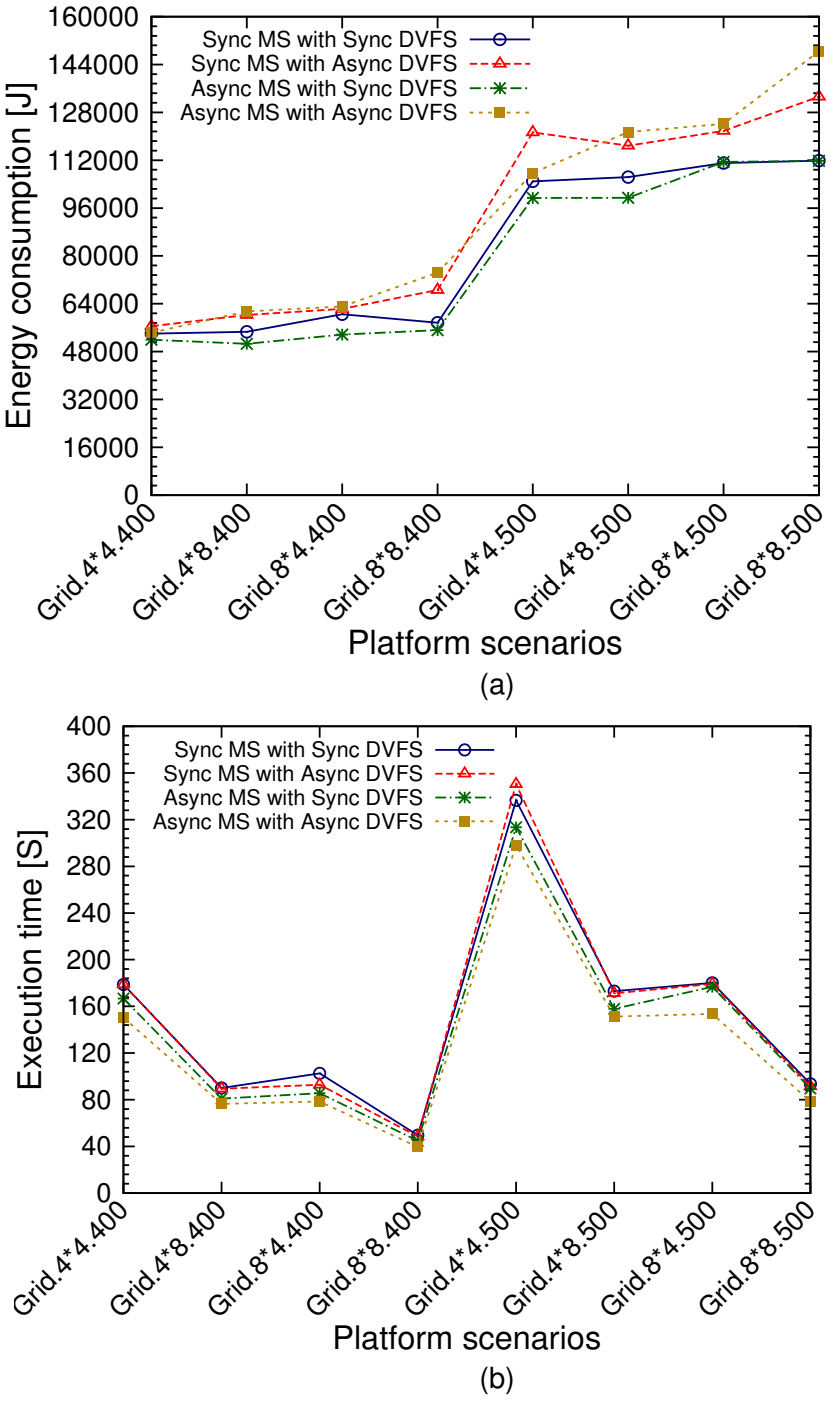

Figure 4.4: (a) energy consumption and (b) execution time of different versions of the multi-splitting application after applying the HSA algorithm

In Figure 4.5, the energy saving is computed for the four versions of the MS method which are the synchronous or asynchronous MS that apply synchronously or asynchronously the HSA algorithm. The fifth version is the asynchronous MS without any DVFS operations. Figure 4.5 shows that some versions have positive or negative energy saving percentages which means that the corresponding version respectively consumes less or more energy than the reference method. As in Figure 4.4 (a) and for the same

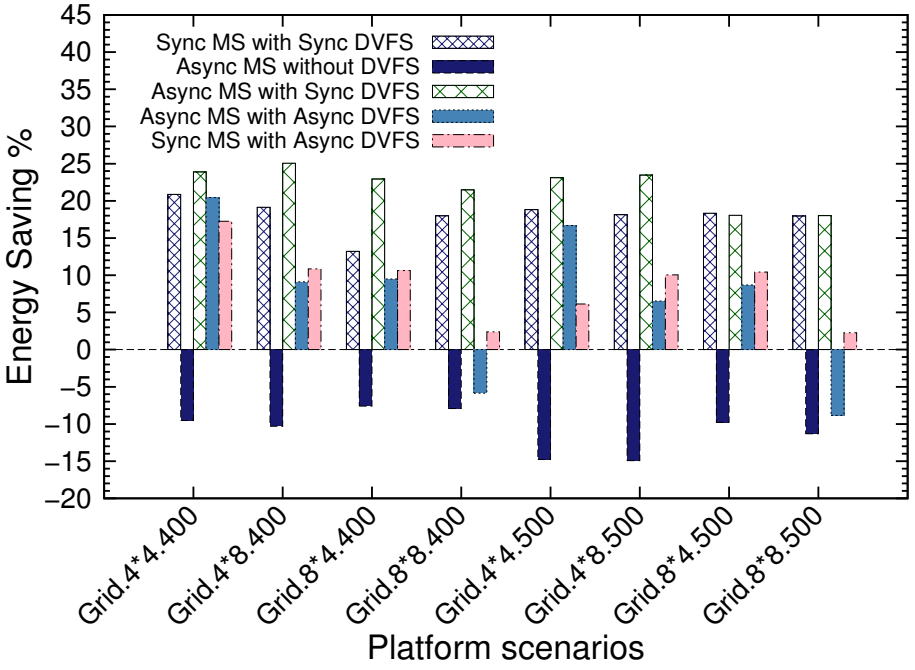

Figure 4.5: The energy saving percentages after applying the HSA algorithm to the different versions and scenarios

reasons presented above, the asynchronous MS with synchronous DVFS version gives the best energy saving percentage when compared to the other versions.

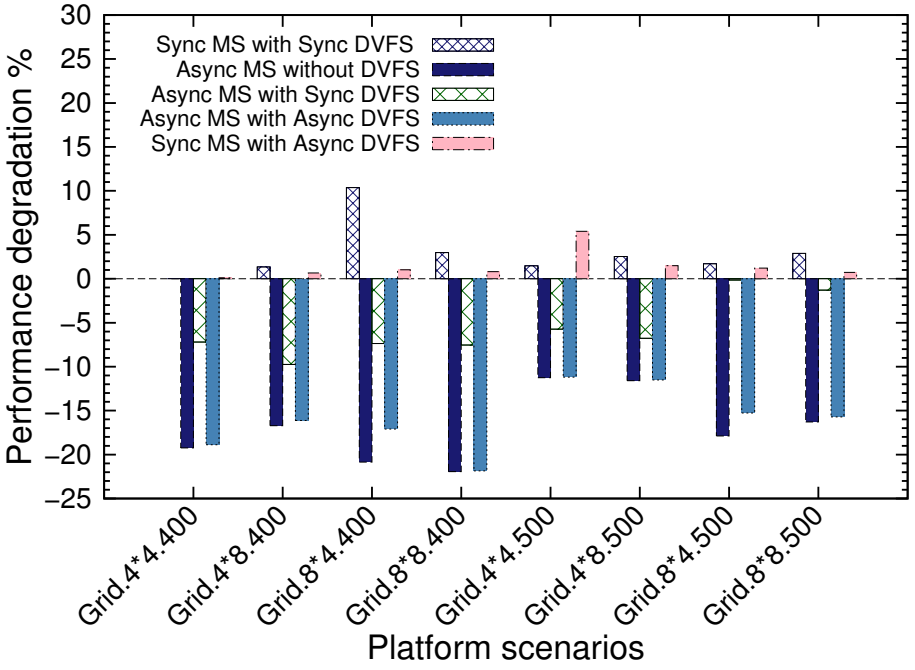

Figure 4.6: The results of the performance degradation

Figure 4.6 shows that some versions have negative performance degradation percentages which means that the new execution time of a given version of the application is less

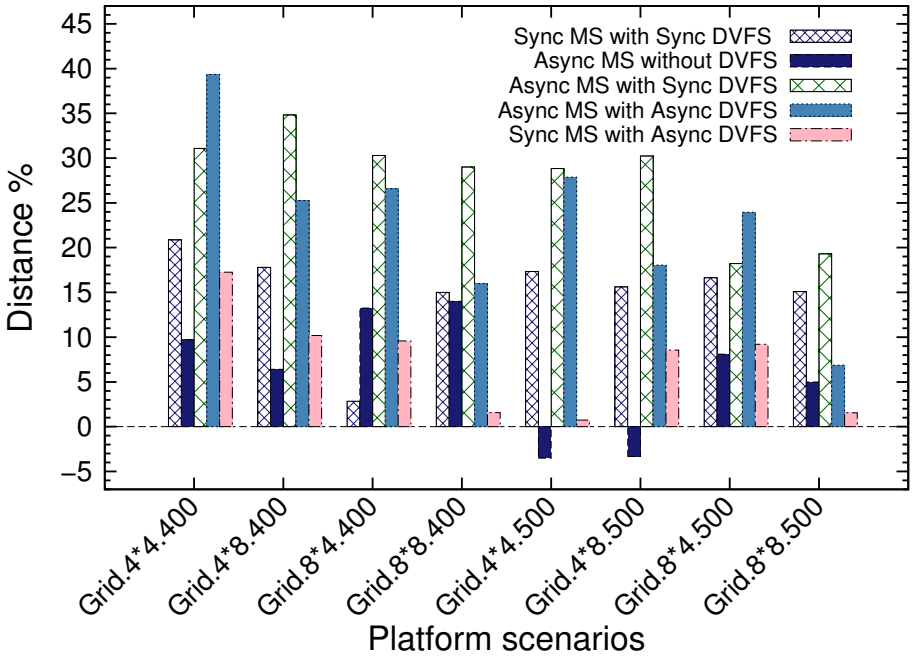

Figure 4.7: The results of the tradeoff distance

than the execution time of the synchronous MS without DVFS. Therefore, the version with the smallest negative performance degradation percentage has actually the best speed up when compared to the other versions. The version that gives the best execution time is the asynchronous MS without DVFS which on average outperforms the synchronous MS without DVFS version by 16.9%. While the worst case is the synchronous MS with synchronous DVFS where the performance is on average degraded by 2.9% when compared to the reference method.

The energy consumption and performance trade-off between these five versions is presented in Figure 4.7. These distance values are computed as the differences between the energy saving and the performance degradation percentages as in the optimization function (3.12). Thus, the best MS version is the one that has the maximum distance between the energy saving and performance degradation. The distance can be negative if the energy saving percentage is less than the performance degradation percentage. The asynchronous MS applying synchronously the HSA algorithm gives the best distance which is on average equal to 27.72%. This version saves on average up to 22% of energy and on average speeds up the application by 5.72%. This overall improvement is due to combining asynchronous computing and the synchronous application of the HSA algorithm.

The two platform scenarios, Grid 4*8 and Grid 8*4, use the same number of computing nodes but give different trade-off results. The versions applying the HSA algorithm and running over the Grid 4*8 platform, give higher distance percentages than those running on the Grid 8*4 platform. In the Grid 8*4 platform scenario more clusters are used than in the Grid 4*8 platform and thus the global system is divided into 8 small subsystems instead of 4. Indeed, each subsystem is assigned to a cluster and synchronously solved by the nodes of that cluster. Dividing the global system into smaller subsystems, increases the number of outer iterations required for the global convergence of the system because

for the multi-splitting system the more the system is decomposed the higher the spectral radius is. For example, the asynchronous MS, applying synchronously the HSA algorithm, requires on average 135 outer iterations when running over the Grid 4*8 platform and 148 outer iterations when running over the Grid 8*4 platform. The increase in the number of executed iterations over the Grid 8*4 platform justifies the increase in energy consumption by applications running over that platform.

4.6.3/ COMPARING THE NUMBER OF ITERATIONS EXECUTED BY THE DIFFERENT MS VERSIONS

The heterogeneity in the computing power of the nodes in the grid has a direct effect on the number of iterations executed by the nodes of each cluster when running an asynchronous iterative message passing method. The fast nodes execute more iterations than the slower ones because the iterations are not synchronized. On the other hand, in the synchronous versions, all the nodes in all the clusters have the same number of iterations and have to wait for the slowest node to finish its iteration before starting the next iteration because the iterations are synchronized.

When the fast nodes asynchronously execute more iterations than the slower ones, they consume more energy without significantly improving the global convergence of the system. Reducing the frequency of the fast nodes will decrease the number of iterations executed by them. If all the nodes, the fast and the slow ones, execute close numbers of iterations, the asynchronous application will consume less energy and its performance will not be significantly affected. Therefore, applying the HSA algorithm over asynchronous applications is very promising. In this section, the number of iterations executed by the asynchronous MS method, while solving a 3D problem of size 400^3 with and without applying the HSA algorithm, is evaluated. In Table 4.3, the standard deviation of the number of iterations executed by the asynchronous application over all the grid platform scenarios, is presented.

Table 4.3: The standard deviation of the numbers of iterations for different asynchronous MS versions running over different grid platforms

Grid platform	Standard deviation		
	Asyn. MS without HSA	Asyn. MS with Asyn. HSA	Asyn. MS with Syn. HSA
Grid.4*4.400	60.43	13.86	1.12
Grid.4*8.400	58.06	27.43	1.22
Grid.8*4.400	50.97	20.76	1.15
Grid.8*8.400	52.46	48.40	2.38

A small standard deviation value means that there is a very small difference between the numbers of iterations executed by the nodes which means fast nodes did not uselessly execute more iterations than the slower ones and the application does not waste a lot of energy. As shown in Table 4.3, the asynchronous MS that applies synchronously the HSA algorithm has the best standard deviation value when compared to the other versions. Two reasons explain the advantage of this method:

1. The applied HSA algorithm selects new frequencies that reduce the computation power of the fast nodes while maintaining the computation power of the slower

nodes. Therefore, it tries to balance as much as possible the computation powers of the heterogeneous nodes.

2. Applying synchronously the HSA algorithm scales down the frequencies of the CPUs at the end of the first iteration of the application. Therefore the computation power of all the nodes is balanced as much as possible since the beginning of the application. On the other hand, applying asynchronously the HSA algorithm onto the asynchronous MS application only changes the frequencies of the nodes after executing many iterations. Therefore, before the frequencies are scaled down, the fast nodes have enough time to execute many more iterations than the slower ones and consequently increase the overall energy consumption of the application.

Finally, the asynchronous MS version that does not apply the HSA algorithm gives the worst standard deviation values because there is a big difference between the numbers of iterations executed by the heterogeneous nodes. Therefore, this version consumes more energy than the other versions and thus saves less energy as shown in Figure 4.4 (a).

4.6.4/ COMPARING DIFFERENT POWER SCENARIOS

In the previous sections, all the results were obtained by assuming that the dynamic and the static powers are respectively equal to 80% and 20% of the total power consumed by a CPU during computation at its highest frequency. The goal of this section is to evaluate the proposed frequency scaling factors selection algorithm when these two power ratios are changed. Two new power scenarios are proposed in this section:

1. The dynamic and the static power are respectively equal to 90% and 10% of the total power consumed by a CPU during computation at its highest frequency.

2. The dynamic and the static power are respectively equal to 70% and 30% of the total power consumed by a CPU during computation at its highest frequency.

The asynchronous MS method solving a 3D problem of size 400^3 was executed over two platform scenarios, the Grid 4*4 and Grid 8*4. Two versions of the asynchronous MS method, with synchronous or asynchronous application of the HSA algorithm, were evaluated on each platform scenario. The energy saving, performance degradation and distance percentages for both versions over both platform scenarios and with the three power scenarios are presented in Figures 4.8 and 4.9.

The displayed results are the average of the percentages obtained from multiple runs. Both figures show that the 90 %-10 % power scenario gives the biggest energy saving percentages. The high dynamic power ratio pushes the HSA algorithm to select bigger scaling factors which decreases exponentially the dynamic energy consumption. Figure 4.10 shows that the HSA algorithm selects in the 90 %-10 % power scenario higher frequency scaling factors than in the other power scenarios for the same application. Moreover, the 90 %-10 % power scenario has the smallest static power consumption per CPU which reduces the effect of the performance degradation, due to scaling down the frequencies of the CPUs, on the total energy consumption of the application. Finally, the 90 %-10 % power scenario gives higher distance percentages than the other two scenarios which means the difference between the energy reduction and the performance

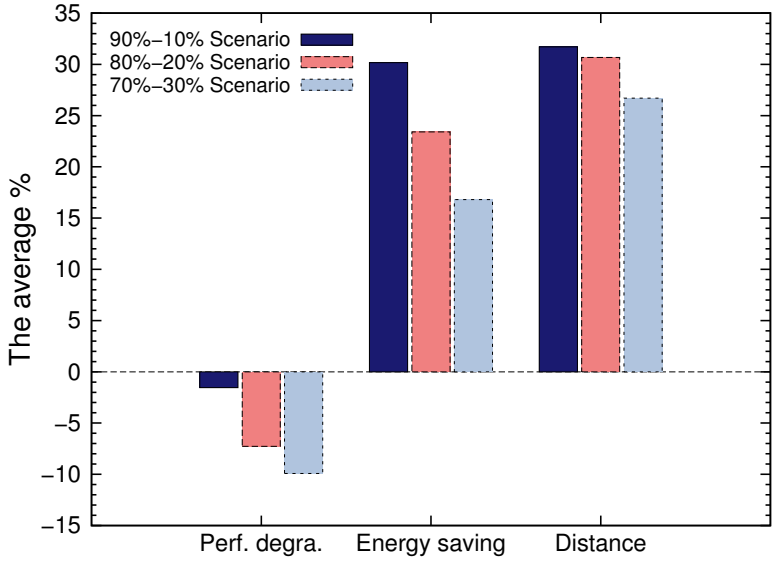

Figure 4.8: The results of the three power scenarios: Synchronous application of the HSA algorithm

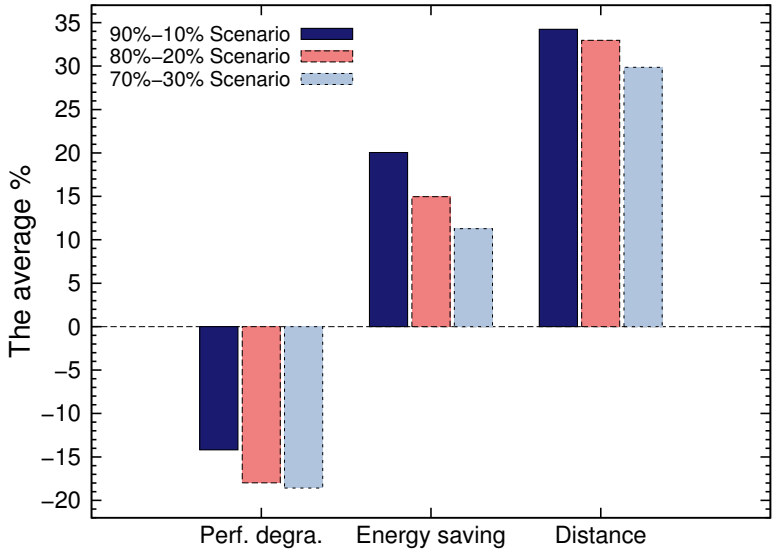

Figure 4.9: The results of the three power scenarios: Asynchronous application of the HSA algorithm

degradation percentages is the highest for this scenario. From these observations, it can be concluded that in a platform with CPUs that consume low static power and high dynamic power, a lot of energy consumption can be reduced by applying the HSA algorithm but the performance degradation might be significant.

The energy saving percentages are the smallest with the 70 %-30 % power scenario. The high static power consumption in this scenario force the HSA algorithm to select small scaling factors in order not to significantly decrease the performance of the application. Indeed, scaling down more the frequency of the CPUs will significantly increase the total execution time and consequently increase the static energy consumption which

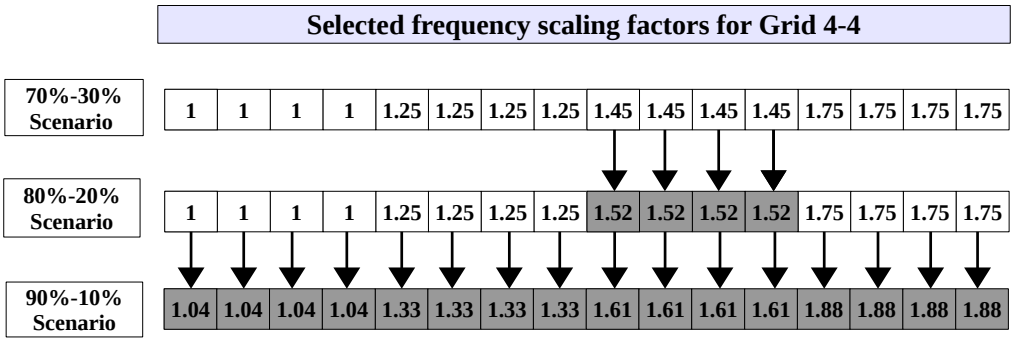

Figure 4.10: Comparison of the selected frequency scaling factors by the HSA algorithm for the three power scenarios

will outweigh the reduction of dynamic energy consumption. Finally, since the dynamic power consumption ratio is relatively small in this power scenario less dynamic energy reduction can be gained in lowering the frequencies of the CPUs than in the other power scenarios. On the other hand, the 70 %-30 % power scenario's main advantage is that its performance suffers the least from the application of the HSA algorithm. From these observations, it can be concluded that in a high static power model just a small percentage of energy can be saved by applying the HSA algorithm.

The asynchronous application of the HSA algorithm on average improves the performance of the application more than the synchronous application of the HSA algorithm. This difference can be explained by the fact that applying the HSA algorithm synchronously scales down the frequencies of the CPUs after the first iteration, while applying the HSA algorithm asynchronously scales them down after many iterations, depending on the heterogeneity of the platform. However, for the same reasons as above, the synchronous application of the HSA algorithm reduces the energy consumption more than the asynchronous one even though, the method applying the first has a bigger execution time than the one applying the latter.

4.6.5/ COMPARING THE HSA ALGORITHM TO THE ENERGY AND DELAY PRODUCT METHOD

Many methods have been proposed to optimize the trade-off between the energy consumption and the performance of message passing applications. A well known optimization model used to solve this problem is the energy and delay product, $EDP = energy \times delay$. In [13, 24, 63], the researchers used equal weights for the energy and delay factors. However, others added some weights to the factors in order to direct the optimization towards more energy saving or less performance degradation. For example, in [55] they used the product $EDP = energy \times delay^2$ which favour performance over energy consumption reduction.

In this work, the proposed scaling factors selection algorithm optimizes both the energy consumption and the performance at the same time and gives the same weight to both factors as in Equation 3.12. In this section, to evaluate the performance of the HSA algorithm, it is compared to the algorithm proposed by Spiliopoulos et al. [75]. The latter is an online method that selects for each processor the frequency that minimizes

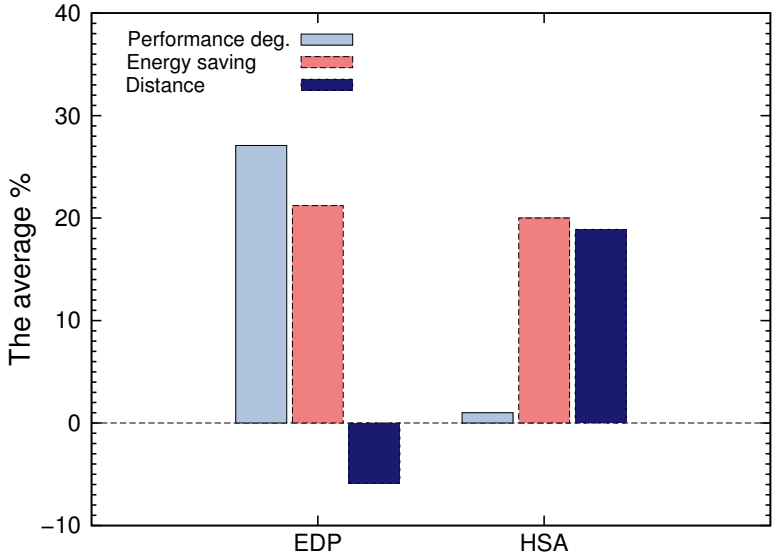

Figure 4.11: Synchronous application of the frequency scaling selection method on the synchronous MS version

the energy and delay product in order to reduce the energy consumption of a parallel application running over a homogeneous multi-cores platform. It gives the same weight to both metrics and predicts both the energy consumption and the execution time for each frequency gear as in the HSA algorithm.

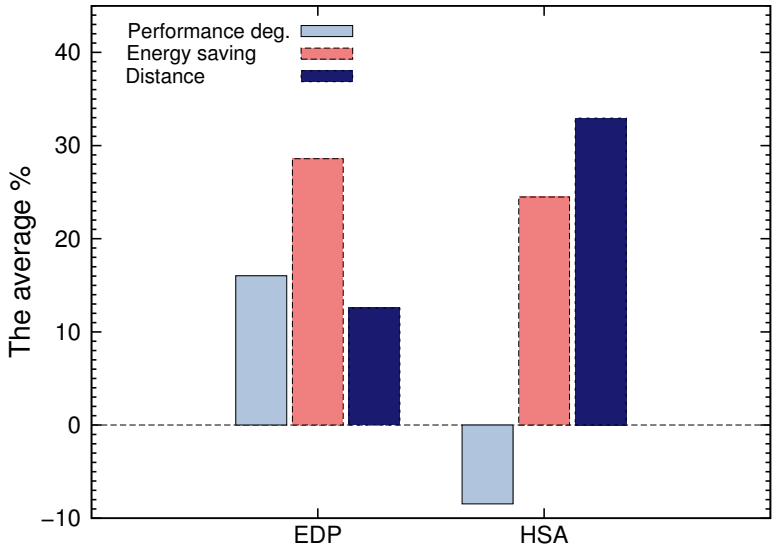

Figure 4.12: Synchronous application of the frequency scaling selection method on the asynchronous MS version

To fairly compare the HSA algorithm with the algorithm of Spiliopoulos et al., the same energy models, Equation (3.9) or (4.4), and execution time models, Equation (3.8) or (4.2), are used to predict the energy consumptions and the execution times. The EDP objective function can be equal to zero when the predicted delay is equal to zero. More-

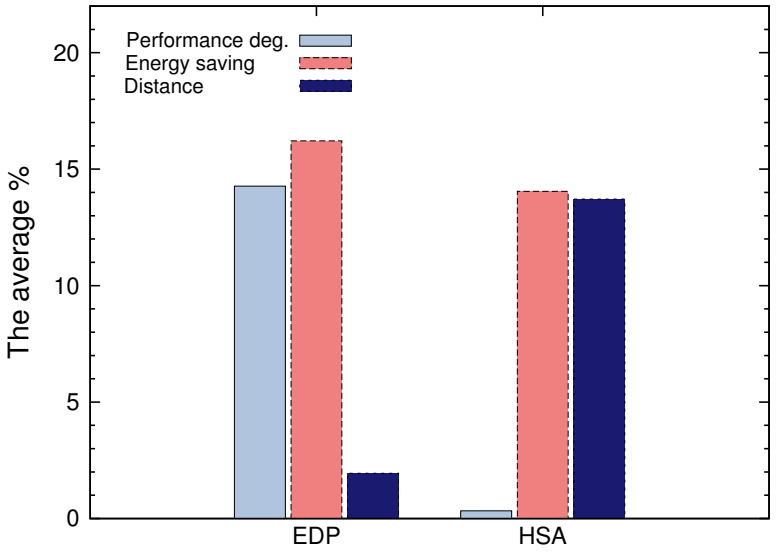

Figure 4.13: Asynchronous application of the frequency scaling selection method on the synchronous MS version

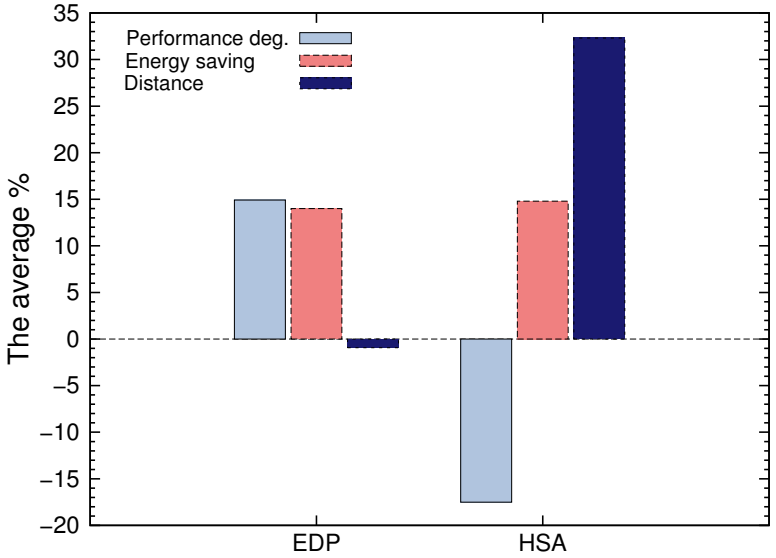

Figure 4.14: Asynchronous application of the frequency scaling selection method on the asynchronous MS version

over, this product is equal to zero before applying any DVFS operation. To eliminate the zero values, the EDP function must take the following form:

$$EDP = E_{Norm} \times (1 + D_{Norm}) \tag{4.8}$$

where E_{Norm} is the normalized energy consumption which is computed as in Equation (2.5) and D_{Norm} is the normalized delay of the execution time which is computed as follows:

$$D_{Norm} = 1 - P_{Norm} = 1 - (\frac{T_{old}}{T_{new}}) \tag{4.9}$$

Where P_{Norm} is computed as in Equation (2.6). Furthermore, the EDP algorithm starts

the search process from the initial frequencies that are computed as in Equation (3.7). It stops the search process when it reaches the minimum available frequency for each processor. The EDP algorithm was applied to the synchronous and asynchronous MS algorithm solving a 3D problem of size 400^3. Two platform scenarios, Grid 4*4 and Grid 4*8, were chosen for this experiment. The EDP method was applied synchronously and asynchronously to the MS application as for the HSA algorithm. The comparison results of the EDP and HSA algorithms are presented in the Figures 4.11, 4.14,4.13 and 4.14. Each of these figures presents the energy saving, performance degradation and distance percentages for one version of the MS algorithm. The results shown in these figures are also the average of the results obtained from running each version of the MS method over the two platform scenarios described above. All the figures show that the proposed

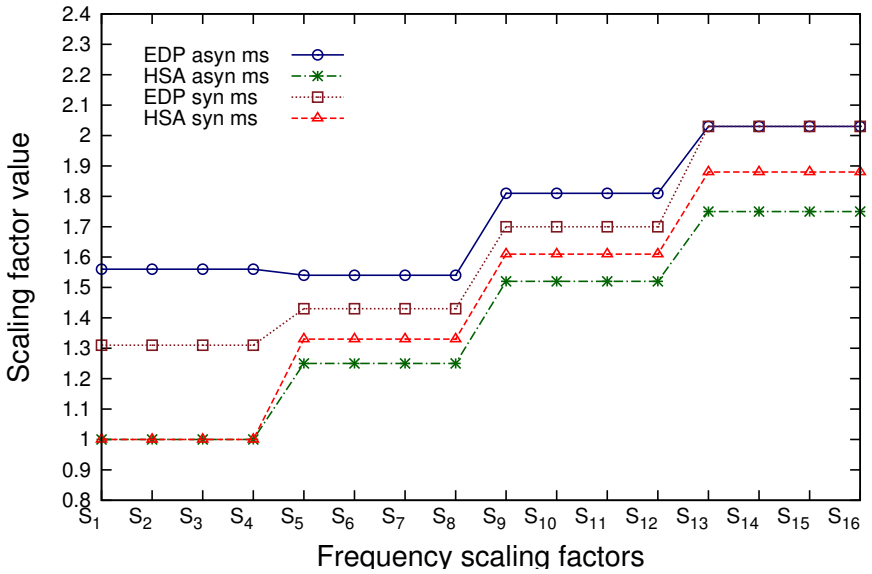

Figure 4.15: Comparison of the selected frequency scaling factors by the two algorithms over the Grid 4*4 platform scenario

HSA algorithm outperforms the EDP algorithm in terms of energy saving and performance degradation. EDP gave for some scenarios negative trade-off values which mean that the performance degradation percentages are higher than the energy saving percentages, while the HSA algorithm gives positive trade-off values over all scenarios. The frequency scaling factors selected by the EDP are most of the time higher than those selected by the HSA algorithm as shown in Figure 4.15. The results confirm that higher frequency scaling factors do not always give more energy saving, especially when the overall execution time is drastically increased. Therefore, the HSA method that computes the maximum distance between the energy saving and the performance degradation is an effective method to optimize these two metrics at the same time.

4.7/ THE EXPERIMENTAL RESULTS OVER GRID'5000

The performance of algorithm (8) was evaluated by executing the iterative multi-splitting method on the Grid'5000 textbed [3]. This testbed is a large-scale platform that consists of ten sites distributed all over metropolitan France and Luxembourg. Moreover, some

sites are equipped with power measurement tools that capture the power consumption for each node on those sites. Same method for computing the dynamic power consumption described in section 3.6 is used. Table 4.4 presents the characteristics of the selected clusters which are located on four different sites.

Table 4.4: CPUs characteristics of the selected clusters

Cluster Name	CPU model	Max Freq. GHz	Min Freq. GHz	Diff. Freq. GHz	Site	Dynamic power of one core
Taurus	Intel E5-2630	2.3	1.2	0.1	Lyon	35 W
Graphene	Intel X3440	2.53	1.2	0.133	Nancy	23 W
Parapide	Inte X5570	2.93	1.6	0.133	Rennes	23 W
StRemi	AMD 6164 HE	1.7	0.8	0.2	Reims	6 W

The dynamic power of each core with maximum frequency is computed as the difference between the measured power of the core, only when it is computing at maximum frequency, and the measured power of that core when it is idle as in 3.15. The CPUs' constructors do not specify the amount of static power their CPUs consume. Therefore, the static power consumption is assumed to be equal to 20 % of the dynamic power consumption. The experiments were conducted on problems of size $N = 400^3$ and $N = 500^3$ over 4 distributed clusters described in Table 4.4. Each cluster is composed of 8 homogeneous nodes.

Algorithm 8 was applied synchronously and asynchronously to both synchronous and asynchronous MS applications. Figures 4.16 and 4.17 show the energy consumption and the execution time of the multi-splitting application with and without the application of the HSA algorithm respectively. The asynchronous MS consumes more energy than the synchronous one. Also, it can be noticed that both the asynchronous and synchronous MS with synchronous application of the HSA algorithm consume less energy than the other versions of the application. Synchronously applying the HSA algorithm allows them to scale down the CPUs' frequencies at the beginning of the second iteration. Thus, the consumption of dynamic energy by the application is reduced from the second iteration until the end of the application. On the contrary, with the asynchronous application of the HSA algorithm, the new frequencies cannot be computed at the end of the first iteration and consequently cannot be applied at the beginning of the second iteration. Indeed, since the performance information gathered during the first iteration is not sent synchronously at the end of the first iteration, fast nodes might execute many iterations before receiving the performance information, computing the new frequencies based on this information and applying the new computed frequencies. Therefore, many iterations might be computed by CPUs running on their highest frequency and consuming more dynamic energy than the scaled down processors. Moreover, the execution time of the asynchronous MS version is lower than the execution time of the synchronous MS version because there is no idle time in the asynchronous version and the communications are overlapped by computations. Since the consumption of static energy is proportional to the execution time, the asynchronous MS version consumes less static energy than the synchronous version.

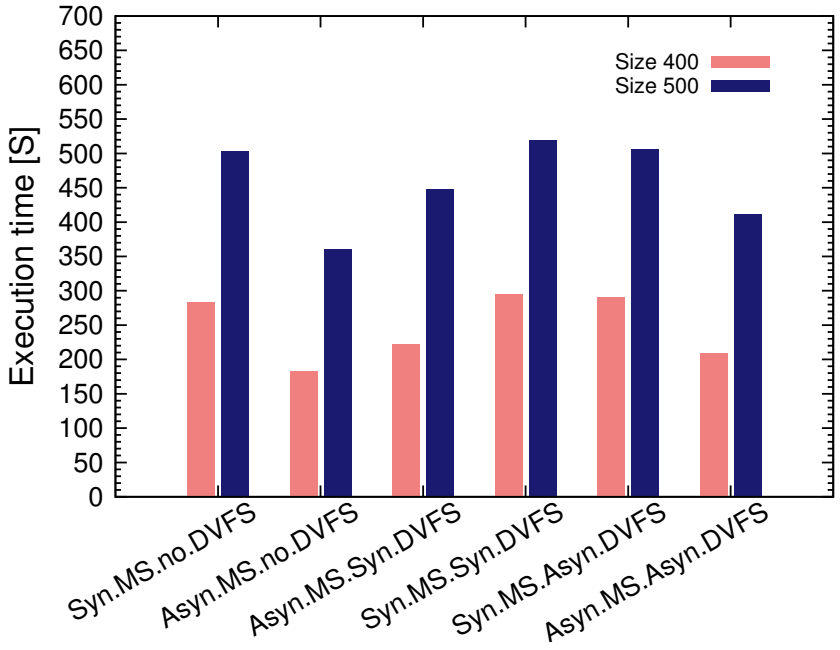

Figure 4.16: Comparing the execution time

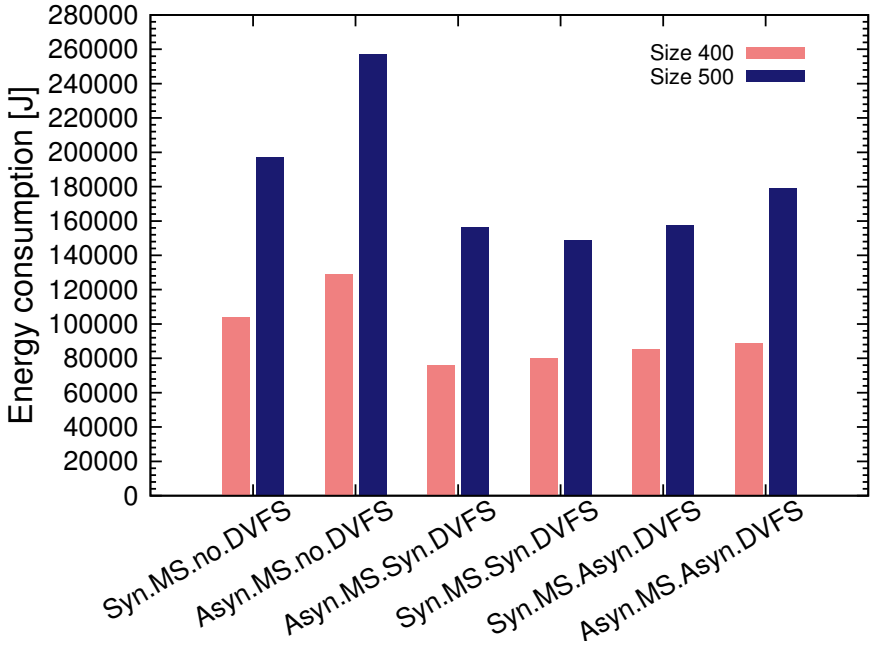

Figure 4.17: Comparing the energy consumption

Table 4.5 shows that there are positive and negative performance degradation percentages. A negative value means that the new execution time of a given version of the application is less than the execution time of the synchronous MS without DVFS. Therefore, the version with the smallest negative performance degradation percentage has actually the best speed up when compared to the other versions. The energy consumption and performance trade-offs between these four versions can be computed as in the optimization Function (3.12). The asynchronous MS applying synchronously the

Size	Method	Energy saving %	Perf. degra.%	Distance
400	Sync MS with Sync DVFS	23.16	4.12	19.04
	Sync MS with Async DVFS	18.36	2.59	15.77
	Async MS with Sync DVFS	26.93	-21.48	48.41
	Async MS with Async DVFS	14.9	-26.41	41.31
500	Sync MS with Sync DVFS	24.57	3.15	21.42
	Sync MS with Async DVFS	19.97	0.60	19.37
	Async MS with Sync DVFS	20.69	-10.95	31.64
	Async MS with Async DVFS	9.06	-18.22	27.28

Table 4.5: The experimental results of HSA algorithm

HSA algorithm gives the best distance which is equal to 48.41%. This version saves up to 26.93% of energy and even reduces the execution time of the application by 21.48%. This overall improvement is due to combining asynchronous computing and the synchronous application of the HSA algorithm.

Finally, this section shows that the obtained results over Grid'5000 are comparable to the simulation results of Section 4.6.2, the asynchronous MS applying synchronously the HSA algorithm is the best version in both of sections. Moreover, the results over Grid'5000 are better than simulation results because the computing clusters used in the Grid'5000 experiments are more heterogeneous in term of the computing power and network characteristics than the simulated platform with SimGrid. For example, the nodes in StRemi cluster have lower computing powers compared to the other used three clusters of Grid'5000 platform. As a result, the increase in the heterogeneity between the clusters' computing nodes increases the idle times which forces the proposed algorithm to select a big scaling factors and thus saving more energy.

4.7.1/ COMPARING THE HSA ALGORITHM TO THE ENERGY AND DELAY PRODUCT METHOD

The EDP algorithm, described in Section 4.6.5, was applied synchronously and asynchronously to both the synchronous and asynchronous MS application of size $N = 400^3$. The experiments were conducted over 4 distributed clusters, described in Table 4.4, and 8 homogeneous nodes were used from each cluster. Table 4.6 presents the results of energy saving, performance degradation and distance percentages when applying the EDP method on four different MS versions. Figure 4.18 compares the distance percentages, computed as the difference between energy saving and performance degradation percentages, of the EDP and HSA algorithms. This comparison shows that the proposed HSA algorithm gives better energy reduction and performance trade-off than the EDP method. EDP gives better results when evaluated over Grid'5000 than over the simulator because the nodes used from Grid'5000 are more heterogeneous than those simulated with SimGrid.

Table 4.6: The EDP algorithm results over the Grid'5000

Method name	Energy saving %	Perf. degra.%	Distance %
Sync MS with Sync DVFS	21.83	12.78	9.05
Sync MS with Async DVFS	18.26	7.68	10.58
Async MS with Sync DVFS	24.95	-12.24	37.19
Async MS with Async DVFS	10.32	-17.04	27.36

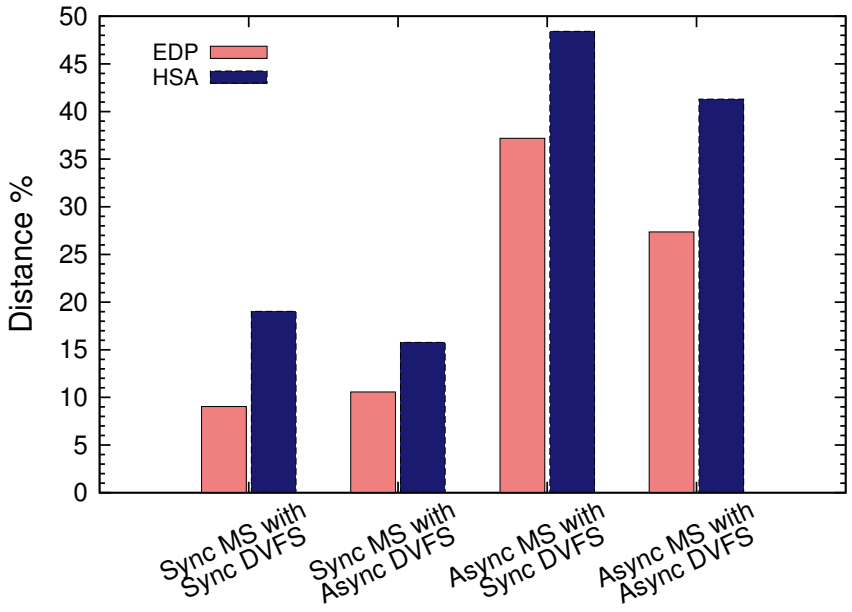

Figure 4.18: Comparing the trade-off percentages of HSA and EDP methods over the Grid'5000

4.8/ CONCLUSIONS

This chapter presents a new online frequency selection algorithm for asynchronous iterative applications running over a grid. It selects the best vector of frequencies that maximizes the distance between the predicted energy consumption and the predicted execution time. The algorithm uses new energy and performance models to predict the energy consumption and the execution time of asynchronous or hybrid message passing iterative applications running over grids. The proposed algorithm was evaluated twice over the SimGrid simulator and Grid'5000 testbed while running a multi-splitting (MS) application that solves 3D problems. The experiments were executed over different grid scenarios composed of different numbers of clusters and different numbers of nodes per cluster. The HSA algorithm was applied synchronously and asynchronously on a synchronous and an asynchronous version of the MS application. Both the simulation and real experiment results show that applying synchronous HSA algorithm on an asynchronous MS application gives the best trade-off between energy consumption reduction and performance compared to other scenarios. In the simulation results, this scenario saves on average the energy consumption by 22% and reduces the execution time of the application by 5.72%. This version optimizes both of the dynamic energy consumption by applying synchronously the HSA algorithm at the end of the first iteration and the static energy consumption by using asynchronous communications between nodes from

different clusters which are overlapped by computations. The HSA algorithm was also evaluated over three power scenarios. As expected, the algorithm selects different vectors of frequencies for each power scenario. The highest energy consumption reduction was achieved in the power scenario with the highest dynamic power and the lowest performance degradation was obtained in the power scenario with the highest static power. The proposed algorithm was compared to another method that uses the well known energy and delay product as an objective function. The comparison results showed that the proposed algorithm outperforms the latter by selecting a vector of frequencies that gives a better trade-off between the energy consumption reduction and the performance.

The experiments conducted over Grid'5000 showed that applying the synchronous HSA algorithm on an asynchronous MS application gives the best results. It saves the energy consumption by 26.93% and reduces the execution time of the application by 21.48%. The experiments executed over Grid'5000 give better results than those simulated with SimGrid because the nodes used in Grid'5000 were more heterogeneous than the ones simulated by SimGrid.

III

CONCLUSION AND PERSPECTIVES

5

CONCLUSION AND PERSPECTIVES

5.1/ CONCLUSION

In this dissertation, we have proposed a method to optimize both the energy consumption and the performance at the same time of synchronous and asynchronous applications with iterations running over cluster and grid platforms. Dynamic voltage and frequency scaling (DVFS) technique was used to lower the frequency of the processor to reduce its energy consumption while computing. Reducing the frequency of the processor decreases its computing power which might increase the execution time. In this work, different energy consumption and performance models were developed to predict the energy consumption and performance of parallel applications with iterations. Depending on these models, an objective function was defined as the best trade-off relation between the energy consumption and the performance of the parallel application. This objective function was used in the frequency selecting algorithms which optimize at the same time both the energy consumption and the performance of the parallel application with iterations.

The first part of this dissertation, chapter 1, presented different types of parallelism levels which have been classified according to the used hardware and software techniques. Different parallel architectures have also been described and classified according to the used memory model: shared and distributed memory. The two types of parallel applications with iterations: synchronous and asynchronous ones have been discussed and compared to each others. Synchronous distributed applications are well adapted to local homogeneous clusters with a high speed network link, while the asynchronous ones are more suited to grids. At the end of this chapter, an energy consumption model proposed in the literature to estimate the energy consumption of parallel applications was explained. This model does not take into account the communication time of the parallel application being executed. Also, it is not well adapted to a heterogeneous architecture where each type of processor might have a different power consumption value.

In the second part of the dissertation, a new energy and performance models for synchronous and asynchronous message passing applications with iterations running over clusters and grid, were presented. To simultaneously optimize the energy and performance of these applications, the trade-off relation has been defined as the maximum distance between the predicted energy and performance curves. This objective function is used by the frequency selecting algorithm to select the available frequency scaling factors that give the optimal energy consumption to performance trade-off. We have proposed four different frequency scaling algorithms, each one of them is adapted to a different execution context, such as synchronous or asynchronous communications, homogeneous or

heterogeneous nodes, and local or distributed architectures. They used the computation and communication times measured at the first iteration of the parallel application with iterations to predict the energy consumption and the performance of the parallel application at every available frequency. All these algorithms work online and introduce a very small runtime overhead. They also do not require any profiling or training.

In chapter 2, a new online scaling factor selection method that optimizes simultaneously the energy and performance of a distributed synchronous application with iterations running on a homogeneous cluster has been proposed. This algorithm was applied to the NAS benchmarks of the class C and executed over the SimGrid simulator. Firstly, Rauber and Rünger's energy model was used in the proposed algorithm to select the best frequency gear. The proposed algorithm was compared to the Rauber and Rünger's optimization method. The results of the comparison showed that the proposed algorithm gives better energy to performance trade-off ratios compared to their methods while using the same energy model. Secondly, a new energy consumption model was developed to take into consideration both the computation and communication times and their relation with the frequency scaling factor. The new energy model was used by the proposed algorithm. The new simulation results demonstrated that the new model is more accurate and realistic than the previous one.

In chapter 3, two new online frequency scaling factors selecting algorithms adapted for synchronous application with iterations running over a heterogeneous cluster and a grid were presented. Each algorithm uses new energy and performance models which take into account the characteristics of the parallel platform being used. Firstly, the proposed scaling factors selection algorithm for a heterogeneous local cluster was applied to the NAS parallel benchmarks and evaluated over SimGrid. The results of the experiments showed that the algorithm on average reduces by 29.8% the energy consumption of the class C of the NAS benchmarks executed over 8 nodes while limiting the degradation of the performance to 3.8%. Different frequency scaling factors were selected by the algorithm according to the ratio between the computation and communication times when different number of nodes were used, and when different static and dynamic CPU powers have been used. Secondly, the proposed scaling factors selection algorithm for a grid was applied to the NAS parallel benchmarks and the class D of these benchmarks was executed over the Grid5000 testbed platform. The experiments conducted over 16 nodes distributed over three clusters, showed that the algorithm on average reduces by 30% the energy consumption for all the NAS benchmarks while on average only degrading by 3.2% their performance. The algorithm was also evaluated in different scenarios that vary in the distribution of the computing nodes between different clusters' sites or use multi-cores per node architectures or consume different static power values. The algorithm selects different vectors of frequencies according to the computations and communication times ratios, and the values of the static and measured dynamic powers of the CPUs. Both of the proposed algorithms were compared to another method that uses the well known energy and delay product as an objective function. The comparison results showed that the proposed algorithms outperform the latter by selecting vectors of frequencies that give a better trade-off between energy consumption reduction and performance.

In chapter 4, a new online frequency selection algorithm were adapted for asynchronous iterative applications running over a grid was presented. The algorithm uses new energy and performance models to predict the energy consumption and the execution time of asynchronous or hybrid message passing iterative applications running

over a grid. The proposed algorithm was evaluated twice over the SimGrid simulator and Grid'5000 testbed while running a multi-splitting (MS) application that solves 3D problems. The experiments were executed over different grid scenarios composed of different numbers of clusters and different numbers of nodes per cluster. The proposed algorithm was applied synchronously and asynchronously on synchronous and asynchronous versions of the MS iterative application. Both the simulations and real experiments results showed that applying synchronously the frequency selecting algorithm on an asynchronous MS application gives the best tradeoff between energy consumption reduction and performance when compared to the other scenarios. In the simulation results, this scenario reduces on average the energy consumption by 22% and decreases the execution time of the application by 5.72%. This version optimizes both of the dynamic energy consumption by applying synchronously the HSA algorithm at the end of the first iteration of the iterative application and the static energy consumption by using asynchronous communications between nodes from different clusters which are overlapped by computations. The proposed algorithm was also evaluated over three power scenarios which selects different vectors of frequencies proportionally to the dynamic and static powers values. More energy reduction was achieved when the ratio of the dynamic power was increased and vice versa. Whereas, the performance degradation percentages were decreased when the static power ratio was increased. In the Grid'5000 experiments, this scenario reduces the energy consumption by 26.93% and decreases the execution time of the application by 21.48%. The experiments executed over Grid'5000 give better results than those simulated with SimGrid because the nodes used in Grid'5000 were more heterogeneous than the ones simulated by SimGrid. In both of the Simulations and real experiments, the proposed algorithm was compared to a method that uses the well known energy and delay product as an objective function. The comparison results showed that the proposed algorithm outperforms the latter by selecting a vector of frequencies that gives a better trade-off between the energy consumption reduction and the performance.

5.2/ PERSPECTIVES

In the near future, we will adapt the proposed algorithms to take into consideration the variability between some iterations. For example, each proposed algorithm can be executed twice: after the first iteration the frequencies are scaled down according to the execution times measured in the first iteration, then after a fixed number of iterations, the frequencies are adjusted according to the execution times measured during the fixed number of iterations. If the computing power of the system is constantly changing, it would be interesting to implement a mechanism that detects this change and adjusts the frequencies according to the variability of the system. Also, it would be interesting to evaluate the scalability of the proposed algorithms by running them on large platforms composed of many thousands of cores. The scalability of the algorithms can be improved by distributing them in a hierarchical manner where a leader is chosen for each cluster or a group of nodes to compute their scaled frequencies and by using asynchronous messages to exchange the the data measured at the first iteration.

The proposed algorithms should be applied to other message passing methods with iterations in order to see how they adapt to the characteristics of these methods. Also, it would be interesting to explore if a relation can be found between the numbers of asynchronous iterations required to global convergence and the applied frequencies to the

nodes. The number of iterations required by each node for global convergence is not known in advance and the change in CPUs frequencies changes the number of iterations required by each node for global convergence.

Furthermore, the proposed algorithms for heterogeneous platforms, in chapters 3 and 4, should be applied to heterogeneous platforms composed of CPUs and GPUs. Indeed, most of the works in the green computing field showed that these mixed platforms of GPUs and CPUs are more energy efficient than those composed of only CPUS.

Finally, it would be interesting to verify the accuracy of the results returned by the energy models by comparing them to the values given by instruments that measure the energy consumptions of CPUs during the execution time, as in [70].

PUBLICATIONS

JOURNAL ARTICLES

[**1**] Ahmed Fanfakh, Jean-Claude Charr, Raphaël Couturier, Arnaud Giersch. Optimizing the energy consumption of message passing applications with iterations executed over grids. *Journal of Computational Science*, 2016.

[**2**] Ahmed Fanfakh, Jean-Claude Charr, Raphaël Couturier, Arnaud Giersch. Energy Consumption Reduction for Asynchronous Message Passing Applications. *Journal of Supercomputing*, 2016, (Accepted with minor revisions)

CONFERENCE ARTICLES

[**1**] Jean-Claude Charr, Raphaël Couturier, Ahmed Fanfakh, Arnaud Giersch. Dynamic Frequency Scaling for Energy Consumption Reduction in Synchronous Distributed Applications. *ISPA 2014: The* 12^{th} *IEEE International Symposium on Parallel and Distributed Processing with Applications*, pp. 225-230. IEEE Computer Society, Milan, Italy (2014).

[**2**] Jean-Claude Charr, Raphaël Couturier, Ahmed Fanfakh, Arnaud Giersch. Energy Consumption Reduction with DVFS for Message Passing Iterative Applications on Heterogeneous Architectures. *The* 16^{th} *IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing*. pp. 922-931. IEEE Computer Society, INDIA (2015).

[**3**] Ahmed Fanfakh, Jean-Claude Charr, Raphaël Couturier, Arnaud Giersch. CPUs Energy Consumption Reduction for Asynchronous Parallel Methods Running over Grids. *The* 19^{th} *IEEE International Conference on Computational Science and Engineering*. IEEE Computer Society, Paris (2016).

BIBLIOGRAPHY

[1] CPU frequency scaling. [online], https://wiki.archlinux.org.

[2] Geforce graphics processors GTX series. [online], http://www.nvidia.com.

[3] Grid'5000, [online], http://www.grid5000.fr.

[4] Mellanox technologies completes acquisition of ezchip. [online], http://www.tilera.com.

[5] Oar, [online], http://www.oar.imag.fr.

[6] The Green500 List of Heterogeneous Supercomputing Systems.

[7] TOP500 Supercomputers Sites.

[8] U.S. Energy Information Administration, Annual Energy Outlook 2015.

[9] G. S. Almasi and A. Gottlieb. *Highly Parallel Computing*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1989.

[10] Hartwig Anzt. *Asynchronous and Multiprecision Linear Solvers*. PhD thesis, Karlsruher Institut für Technologie, Bade-Wurtemberg,Germany, 2012.

[11] A. Azevedo, I. Issenin, R. Cornea, R. Gupta, N. Dutt, A. Veidenbaum, and A. Nicolau. Profile-based dynamic voltage scheduling using program checkpoints. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 168–175, Washington, DC, USA, 2002. IEEE Computer Society.

[12] Jacques Bahi, Sylvain Contassot-Vivier, and Raphaël Couturier. *Parallel Iterative Algorithms: from sequential to grid computing*, volume 1 of *Numerical Analysis and Scientific Computating*. Chapman and Hall/CRC, 2007.

[13] A. Baldassin, J.P.L. de Carvalho, L.A.G. Garcia, and R. Azevedo. Energy-performance tradeoffs in software transactional memory. In *Computer Architecture and High Performance Computing (SBAC-PAD), 2012 IEEE 24th International Symposium on*, pages 147–154, Oct 2012.

[14] Anton Beloglazov and Rajkumar Buyya. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. *Concurrency and Computation:Practice and Experience*, 24(13):1397–1420, September 2012.

[15] Dimitri P. Bertsekas and John N. Tsitsiklis. *Parallel and distributed computation : numerical methods*. Prentice-Hall International, Englewood Cliffs N.J, 1989. Includes index.

[16] Aurelien Bouteiller, Thomas Hérault, Géraud Krawezik, Pierre Lemarinier, and Franck Cappello. Mpich-v project: A multiprotocol automatic fault-tolerant mpi. *IJHPCA*, 20(3):319–333, 2006.

[17] D.C. Brock and G.E. Moore. *Understanding Moore's Law: Four Decades of Innovation*. Chemical Heritage Foundation, 2006.

[18] Henri Casanova, Arnaud Legrand, and Martin Quinson. SimGrid: a generic framework for large-scale distributed experiments. In *Proceedings of the Tenth International Conference on Computer Modeling and Simulation*, UKSIM '08, pages 126–131, Washington, DC, USA, 2008. IEEE Computer Society.

[19] Rohit Chandra, Leonardo Dagun, and Dave Kohr ... [et al.]. *Parallel programming in OpenMP*. Morgan Kaufmann Publishers, San Francisco, CA, 2001.

[20] Jian Chen and L.K. John. Energy-aware application scheduling on a heterogeneous multi-core system. In *Workload Characterization, 2008. IISWC 2008. IEEE International Symposium on*, pages 5–13, Sept 2008.

[21] Jian-Jia Chen, Kai Huang, and Lothar Thiele. Dynamic frequency scaling schemes for heterogeneous clusters under quality of service requirements. *Journal of Information Science and Engineering*, 28(6):1073–1090, 2012.

[22] Yu-Liang Chou, Shaoshan Liu, Eui-Young Chung, and Jean-Luc Gaudiot. An energy and performance efficient DVFS scheme for irregular parallel divide-and-conquer algorithms on the Intel SCC. *IEEE Computer Architecture Letters*, 99:1, 2013.

[23] A. Cocana-Fernandez, L. Sanchez, and J. Ranilla. A software tool to efficiently manage the energy consumption of hpc clusters. In *Fuzzy Systems (FUZZ-IEEE), 2015 IEEE International Conference on*, pages 1–8, Aug 2015.

[24] Ryan Cochran, Can Hankendi, Ayse Coskun, and Sherief Reda. Identifying the optimal energy-efficient operating points of parallel workloads. In *Proceedings of the International Conference on Computer-Aided Design*, ICCAD '11, pages 608–615, NJ, USA, 2011. IEEE Press.

[25] Ryan Cochran, Can Hankendi, Ayse K. Coskun, and Sherief Reda. Pack & cap: Adaptive DVFS and thread packing under power caps. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 175–185, NY, USA, 2011. ACM.

[26] Shane Cook. *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2013.

[27] David E. Culler, Anoop Gupta, and Jaswinder Pal Singh. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 1997.

[28] G. Da Costa, J.-P. Gelas, Y. Georgiou, L. Lefevre, A.-C. Orgerie, J. Pierson, O. Richard, and K. Sharma. The green-net framework: Energy efficiency in large scale distributed systems. In *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–8, May 2009.

[29] Georges Da Costa, Marcos Dias de Assunção, Jean-Patrick Gelas, Yiannis Georgiou, Laurent Lefèvre, Anne-Cécile Orgerie, Jean-Marc Pierson, Olivier Richard, and Amal Sayah. Multi-facet approach to reduce energy consumption in clouds and grids: The green-net framework. In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, pages 95–104, New York, NY, USA, 2010. ACM.

[30] G. Dhiman and T.S. Rosing. Dynamic voltage frequency scaling for multi-tasking systems using online learning. In *Low Power Electronics and Design (ISLPED), 2007 ACM/IEEE International Symposium on*, pages 207–212, Aug 2007.

[31] Hesham El-Rewini and Mostafa Abd-El-Barr. *Advanced Computer Architecture and Parallel Processing (Wiley Series on Parallel and Distributed Computing)*. Wiley-Interscience, 2005.

[32] M.N. El Tarazi. Some convergence results for asynchronous algorithms. *Numerische Mathematik*, 39:325–340, 1982.

[33] Michael J. Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, C-21(9):948–960, September 1972.

[34] Vincent W. Freeh, Feng Pan, Nandini Kappiah, David K. Lowenthal, and Rob Springer. Exploring the energy-time tradeoff in MPI programs on a power-scalable cluster. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Papers - Volume 01*, Washington, DC, USA, 2005. IEEE Computer Society.

[35] Andreas Frommer and Daniel B. Szyld. Asynchronous iterations with flexible communication for linear systems. *Calculateurs Parallèles Réseaux et Systèmes Répartis*, 10:421–429, 1998.

[36] Andreas Frommer and Daniel B. Szyld. On asynchronous iterations. *Journal of Computational and Applied Mathematics*, 123(1–2):201 – 216, 2000. Numerical Analysis 2000. Vol. III: Linear Algebra.

[37] Rong Ge, R. Vogt, J. Majumder, A. Alam, M. Burtscher, and Ziliang Zong. Effects of dynamic voltage and frequency scaling on a k20 gpu. In *Parallel Processing (ICPP), 2013 42nd International Conference on*, pages 826–833, 2013.

[38] Fayez Gebali. *Algorithms and Parallel Computing*. Wiley Publishing, 1st edition, 2011.

[39] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: Portable Parallel Programming with the Message-passing Interface*. MIT Press, Cambridge, MA, USA, 1999.

[40] Amina Guermouche, Nicolas Triquenaux, Benoît Pradelle, and William Jalby. Minimizing energy consumption of MPI programs in realistic environment. *Computing Research Repository*, 2015.

[41] C.-H. Hsu and Wu chun Feng. A power-aware run-time system for high-performance computing. In *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*, pages 1–9, Nov 2005.

[42] Kai Hwang. *Advanced Computer Architecture: Parallelism, Scalability, Programmability*. McGraw-Hill Higher Education, 1st edition, 1992.

[43] Kaustubh R. Joshi, Matti A. Hiltunen, Richard D. Schlichting, and William H. Sanders. Blackbox prediction of the impact of DVFS on end-to-end performance of multitier systems. *ACM SIGMETRICS Performance Evaluation Review*, 37(4):59–63, 2010.

[44] Nam Sung Kim, Todd Austin, David Blaauw, Trevor Mudge, Krisztián Flautner, Jie S. Hu, Mary Jane Irwin, Mahmut Kandemir, and Vijaykrishnan Narayanan. Leakage current: Moore's law meets static power. *Computer*, 36(12):68–75, December 2003.

[45] H. Kimura, M. Sato, Y. Hotta, T. Boku, and D. Takahashi. Emprical study on reducing energy of parallel programs using slack reclamation by DVFS in a power-scalable high performance cluster. In *IEEE Cluster Computing, 2006*, pages 1–10, Sept 2006.

[46] David Kirk and Wen-mei Hwu. *Programming massively parallel processors hands-on with CUDA*. Morgan Kaufmann Publishers, 1st edition, 2010.

[47] Etienne Le Sueur and Gernot Heiser. Dynamic voltage and frequency scaling: The laws of diminishing returns. In *Proceedings of the 2010 Workshop on Power Aware Computing and Systems (HotPower'10)*, Vancouver, Canada, October 2010.

[48] D. Li and J. Wu. Minimizing energy consumption for frame-based tasks on heterogeneous multiprocessor platforms. *Parallel and Distributed Systems, IEEE Transactions on*, 2014.

[49] Min Yeol Lim, Vincent W. Freeh, and David K. Lowenthal. Adaptive, transparent frequency and voltage scaling of communication phases in MPI programs. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, New York, NY, USA, 2006. ACM.

[50] Ryan Luley, Courtney Usmail, and Mark Barnell. Energy efficiency evaluation and benchmarking of AFRL's condor high performance computer. Technical report, DTIC Document, 2011.

[51] Kai Ma, Xue Li, Wei Chen, Chi Zhang, and Xiaorui Wang. Greengpu: A holistic approach to energy efficiency in gpu-cpu heterogeneous architectures. In *Parallel Processing (ICPP), 2012 41st International Conference on*, pages 48–57, September 2012.

[52] Konrad Malkowski. *Co-adapting scientific applications and architectures toward energy-efficient high performance computing*. PhD thesis, The Pennsylvania State University, USA, 2009.

[53] Rajkumar Buyya Mark Baker, Amy Apon and Hai Jin. Cluster computing and applications, 2000.

[54] Daniel Minoli. *A Networking Approach to Grid Computing*. Wiley-Interscience, 2004.

[55] Naveen Muralimanohar, Karthik Ramani, and Rajeev Balasubramonian. Power efficient resource scaling in partitioned architectures through dynamic heterogeneity. In *In Proceedings of ISPASS*, 2006.

[56] Rahul Nagpal and Y.N. Srikant. Exploring energy-performance trade-offs for heterogeneous interconnect clustered vliw processors. In Yves Robert, Manish Parashar, Ramamurthy Badrinath, and ViktorK. Prasanna, editors, *High Performance Computing - HiPC 2006*, volume 4297, pages 497–508. Springer Berlin Heidelberg, 2006.

[57] NASA Advanced Supercomputing Division. NAS parallel benchmarks, March 2012.

[58] Dianne P. O'Leary and Robert E. White. Multi-splittings of matrices and parallel solution of linear systems. *SIAM Journal on Algebraic Discrete Methods*, 6(4):630–640, 1985.

[59] A.-C. Orgerie, L. Lefevre, and J.-P. Gelas. Save watts in your grid: Green strategies for energy-aware framework in large scale distributed systems. In *Parallel and Distributed Systems, 2008. ICPADS '08. 14th IEEE International Conference on*, pages 171–178, Dec 2008.

[60] David Padua. *Encyclopedia of Parallel Computing*. Springer Publishing Company, Incorporated, 2011.

[61] David A. Patterson and John L. Hennessy. *In Praise of Computer Architecture: A Quantitative Approach Fourth Edition*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.

[62] David A. Patterson and John L. Hennessy. *Computer Organization and Design, Fourth Edition: The Hardware/Software Interface (The Morgan Kaufmann Series in Computer Architecture and Design)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 4th edition, 2012.

[63] J. Peraza, A. Tiwari, M. Laurenzano, Carrington L., and Snavely. PMaC's green queue: a framework for selecting energy optimal DVFS configurations in large scale MPI applications. *Concurrency and Computation: Practice and Experience*, pages 1–20, 2012.

[64] G. Prinslow. Overview of performance measurement and analytical modeling techniques for multi-core processors, 2011.

[65] C.E. Ramamonjisoa, L. Ziane Khodja, D. Laiymani, A. Giersch, and R. Couturier. Simulation of asynchronous iterative algorithms using simgrid. In *High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC,CSS,ICESS), 2014 IEEE Intl Conf on*, pages 890–895, Aug 2014.

[66] Thomas Rauber and Gudula Rünger. Analytical modeling and simulation of the energy consumption of independent tasks. In *Proceedings of the Winter Simulation Conference*, WSC '12, pages 245:1–245:13. Winter Simulation Conference, 2012.

[67] Thomas Rauber and Gudula Rünger. *Parallel programming : for multicore and cluster systems*. Springer-Verlag, Berlin, 2010.

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

[69] Nikzad Babaii Rizvandi, Javid Taheri, and Albert Y. Zomaya. Some observations on optimal frequency selection in DVFS-based energy consumption minimization. *J. Parallel Distrib. Comput.*, 71(8):1154–1164, August 2011.

[70] Gustavo Rostirolla, Rodrigo Da Rosa Righi, Vinicius Facco Rodrigues, Pedro Velho, and Edson Luiz Padoin. Greenhpc: a novel framework to measure energy consumption on hpc applications. In *2015 Sustainable Internet and ICT for Sustainability, SustainIT 2015, Madrid, Spain, April 14-15, 2015*, pages 1–8. IEEE, 2015.

[71] B. Rountree, D.K. Lowenthal, S. Funk, Vincent W. Freeh, B.R. De Supinski, and M. Schulz. Bounding energy consumption in large-scale MPI programs. In *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*, pages 1–9, November 2007.

[72] Vijayalakshmi Saravanan, Alagan Anpalagan, and Isaac Woungang. An energy-delay product study on chip multi-processors for variable stage pipelining. *Human-centric Computing and Information Sciences*, 5(1), 2015.

[73] D. Shelepov and A. Fedorova. Scheduling on heterogeneous multicore processors using architectural signatures. In *Workshop on the Interaction between Operating Systems and Computer Architecture, in conjunction with ISCA*, 2008.

[74] Hao Shen, Jun Lu, and Qinru Qiu. Learning based DVFS for simultaneous temperature, performance and energy management. In *ISQED*, pages 747–754, 2012.

[75] V. Spiliopoulos, S. Kaxiras, and G. Keramidas. Green governors: A framework for continuously adaptive dvfs. In *International Green Computing Conference and Workshops (IGCC)*, pages 1–8, July 2011.

[76] William Stallings. *Computer Organization and Architecture (4th Ed.): Designing for Performance*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[77] Cheikhou Thiam, Georges Da Costa, and Jean-Marc Pierson. Energy aware clouds scheduling using anti-load balancing algorithm : EACAB. In *3rd International Conference on Smart Grids and Green IT Systems (SMARTGREENS 2014)*, pages pp. 82–89, Barcelona, Spain, April 2014.

[78] Ryoji Tsuchiyama, Takashi Nakamura, Takuro Iizuka, Akihiro Asahara, Satoshi Miki, Satoru Tagawa, and Satoru Tagawa. *The OpenCL Programming Book*. Fixstars Corporation, 1st edition, 2010.

[79] Abhinav Vishnu, Shuaiwen Song, Andres Marquez, Kevin Barker, Darren Kerbyson, Kirk Cameron, and Pavan Balaji. Designing energy efficient communication runtime systems: a view from pgas models. *The Journal of Supercomputing*, 63(3):691–709, 2013.

[80] John von Neumann. First draft of a report on the edvac. *IEEE Ann. Hist. Comput.*, 15(4):27–75, October 1993.

[81] Lizhe Wang, Samee U. Khan, Dan Chen, Joanna Kołodziej, Rajiv Ranjan, Cheng zhong Xu, and Albert Zomaya. Energy-aware parallel task scheduling in a cluster. *Future Generation Computer Systems*, 29(7):1661 – 1670, 2013.

[82] Che Wun Chiou Wen-Yew Liang, Po-Ting Lai. An energy conservation dvfs algorithm for the android operating system. *The Journal of Convergence*, 1(1):93–100, 2010.

[83] Fen Xie, M. Martonosi, and S. Malik. Bounds on power savings using runtime dynamic voltage scaling: an exact algorithm and a linear-time heuristic approximation. In *Proceedings of the 2005 International Symposium on Low Power Electronics and Design*, pages 287–292, Aug 2005.

[84] Chee Shin Yeo, Rajkumar Buyya, Hossein Pourreza, M. Rasit Eskicioglu, Peter Graham, and Frank Sommers. Cluster computing: High-performance, high-availability, and high-throughput processing on a network of computers. In Albert Y. Zomaya, editor, *Handbook of Nature-Inspired and Innovative Computing*, pages 521–551. Springer, 2006.

[85] M. Zapater, O. Tuncer, J.L. Ayala, J.M. Moya, K. Vaidyanathan, K. Gross, and A.K. Coskun. Leakage-aware cooling management for improving server energy efficiency. *Parallel and Distributed Systems, IEEE Transactions on*, 26(10):2764–2777, Oct 2015.

[86] Marina Zapater, Jose L. Ayala, José M. Moya, Kalyan Vaidyanathan, Kenny Gross, and Ayse K. Coskun. Leakage and temperature aware server control for improving energy efficiency in data centers. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 266–269, San Jose, CA, USA, 2013.

[87] Yifan Zhu and Frank Mueller. Exploiting synchronous and asynchronous DVS for feedback EDF scheduling on an embedded platform. *ACM Transactions on Embedded Computing*, 7(1), 2007.

[88] Jianli Zhuo and Chaitali Chakrabarti. Energy-efficient dynamic task scheduling algorithms for dvs systems. *ACM Trans. Embed. Comput. Syst.*, 7(2):17:1–17:25, January 2008.

Document generated with LaTeX and:
the LaTeX style for PhD Thesis created by S. Galland — http://www.multiagent.fr/ThesisStyle
the tex-upmethodology package suite — http://www.arakhne.org/tex-upmethodology/

Abstract:

This thesis, presents the algorithms developed to optimize the energy consumption and the performance of synchronous and asynchronous message passing applications with iterations running over clusters or grids. The energy consumption and performance models for each type of parallel application predicts its execution time and energy consumption for any selected frequency according to the characteristics of both the application and the architecture executing this application.
The contribution of this thesis can be divided into three parts: Firstly, optimizing the trade-off between the energy consumption and the performance of the message passing applications with synchronous iterations running over homogeneous clusters. Secondly, adapting the energy and performance models to heterogeneous platforms where each node can have different specifications such as computing power, energy consumption, available frequency gears or network's latency and bandwidth. The frequency scaling algorithm was also modified to suit the heterogeneity of the platform. Thirdly, the models and the frequency scaling algorithm were completely rethought to take into considerations the asynchronism in the communication and computation. All these models and algorithms were applied to message passing applications with iterations and evaluated over either SimGrid simulator or Grid'5000 platform. The experiments showed that the proposed algorithms are efficient and outperform existing methods such as the energy and delay product. They also introduce a small runtime overhead and work online without any training or profiling.

Keywords: Dynamic voltage and frequency scaling, Grid computing, Energy optimization, parallel applications with iterations and online frequency scaling algorithm.

Résumé :

Cette thèse présente des algorithmes développés pour optimiser la consommation d'énergie et les performances des applications parallèles avec des itérations synchrones et asynchrones sur des clusters ou des grilles. Les modèles de consommation d'énergie et de performance proposés pour chaque type d'application parallèle permettent de prédire le temps d'exécution et la consommation d'énergie d'une application pour toutes les fréquences disponibles. La contribution de cette thèse peut être divisé en trois parties. Tout d'abord, il s'agit d'optimiser le compromis entre la consommation d'énergie et les performances des applications parallèles avec des itérations synchrones sur des clusters homogènes. Deuxièmement, nous avons adapté les modèles de performance énergétique aux plates-formes hétérogènes dans lesquelles chaque noeud peut avoir des spécifications différentes telles que la puissance de calcul, la consommation d'énergie, différentes fréquences de fonctionnement ou encore des latences et des bandes passantes réseaux différentes. L'algorithme d'optimisation de la fréquence CPU a également été modifié en fonction de l'hétérogénéité de la plate-forme. Troisièmement, les modèles et l'algorithme d'optimisation de la fréquence CPU ont été complètement repensés pour prendre en considération les spécificités des algorithmes itératifs asynchrones. Tous ces modèles et algorithmes ont été appliqués sur des applications parallèles utilisant la bibliothèque MPI et ont été exécutés avec le simulateur Simgrid ou sur la plate-forme Grid'5000. Les expériences ont montré que les algorithmes proposés sont plus efficaces que les méthodes existantes. Ils n'introduisent qu'un faible surcoût et ne nécessitent pas de profilage au préalable car ils sont exécutés au cours du déroulement de l'application.

Mots-clés : l'ajustement dynamique de la tension et de la fréquence d'un processeur, Grille de calcul, Optimisation de l'énergie, applications parallèles avec des itérations et en ligne algorithme fréquence ajustement.

■ École doctorale SPIM 16 route de Gray F - 25030 Besançon cedex

■ tél. +33 (0)3 81 66 66 02 ■ **ed-spim@univ-fcomte.fr** ■ www.ed-spim.univ-fcomte.fr