Optimization of energy and performance of applications on heterogeneous micro-servers

Massinissa Ait Aba

THÈSE DE DOCTORAT
DE SORBONNE UNIVERSITÉ

Spécialité

Informatique

École doctorale Informatique, Télécommunications et Èlectronique (Paris)

Présentée par

Massinissa AIT ABA Pour obtenir le grade de

DOCTEUR de SORBONNE UNIVERSITÉ

Sujet de la thèse :

Optimisation de l'énergie et de la performance d'applications sur des micro-serveurs hétérogènes

Directrice de thèse : Alix MUNIER KORDON

Encadrante de thèse : Lilia ZAOURAR

Thèse présentée et soutenue à Palaiseau, le 04/06/2020

Devant le jury composé de :

Mme. Safia KEDAD-SIDHOUM	Examinatrice
M. Lionel LACASSAGNE	Examinateur
M. Loris MARCHAL	Rapporteur
Mme. Alix MUNIER KORDON	Directrice
M. Jean-Marc NICOD	Rapporteur
M. Guillaume PALLEZ	Examinateur
M. Denis TRYSTRAM	Examinateur
Mme. Lilia ZAOURAR	Encadrante

Remerciements

Je tiens tout d'abord à remercier grandement ma directrice de thèse, Alix Munier Kordon et mon encadrante Lilia Zaourar, pour toute l'aide qu'elles m'ont apportée tout au long de cette thèse. Qu'elles soient aussi remerciées pour leurs disponibilités permanentes et pour les nombreux encouragements et judicieux conseils qui ont contribué à alimenter ma réflexion.

Merci à monsieur Lionel LACASSAGNE pour avoir accepté de présider mon jury de thèse.

Je voudrais également remercier mes deux rapporteurs, Loris MARCHAL et Jean-Marc NICOD, d'avoir accepté de relire en détail ce manuscrit.

Merci à madame Safia KEDAD-SIDHOUM et messieurs Guillaume PALLEZ et Denis TRYSTRAM de l'attention qu'ils portent à mon travail en acceptant d'être membres du jury.

Le travail qui est présenté dans ce manuscrit a été effectué au CEA List au sein du Laboratoire LCE. Aussi, merci à messieurs Nicolas VENTROUX, David RAPHAEL et Christian GAMRAT, de m'avoir accueilli et mis à ma disposition tous les moyens nécessaires aussi bien techniques que financiers, afin de mener à bien ce travail.

J'aimerais ensuite remercier toutes les personnes que j'ai rencontrées au CEA tout au long de ma thèse. En particulier, je tiens à remercier mes anciens collègues Jason L., Dina Y., Nermine A., Erwan L., Farouk H., Divya G., François G., Sergiu K., Amir C., Aymen B., David B., Kods T., Daniel V., Philippe G., Emine L., Farid L., Martin Z. pour toutes les discussions intéressantes que nous avons pu avoir.

Je souhaite remercier également les différentes personnes que j'ai pu croiser durant ces dernières années et qui ont su par leurs questions ou leurs conseils me permettre de toujours pousser un peu plus mon raisonnement.

Je voudrais également remercier toutes les personnes extérieures du domaine universitaire qui m'ont, à leur façon, apporté leur aide. En premier lieu, je voudrais exprimer ma profonde gratitude à mes parents, Malek et Hakima, ainsi que mon frère Gaya et ma soeur Kahina, et à tous les membres de ma famille (Fatima, Mokrane, Mustapha, Omar, Djilali...), qui ont su croire en moi et qui m'ont apporté toute leur aide quand j'en ai eu besoin. Un grand merci à ma Sosso pour son soutien inconditionnel. Je souhaite également remercier tous mes amis de grande date qui ont su m'apporter confiance et écoute à tous les moments. Ainsi, je remercie particulièrement Abderrezak, Sofiane, Abderrahmane, Zineb, Mounia, Amine, les deux Mehdi, Mustapha, Yanis, Raouf, Fatima, Ahmed, Ilyes, Nacer, Said, Ali, Abdesslem, Asmaa, Sedik, Hacene et tous les autres.

Enfin, merci à toutes celles et ceux qui ont pris le temps de venir voir ma soutenance, à celles et ceux qui m'ont aidé à organiser le pot.

Contents

Symbols and Acronyms

Symbols

n	Number of tasks
m	Number of processing elements
t_i	Task indexed by i
$T = \{t_1, t_2, ..., t_n\}$	Set of the tasks
p_j	Processing element number j
$M = \{p_1, p_2, ..., p_m\}$	Set of the processing elements
$\theta(t_i)$	Processor assignment of t_i(ex: $\theta(t_i) = p_1$ means that t_i is assigned to p_1)
$\tau(r)$	Type of p_r(ex: $\tau(r) = \mathcal{A}$ if $r \leqslant \ell$ and $\tau(r) = \mathcal{B}$ if $r > \ell$)
$\mathcal{G} = (V, E)$	A directed acyclic graph
V	The set of nodes in $\mathcal{G}$
E	The set of arcs in $\mathcal{G}$
$c_{i,j}$	Communication rate between $(t_i, t_j) \in E$, if $\theta(t_i) = p_l$ and $\theta(t_j) = p_k$
$cm_{i,l,j,k}$	Communication cost between $(t_i, t_j) \in E$, if $\theta(t_i) = p_l$ and $\theta(t_j) = p_k$
S_i	Starting execution time of the task t_i
C_i	Completion time of the task t_i
$\Gamma^+(t_i)$	The set of the immediate successors of the task t_i
$\Gamma^-(t_i)$	The set of the immediate predecessors of the task t_i

Acronyms

DAG	Directed Acyclic Graph
HPC	High-performance computing
CPU	Central Processing Unit
GPU	Graphics Processing Unit
FPGA	Field-Programmable Gate Array
DVFS	Dynamic voltage and frequency scaling
DPM	Dynamic power management
EFT	Earliest Finish Time
MIP	Mixed Integer Program
NLMIP	Non-Linear Mixed Integer Program

Résumé en français

Les applications récentes dans l'industrie ou dans la recherche nécessitent souvent des calculs massifs. Ainsi, les applications deviennent plus exigeantes en vitesse de calcul, ce qui engendre une très grande consommation énergétique des plateformes matérielles. Les plateformes de calcul hétérogènes offrent un bon compromis avec une puissance de calcul importante tout en préservant l'énergie consommée pour l'exécution d'applications parallèles de hautes performances. Elles représentent donc de nos jours des moyens de calcul intéressants. Afin de profiter des avantages offerts par l'hétérogénéité en termes de performance, la gestion efficace et automatique des ressources de calcul est de plus en plus importante pour exécuter des applications parallèles. Ces nouvelles architectures ont ainsi donné lieu à de nouveaux problèmes d'ordonnancement qui allouent et séquencent les calculs sur les différentes ressources en optimisant un ou plusieurs critères.

L'objectif de cette thèse est de déterminer un ordonnancement efficace d'une application parallèle sur un système de ressources hétérogènes afin de minimiser le temps d'exécution total (makespan, C_{max}) de l'application tout en respectant une contrainte d'énergie.

Deux classes de plateformes hétérogènes ont été considérées dans notre travail : des architectures totalement *hétérogènes* qui combinent plusieurs éléments de traitement (CPUs, GPUs, FPGAs), et des plateformes *hybrides* limitées à deux types de processeurs (CPU + GPU par exemple) en très grand nombre.

Deux modèles d'exécution ont été également considérés. Le modèle *consistant*, où chaque processeur P_j est caractérisé par une fréquence d'exécution f_j, et chaque tâche t_i est caractérisée par un poids w_i. Le temps d'exécution et l'énergie associée à une tâche t_i allouée à P_j sont respectivement $\frac{w_i}{f_j}$ et $w_i * f_j^2$. Le modèle *inconsistant*, où il n'y a pas d'hypothèse sur les temps d'exécution des tâches et les consommations d'énergie sur les différents processeurs. De plus, dans ce travail, des délais de communication sont pris en compte entre les processeurs et les tâches pour les deux modèles.

Nous proposons plusieurs stratégies d'ordonnancement d'applications sur les deux plateformes avec les deux modèles d'exécution. Nous avons d'abord étudié le cas particulier d'une application représentée par une chaîne de tâches. Nous proposons un algorithme polynomial avec une garantie de performance par rapport à la solution optimale. Ensuite, nous nous sommes concentré sur un cas plus général avec des applications représentées par des graphes acycliques quelconques. Nous avons d'abord étudié le problème pour les plateformes à nombre illimité de processeurs. Nous prouvons que le problème est NP-complet, et proposons des algorithmes polynomiaux pour des cas particuliers de graphes. Un algorithme non polynomial avec une garantie de performance de 6 est ensuite proposé pour le problème d'ordonnancement sur ressources limitées avec et sans contrainte d'énergie. Enfin, nous modifions cet algorithme pour obtenir un algorithme polynomial mais avec une garantie de performance relative.

Les expériences préliminaires des algorithmes proposés en utilisant différentes applications et des plateformes de tailles différentes ont donné de bons résultats par rapport aux méthodes existantes dans la littérature.

Abstract

Recent applications, both in industry and research often need massive calculations. They have different hardware requirements in terms of computing speed, which leads to very high energy consumption of hardware platforms. Heterogeneous computing platforms offer a good compromise with high computing power while preserving the energy consumed to run high-performance parallel applications. They are therefore nowadays an interesting computing resource. In order to exploit the advantages offered by heterogeneity in terms of performance, efficient and automatic management of computing resources is becoming increasingly important to execute parallel applications. These new architectures have thus given rise to new scheduling problems that allocate and sequence calculations on the different resources by optimizing one or more criteria.

The objective of this thesis is to determine an efficient scheduling of a parallel application on a heterogeneous resource system in order to minimize the total execution time (makespan, C_{max}) of the application while respecting an energy constraint.

Two classes of heterogeneous platforms have been considered in our work: *fully heterogeneous* architectures that combine several processing elements (CPUs, GPUs, FPGAs), and *hybrid platforms* limited to two types of processors (CPU + GPU for example).

Two execution models were also considered. In the *consistent* one, each processor P_j is characterized by an execution frequency f_j, and each task t_i is characterized by a weight w_i. The execution time and the energy associated to a task t_i allocated to PE_j are respectively $\frac{w_i}{f_j}$ and $w_i * f_j^2$. For the *inconsistent* model, there is no assumption for the execution time of tasks on the different processors. In addition, communication delays are taken into account between processors and tasks for both models.

We propose several application scheduling strategies on both platforms with both execution models. We first studied the particular case of an application represented by a chain of tasks. We proposed a polynomial algorithm with a performance guarantee.

Then, we focused on a more general case with applications represented by general directed acyclic graphs. We first studied the problem for platforms with an unlimited number of processors. We proved that the problem is NP-complete, and proposed polynomial algorithms for some particular graph cases. A non-polynomial algorithm with a performance guarantee of 6 is then proposed for the scheduling problem on limited resources with and without energy constraint. Finally, we modify this algorithm to obtain a polynomial algorithm, but with a relative performance guarantee. Preliminary experiments with the proposed algorithms using different applications and platforms of different sizes have shown good results compared to existing methods in the literature.

Introduction

Today, our daily life requires massive calculations on different computing systems (desktop, data centers) to perform various needs such as medical simulations and physical simulations. These applications can be executed on a single processing element, but they take a long runtime. In order to improve the performance of these applications, the past few years have seen an increasing demand for developing efficient large computing resources. The goal is to process large amount of computations related to high performance parallel applications.

Thus, heterogeneous computing systems become a popular and powerful commercial platform, containing several heterogeneous processing elements such as Central Processing Unit (CPU), Graphics Processing Unit (GPU) and some Field Programmable Array (FPGA) with different computational characteristics. These processing elements are heterogeneous because they have different characteristics in terms of execution time and energy consumption. The *Top500*[1] list, which has been updated semiannually for the past decade, ranks the 500 most powerful computers installed worldwide. However, using efficiently these platforms become very complicated and very challenging. In fact, with the complexity of applications and architectures, it becomes increasingly difficult to distribute efficiently the tasks of an application. The aim to execute it in parallel by limiting unnecessary communication delays. Otherwise, more time is spent communicating than computing. Several questions arise then : is parallelism useful? Should we rather execute all the application on a single processing element? When should we parallelize? Which part of the application will be executed on which processor?

More than a simple load balancing problem, heterogeneity leads to consider efficient scheduling techniques to take into account the different resources specificities. Consequently, more and more attention has been focused on scheduling techniques for solving the problem of optimizing the execution of parallel applications on heterogeneous computing systems [1–5].

The most common objective function of task scheduling problems is makespan. However, energy consumption is also an important issue to be considered. Indeed, the resulting energy consumption of these platforms is very high and its increase must be kept reasonable. As an example, the U.S. data centers consumed an estimated 70 billion kWh (about 1.8% of total U.S. electricity consumption) [6]. Thus, it is time to invest in green computing, and computing servers must be built with energy-aware resource management. This focus on energy efficiency must also have as much as possible little impact on performance as possible. Thus, heterogeneous systems offer the opportunity to achieve high performance while saving energy [7] and give an attractive alternative to massively parallel servers.

The objective of this thesis is to find a generic approach to schedule parallel applications presented by graphs of type DAG (Directed Acyclic Graph) on a heterogeneous resource system in order to minimize both the total execution time (makespan) and the energy consumption. For this purpose, we introduce a constraint on the total energy consumed by the platform. The aim is then to propose efficient scheduling methods to minimize execution time while respecting an energy constraint. In addition, these methods must guarantee the quality of the solution in the worst case compared to the optimal solution.

We now summarize the different chapters of this thesis as follow. We give a concluding

[1] Top500.org ranking. URL https://www.top500.org/lists/2018/11/

chapter for the whole thesis at the end.

Chapter 1 Preliminary notions and context

The aim of this chapter is to situate the context of this thesis. We first give a brief overview of heterogeneous platforms, in particular fully heterogeneous and hybrid platforms that are studied in this work. We then discuss the two main scheduling strategies in the literature, namely dynamic and static scheduling. Then, we detail the execution model of the applications and the energy models that will be used in this thesis. Finally, we introduce some helpful notations for the rest of the manuscript.

Chapter 2 Scheduling sequential applications (chain of tasks) on heterogeneous platforms

This chapter presents an efficient approximation algorithm to solve a task scheduling problem on heterogeneous platforms for the particular case of linear chains of tasks (sequential applications). The objective is to minimize the total execution time (makespan) respecting an energy constraint on the total energy consumed by the system. The main contribution of this chapter is an algorithm which provides a solution with small running time, and also guarantees the quality of the solution obtained compared to the optimal solution.

Chapter 3 Scheduling parallel applications on hybrid platforms with an unlimited number of processors

This chapter addresses the problem of scheduling parallel applications onto a particular case of HPC which is known as hybrid platforms. Specifically, our platform type consists of two types of processing elements (e.g. CPU+GPU), each with an unbounded number of resources. We have shown the intractability of the problem and proved that there does not exist 3/2-approximation algorithms unless P=NP. We further provide some polynomial time algorithms for special cases of graphs.

Chapter 4 Hybrid platform with a limited number of processors

In this chapter, we suppose that our hybrid platform is composed of a limited number of two types of processing elements. First, we propose a non polynomial two-phase 6-approximation algorithm named List Scheduling algorithm With Pre-Allocation (LSWPA). The first phase consists in solving two models to find the type of processor assigned to execute each task (mapping). In the second phase, we compute the start execution time of each task to generate a feasible schedule (assignment). Then we propose a polynomial three-phase algorithm named Polynomial List Scheduling algorithm With Pre-Allocation (PLSWPA). The first two phases consist in solving linear models to find the mapping of the tasks. In the third phase, we compute the assignment of the tasks.

Chapter 5 Hybrid platform with a limited number of processors with energy constraint

This chapter presents two efficient algorithms to solve the problem of scheduling parallel applications on hybrid platforms with communication delays. The objective is to minimize the total

execution time (makespan) respecting an energy constraint. First, we modify LSWPA algorithm to provide a 6-approximation algorithm LSWPAe with two phases. In the first phase, we add some constraints to the model used for LSWPA algorithm to respect the energy constraint. The second phase is the same as the second phase of LSWPA algorithm.

Then, we modify PLSWPA algorithm to provide a polynomial three-phase algorithm PLSWPAe. The first two phases consist in solving linear models to find the type of processor assigned to execute each task. We obtain a mapping that consumes at most twice the amount of authorized energy. In the third phase, we compute the start execution time of each task to generate a feasible schedule.

Chapter

1 Preliminary notions and context

Chapter content

1.1 Introduction

The objective of this thesis is to determine an efficient scheduling of parallel applications on a heterogeneous resource system in order to minimize the total execution time (makespan) of the application while respecting an energy constraint. The aim of this chapter is to describe the context of this thesis. We first give a brief overview of heterogeneous platforms, in particular fully heterogeneous and hybrid ones. Then, we discuss the two main scheduling strategies in the literature, namely *dynamic* and *static* scheduling. After this, we detail the execution model of the applications and the energy models that will be used in the rest of this thesis. Finally, a conclusion of the chapter is given after some helpful notations for the rest of the manuscript.

1.2 Platform architectures

Nowadays, numerous services need the execution of complex applications, such as weather forecasting, search engines and big data medical analyzes. These applications are studied, tested and simulated in data centers with massively parallel computing systems. Unfortunately, the resulting energy consumption is very high and its increase must be kept reasonable. As an example, the U.S. data centers consumed an estimated 70 billion kWh (about 1.8% of total U.S. electricity consumption) [6]. Thus, it is time to invest in green computing, and computing servers must be built with energy-aware resource management. This focus on energy efficiency must also have as little impact on performance as possible. Heterogeneous systems offer the opportunity to achieve high performance while saving energy [7] and give an attractive alternative to massively parallel

servers. The processing elements have different characteristics in terms of execution time and energy consumption. This heterogeneity offers several scheduling options, which makes it easier to find efficient scheduling. Two classes of heterogeneous platforms have been considered in this thesis, the fully heterogeneous and hybrid platforms as detailed below.

1.2.1 Fully heterogeneous platform

Recent High Performance Computing (HPC) systems have heterogeneous architectures using accelerators (GPUs such as NVIDIA Tesla or manycore architectures like Intel Xeon Phi). Even more heterogeneous systems can be found in micro-server systems which are designed to host fairly high computing power in a small form factor such as Mont-Blanc [8] and Christmann RECS|BOX [9, 10].

This kind of platforms allows tight integration of general purpose computing architectures such as CPUs, embedded CPUs with possibly accelerators (e.g. ARM-based SoCs with GPUs or FPGAs), GPUs, FPGAs. This enables the realization of a truly heterogeneous hyperscale server architecture.

Figure 1.1 presents an example of a sample configured server, which can be composed of two Intel Core-I7 4700EQ nodes (16GB memory), one NVIDIA Tesla K80 GPU, two baseboards with four Christmann Apalis ARM nodes (Samsung Exynos 5250 - dual ARM A15 and Mali-T604MP4) and finally a prototype FPGA board from the University of Bielefeld (Xilinx Zynq XC7Z45 - dual ARM A9 and reconfigurable logic).

Figure 1.1: *Picture of a RECS|BOX server composed of two Intel CPUs, one NVIDIA Tesla K80, eight Samsung Exynos ARM CPUs and a Xilinx Zynq 7045.*

1.2.2 Hybrid platform (GPU/CPU)

With the increasing popularity of deep learning in data centers (Facebook, Twitter, Microsoft, Google, etc.) and video games, GPUs become quite common and powerful, enclosing great computation capability. In fact, in several applications, we observe an acceleration of the execution time of tasks if they are executed on a GPU compared to their execution time on a CPU. Thus, Hybrid platforms are increasingly used in the field of HPC, where the number of platforms of the *Top500* equipped with accelerators has significantly increased during the last years like TGCC Curie supercomputer[1].

The NVIDIA Drive PX2 platform (see Figure 1.2) is an interesting example in this area, aimed at providing autonomous car and driver assistance functionality. The platform has an NVIDIA discrete GPU and an integrated NVIDIA GPU. The discrete GPU is relatively more powerful than the integrated GPU. However, the integrated GPU has operators optimized for working

[1]Tgcc Curie supercomputer, http://www-hpc.cea.fr/en/complexe/tgcc-curie.htm

with reduced 16-bit floating (half-precision) accuracy. Using these operators almost doubles the execution speed on this GPU compared to a 32-bit floating (single-precision) implementation.

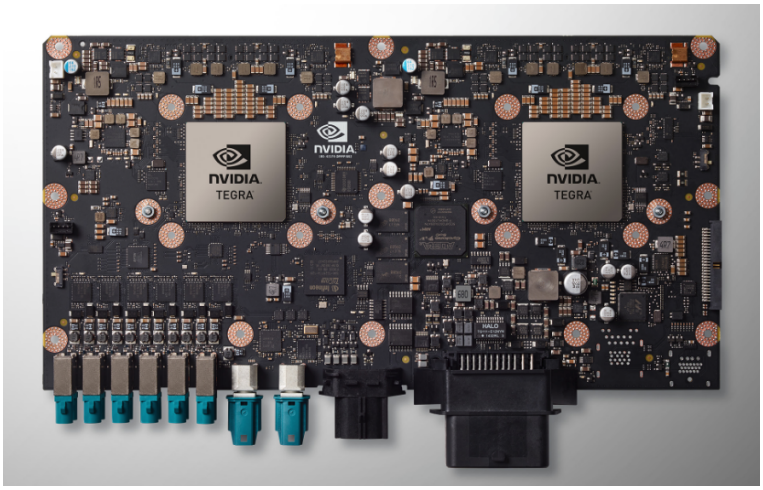


Figure 1.2: *A NVIDIA Drive PX2 platform composed of 4 Denver cores, 8 ARM A57 cores and a 2 GPUs Pascal-based.*

In this thesis, we consider the more general case where the relation between the two types of resources can vary for different tasks. A scheduling method for the fully heterogeneous platforms can also be applied to hybrid platforms, but one can do better for the hybrid platform in terms of algorithms. Indeed, we will see in chapters 3, 4 and 5, that the particularity of this platform (two types of processing elements) gives the possibility to provide more efficient scheduling algorithms.

1.3 Scheduling strategies

Taking advantage of the heterogeneous systems described above requires efficient use of resources. In a heterogeneous environment, tasks have different execution times and power consumption that depend on computing resources. The performance and the energy consumption of the system can be changed by using different resource allocations. Thus, the exploitation of these heterogeneous platforms raises new challenges in terms of application management optimization on available resources. An important research challenge is how to assign the different tasks composing an application to the available resources in order to maximize some performance criterion of the heterogeneous architecture [11].

Tackling this challenge consists in determining efficient strategies to exploit these heterogeneous platforms by finding the best allocation of application tasks to optimize the performance per watt ratio with respect to various constraints. For this, having the right application and hardware models is paramount. On the application side, some high level optimizations have been developed to reduce the execution time [12] and the power consumption [13]. These optimizations provide more freedom for Design Space Exploration [14] and for embedded applications [15–17].

In this thesis, we consider a parallel application represented by a Directed Acyclic Graph (DAG) with precedence constraints between the tasks. The *mapping* problem consists in the assignment of tasks to a set of processing elements. The *scheduling* problem consists in sequencing the order of task execution for each processing element such that precedence relationships between tasks are not violated, and orchestrating inter-node data transfers to optimize the performance per watt ratio.

Static scheduling involves mapping applications in an offline phase while dynamic mapping is done online. In both cases, the mapping problem has been shown, in general, to be NP-

complete [18, 19]. Thus, the development of heuristic techniques to find near-optimal solutions for the mapping problem is an active area of research.

Static and Dynamic scheduling:

In dynamic scheduling, the mapping and scheduling are done online, and based on dynamic information such as the set of available tasks and the state of the resources. In contrast, static scheduling means that the processor allocation, often called mapping, and the ordering of the tasks are determined in an offline phase and based on the whole task graph [20].

The advantage of static scheduling is that it can take all the tasks of the application into consideration in its scheduling decisions by including the dependences and communications among the tasks. However, scheduling on shared resources makes the use of static scheduling strategies more complex [21]. Indeed, it is difficult to precisely predict task execution times and communication costs between processing elements.

Recently, to handle these limitations, several dynamic methods have been proposed such as StarPU [22] and PaRSEC [7]. These methods make their decisions based on the condition of the machine and all available tasks. Though they can lead to intensive use of some resources by often assigning computations to the best ones and make a poor use of "slower" ones like CPUs. Furthermore, most of task programming libraries do not take into account the energy criterion, and only focus on minimizing the completion time.

A deep analysis was realized in [23, 24] to compare static and dynamic strategies for task graph scheduling on heterogeneous platforms. It has been shown that static schedules may be robust to variations in execution times for some applications. Thus, an efficient method for executing applications on shared resources is a method that is based on both dynamic and static aspects at the same time.

Recently, [25] have shown the ability of hybrid dynamic-static strategies to simultaneously reduce the execution time and the amount of communications. The study shows the efficiency of static schedulers with the help of some techniques to fix a possible load imbalance encountered during the execution (when a processing element is idle with no remaining attributed tasks on its memory node). It allows to make the best use of highly heterogeneous platforms.

Thus, the aim of this thesis is to propose efficient static methods, so that their association with dynamics methods will give the most efficient solution as possible.

We propose several methods for the scheduling problem on parallel platforms. We compare the performance of these methods to the optimal solution and lower bounds. Furthermore, the methods proposed for the problem of scheduling without energy constraint will be compared to Heterogeneous Earliest Finish Time (HEFT) [26] which was used as a comparison method for several works. HEFT is a list based approach on two main phases. The first phase uses runtime costs and communication costs to calculate ranks. After rank calculation, the assignment to the processors will take place in the second phase using the earliest finish time of tasks. Each task is then assigned to the processor that produces the minimal completion time.

1.4 Application and platform models

In this thesis, we consider a fully connected heterogeneous multiprocessor platform in which $M = \{p_1, p_2, ..., p_m\}$ is a set of m heterogeneous processing elements (GPU, CPU, FPGA, ...) denoted by p_k, $k = \overline{1..m}$.

In the most general case, an application of n tasks can be represented by a Directed Acyclic Graph (DAG) $\mathcal{G} = (V, E)$ (see figure 1.3). V is the set of nodes in $\mathcal{G}$, and each node $t_i \in V$ represents a task. E is the set of arcs in $\mathcal{G}$, each arc represents a precedence constraint.

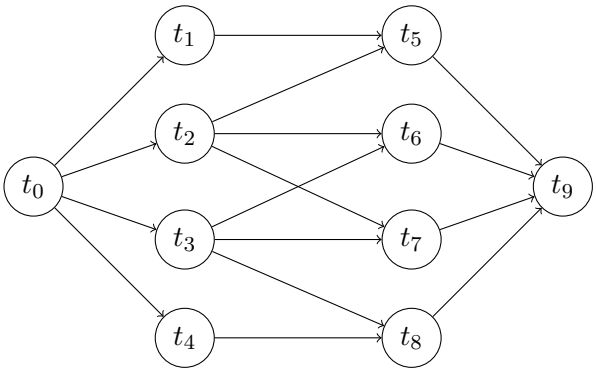


Figure 1.3: *General DAG graph.*

Each task t_i is characterized by its execution time $w_{i,k}$ and its energy consumption $e_{i,k}$ on a processing element p_k, $i = \overline{1..n}$, $k = \overline{1..m}$. Each arc $(t_i, t_j) \in E$ represents a precedence constraint between the tasks t_i and t_j (ex: data transfer). Each arc $(t_i, t_j) \in E$ is characterized by a value $c_{i,j}$ which expresses the communication rate between t_i and t_j. If t_i is executed on the processing element p_l and t_j on p_k and $l \neq k$, then the communication time between t_i and t_j is equal to $cm_{i,l,j,k} = \frac{c_{i,j}}{B_{l,k}} + L_{l,k}$ [10, 27]. $B_{l,k}$ represents the available bandwidth between p_l and p_k. This bandwidth can be the one from an Ethernet link between p_l and p_k. $L_{l,k}$ represents the latency between p_l and p_k. However, other communication models [28] may exist to calculate the value of $cm_{i,l,j,k}$, $i, j = \overline{1..n}$, $k, l = \overline{1..m}$. We suppose that $cm_{i,l,j,k} = cm_{i,k,j,l}$. Let C_i be the end date of the task t_i. For any arc $(t_i, t_j) \in E$, $C_i + w_{j,k} \leq C_j$ if both tasks are executed by the same processing element p_k. Otherwise, $C_i + w_{j,k} + cm_{i,k,j,l} \leq C_j$ where t_j is executed by the processing element p_k and t_j by $p_l \neq p_k$. We suppose that $cm_{i,k,j,l} = 0$ if $p_k = p_l$.

For any task $t_i \in V$, $\Gamma^+(t_i)$ (resp. $\Gamma^-(t_i)$) denotes the set of the immediate successors (resp. predecessors) of t_i. We consider in this thesis that a task t_i can be executed only after the completion time of all its predecessors.

Preemptive scheduling:

In scheduling, preemption is the act of temporarily interrupting the execution of a task on a processor p_j, and later resuming its execution on the same or on another processor. For the general case of applications represented by a DAG, the preemptive scheduling problem is NP-complete even without energy constraint [29].

We do not allow duplication of tasks or preemption. A task can be executed by all processing units. We denote by $\mathbb{E}$ the allowed quantity of energy consumed during the execution. The value $\mathbb{E}$ represents in our case an energy bound that should not be exceeded during the execution. The energy model used in this thesis is described below.

An application is usually represented by DAGs. Nevertheless, in some cases, it is possible for applications to take a particular form. For example, if an application includes only sequential tasks, it can be represented by a linear chain of tasks (see figure 1.5). Other applications may take other forms of graphs such as Serie-Parallel graphs (see figure 1.4), out-tree graphs (see figure 1.6) and bi-partite graph (see figure 1.7). These graph classes will be studied in Chapter 3. The fact that the precedence graph takes a particular form may transform the complexity of the problem. Most of the time, adding precedence constraints makes the problem harder [30].

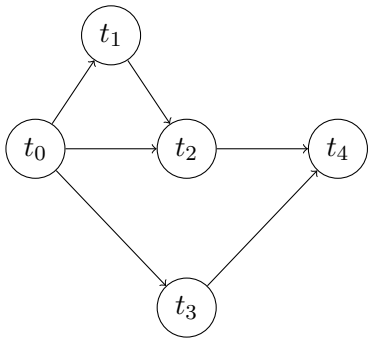


Figure 1.4: *A Serie-Parallel graph.*

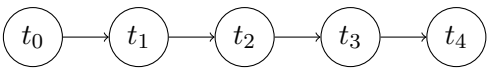


Figure 1.5: *Linear chain graph.*

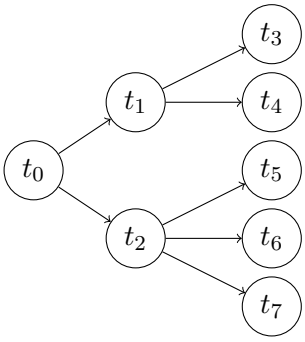


Figure 1.6: *An out-tree graph.*

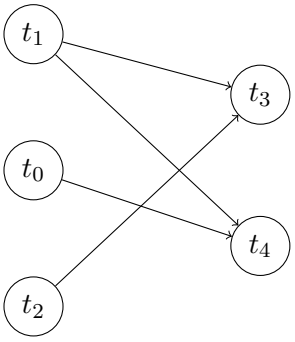


Figure 1.7: *A bi-partite graph*

Scheduling classification

The following classification is used to describe the scheduling problems treated in this thesis. A convenient notation for theoretic scheduling problems was introduced by Graham et al. [31], and updated by Brucker [32], where many scheduling problems can be described by a three field notation $\alpha|\beta|\gamma$ such that:

- α describes the machine environment.
- β describes the characteristics of the tasks.
- γ describes the objective criterion to be minimized.

One field can contain several proprieties. In this thesis, we use the following items:

- $\alpha = Pm$ if there are m parallel identical processing elements. Each task t_i is characterized by its execution time w_i, which is the same for all processing elements.
- $\alpha = Qm$ if there are m parallel processing elements with different given frequencies $F = \{f_1, f_2, ..., f_m\}$. Each task t_i is characterized by its weight w_i, and its execution time $w_{i,k} = \frac{w_i}{f_k}$ on a processing element p_k.
- $\alpha = Rm$ if there are m parallel unrelated processing elements. Each task t_i is characterized by its execution time on a processing element p_k equal to $w_{i,k}$.
- $\beta = prec$: precedence relations are given between tasks. If t_i precedes t_j, t_j may start its execution after the finish execution time of the tasks t_i.
- $\beta = com$: communication delays are given between processing elements.

- $\beta = pmtn$: tasks can be preempted and resumed possibly on another processing element.
- $\beta = chain$: if an application has a particular form of a linear chain of tasks.
- $\beta = \mathbb{E}$: if there is an allowed quantity of energy to not exceed during the execution.
- $\gamma = C_{max}$: the objective is to minimise the finish execution time of the application.

For example, $Pm|prec, \mathbb{E}|C_{max}$ is the scheduling problem on m identical machines that minimizes the completion time by respecting precedence constraints without communication costs while satisfying an energy constraint $\mathbb{E}$.

1.5 Energy model

The heterogeneous computation platform allows very powerful computation performance, but requires a significant amount of energy. Due to its key importance on performance, energy-aware scheduling problem on heterogeneous platform has been extensively studied [33, 34].

Minimizing the energy consummation of these platforms is linked to managing energy consumption in Complementary Metal Oxide Semiconductor (CMOS) circuits. The main power consumption in CMOS circuits for task execution is composed of dynamic energy consumption and static energy consumption [35]. Dynamic power dissipation is only consumed when there is switching activity as transistors switch states in the logic circuit. Static power is the part of power consumption that is independent of activity and due to leakage currents.

Numerous methods and models have been reported in the literature to minimize the energy consumption on a limited number of processing elements such as Dynamic Voltage and Frequency Scaling (DVFS) [36, 37] and Dynamic Power Management (DPM) [38, 39] as defined below.

1.5.1 DVFS and DPM techniques

DVFS technique reduces the dynamic power dissipation by exploring the trade-off between energy consumption and execution time by scaling down the supply voltage and frequency of a processor while tasks are running. DVFS technique reduces the static power dissipation only by scaling the voltage and the frequency of processors to their lowest voltage and lowest frequency.

The DPM policy is to exploit the available idle periods to reduce static power, where the processor must not stay in active mode for the entire scheduling period. It manages the processor transition from a low-power to an active mode to reduce the power consumption.

By using DVFS technique, the scheduling problem of minimizing the makespan remains NP-complete even without energy constraints. In fact, by ignoring the energy constraint, we can choose the voltage and frequency of the processors that generate the smallest execution time for each task. Then, we find the general scheduling problem where the execution time of each task is a fixed value, this problem is NP-complete [29, 40].

Using DVFS technique, Huang et al. [35] proposed an Enhanced Energy-aware Scheduling (EES) heuristic to the problem of reducing energy consumption of parallel applications in heterogeneous distributed computing systems (data centers) by considering communication delays and respecting a deadline D on the finish execution time of the applications. The execution times of the tasks depend on the frequency of each processor. The method is based on two phases: EES algorithm firstly employs the Heterogeneous Earliest Finish Time (HEFT) algorithm [26] to find an initial task schedule without respecting the deadline D and energy consumption. This phase determines the time slots for the task execution respecting the communication delays between processors. The second phases focuses on the study of slack room for the non-critical tasks and try to schedule the tasks nearby running on a uniform frequency for global optimality.

By introducing a deadline on the makespan and a constraint on the reliability, Pallez [41] have used DVFS technique to propose a fully polynomial-time approximation scheme for the problem

of minimizing the energy consumption for the particular case of applications represented by a linear chain of tasks without communication delays between successive tasks. The execution times of the tasks depend on the frequency of each processor. The authors showed that there exists no constant factor approximation algorithm for independent tasks using DVFS, unless P=NP.

Based also on DVFS technique, Xie et al. [42] have treated the problem of minimizing the schedule length of a DAG-based parallel application with energy consumption constraint on heterogeneous distributed systems considering communication delays. They decompose the problem in two sub-problems starting by handling the problem of the energy constraint. At each task assignment phase, the energy consumption constraint of the application can always be satisfied by supposing that the unassigned tasks are assigned to the processor with the minimum energy consumption. Then, they proceed to the minimization of makespan, assigning each task to the processor that provides its earliest finish time.

By introducing a novel objective function, which takes into account makespan and energy consumption, Lee and Zomaya [43] proposed the ECS heuristic (Energy Concious Heuristic) taking into account communication delays and using DVFS technique. For a ready task, the value of the objective function is calculated by varying some parameters linked to the frequencies of the processors. The best combination which minimizes the objective function is then selected.

Dynamic power dissipation (processing elements on activity) represents the most expensive part of the power consumption [44, 45]. Most of the work has used the DVFS technique, which is effective for dynamic energy reduction. However, with a growing number of transistors on computing platform and multicore technology, static power is dissipating exponentially. Thus, DPM technique could be an interesting solution to manage the static energy consumption. For this purpose, Ma et al. [38] have proposed an energy-efficient scheduling algorithm on homogeneous DVS-Unable cluster systems taking into account communication delays. The proposed algorithm aims at the energy efficiency of data-intensive applications while maintaining the application performance by including task clustering and task duplication techniques. Kaur et al. [46] have proposed a static energy-efficient scheduling on heterogeneous computing system for DAG-based parallel application considering communication delays using DPM and duplication tasks techniques. They take into account the total energy consumption comprising processor energy (dynamic and static) and network energy (communication energy). To manage the trade-off between makespan and energy consumption caused by duplicating tasks, an adaptive threshold is selected thereby making the system flexible to yield energy-efficient schedule.

In general, Le Sueur and Heiser [47] have shown that on the most recent platform, the effectiveness of DVFS is markedly reduced. Indeed, most systems still consume considerable power when they are idle, and actual savings are only observed when the energy consumption of processors at higher frequencies are padded with the energy consumed when they are idle. In other words, to maintain good performance, it is better to use a high frequency if the idle processor consumes almost the same amount of energy. Furthermore, many HPC systems do not support DVFS technique [38] and may cause transient failures which have an impact of the failure rate of processors and need a dynamic reliability management [48, 49]. While it only works when the idle time is long enough, DPM technique generates a transition delay and an energy penalty for each processor transition (active and low-power modes).

To avoid reliability problems, we assume in this thesis that the processor frequencies are fixed and the energy consumption of each task on each processor is also fixed and does not depend on processors transitions. The main objective of this thesis, is to minimize the completion time of an application while managing the dynamic energy consumption, i.e., when the processing elements are active. For this, we define a bound on the total energy consumption. The static energy consumption is related to the duration of use of the platform, therefore, to the end time of execution of the applications. Thus, by minimizing the completion time, we minimize the static energy consumption. In what follows, we describe the two models used in this thesis: the

consistent and inconsistent models.

1.5.2 Consistent model

In this model, each processing element $p_k \in M$ is characterized by its execution frequency $f_k \geqslant 1$, $k = \overline{1..m}$. The processing elements are sorted by increasing order of their frequencies ($f_1 \leqslant f_2 \leqslant ... \leqslant f_m$). Each task t_i is characterized by its weight w_i, $i = \overline{1..n}$. We denote by W the total weight of the tasks, $W = \sum_{i=1}^{n} w_i$.

The execution of a task t_i on a processing element p_k generates an execution time equal to $w_{i,k} = \frac{w_i}{f_k}$. The execution of a task t_i on a processing element p_k generates an energy consumption equal to $e_{i,k} = w_i * f_k^2$.

The aim is to find a schedule on m uniform machines that minimizes the completion time by respecting precedence constraints with communication costs and respecting an energy constraint. Taking into account the classification of scheduling problems previously defined, this problem can be represented as $Qm|prec, \mathbb{E}, com|C_{max}$ for this model.

This model is motivated by the fact that computation platforms can contain processors of the same architecture, but with different versions (as an example, Intel Core^{TM} i3-4150 Processor and Intel Core^{TM} i7-6700 Processor). The execution time of the different tasks of the same application (same code with the same programming language, same compiler, identical configuration and so on) on different processors may be relative. A single test code can be used to measure their relative speed and the relative speed will be the same for any application. This approach may also work if the processors used in computations have very similar architectural characteristics [50]. Furthermore, the problem of scheduling on uniform parallel machines represents an interesting theoretical problem, and has been widely studied in literature with different formulations and objectives [51, 52].

1.5.3 Inconsistent model

In this model, we consider the more general case where the relation between processing elements can differ for different tasks. Thus we have to take into account that the execution time for any task of the application depends on the processor used to execute it.

The execution of a task t_i on a processing element p_k generates execution time equal to $w_{i,k}$. The execution of a task t_i on a processing element p_k generates an energy consumption equal to $e_{i,k}$.

The aim is to find a schedule on m unrelated machines that minimizes the completion time by respecting precedence constraints with communication costs and respecting an energy constraint. Taking into account the classification of scheduling problems previously defined, this problem can be represented as $Rm|prec, \mathbb{E}, com|C_{max}$ for this model.

The consistent model represents a particular case of the inconsistent model which considered as a more generic model. In addition to the theoretical interest of scheduling on unrelated parallel machines [26], this model is motivated by the fact that computation platforms can contain processors of different architectures (the number of registers, the structure of memory hierarchy, the size of each memory level and so on). Even different applications of the same narrow class may be executed by two different processors at significantly different relative speeds. Moreover, not all processing elements can be programmed in the same language. Several programming models were developed for general computing on graphical processing units (GPUs) like CUDA [53] (Compute Unified Device Architecture) and OpenCL [54] (Open Computing Language).

1.6 Conclusion

This first chapter presents an overview of the fundamental concepts related to this thesis. We briefly presented some basic concepts related to the field of parallel computing.

We first gave a brief overview of heterogeneous platforms, in particular fully heterogeneous and hybrid platforms, and the challenges associated with their use. We have shown that the fully heterogeneous platforms represent an interesting solution to execute complex applications, enclosing great computation capability. In particular, the hybrid platforms, which are increasingly used in the field of HPC, combining multi-core processors and hardware accelerators such as GPUs.

We then discussed the two main scheduling strategies in the literature, namely dynamic and static scheduling. The aim of this thesis is to propose efficient static methods, so that their association with dynamics methods will give the most efficient solution as possible.

Finally, we detailed the execution model of the applications and the energy models that will be used in the rest of this thesis. We have discussed the limits of DVFS and DPM techniques, and the reliability problems related to their use. To avoid reliability problems, we assume in this thesis that the processor frequencies are fixed and the energy consumption of each task on each processor is also fixed and does not depend on processors transitions. The main objective of this thesis, is to minimize the completion time of an application while respecting a bound on the total energy consumption.

Chapter

2 Scheduling sequential applications (chain of tasks) on heterogeneous platforms

Chapter content

2.1 Introduction

In this chapter, we focus on scheduling sequential applications represented by a chain of tasks on heterogeneous resources system. An application is composed of n successive tasks, sorted from t_1 to t_n, i.e., $T = \{t_1, t_2, \cdots, t_n\}$. Each task t_i is characterized by its weight w_i, $i = \overline{1..n}$. We denote by W the total weight of all the application tasks, $W = \sum_{i=1}^{n} w_i$. The execution platform is composed of m heterogeneous processing elements, $M = \{p_1, p_2, \cdots, p_m\}$. Each processing element $p_k \in M$ is characterized by its execution frequency $f_k \geqslant 1$, $k = \overline{1..m}$. The processing elements are sorted by increasing order of their frequencies ($f_1 \leqslant f_2 \leqslant \cdots \leqslant f_m$). The execution of a task t_i on a processing element p_k generates an execution time equal to $w_{i,k} = \frac{w_i}{f_k}$ and an energy consumption equal to $e_{i,k} = w_i * f_k^2$. Furthermore, we consider here a communication delay $cm_{i,l,i+1,k}$ between each two successive tasks t_i and t_{i+1} if t_i is executed on the processing element p_l and t_{i+1} on p_k and $l \neq k$. The aim is to find a schedule on m machines that minimizes the completion time (makespan) by respecting precedence constraints with communication costs. We also have to respect an energy constraint. Taking into account the classification of scheduling problems previously defined in section 1.5.2, this problem can be represented by $Qm|chain, \mathbb{E}, com|C_{max}$.

This model is motivated by the fact that computation platforms can contain processors of the same architecture, but with different versions (as an example, Intel Core^{TM} i3-4150 Processor and Intel Core^{TM} i7-6700 Processor). The execution time of the different tasks of the same application (same code with the same programming language, same compiler, identical configuration and so on) on different processors may be relative. Thus, for two tasks t_1 and t_2 and two processing

elements p_1 and p_2, if $w_{1,1} = \alpha w_{1,2}$, then $w_{2,1} = \alpha w_{2,2}$, with $\alpha > 0$. A single test code can be used to measure their relative speed which will be the same for any application.

Two algorithms have been proposed. The first provides the optimal solution for preemptive scheduling. This solution is then used in the second algorithm to provide an approximate solution for the non-preemptive scheduling.

The rest of this chapter is organized as follows. The next section discusses works related to the problem we address. Section 2.3 presents the mathematical model used to obtain the optimal solution for small instances. Then, we provide in Section 2.4 a polynomial-time algorithm for the preemptive scheduling. Its solution is used to provide a new algorithm for the non-preemptive scheduling in Section 2.5. Finally, after testing the proposed algorithm on several instances in Section 2.6, we provide concluding remarks and future directions in Section 2.7

2.2 Related work

The problem of scheduling application tasks on uniform parallel machines has been widely studied in literature with different formulations and resolution methods [55, 56]. The problem of obtaining the optimal schedules for a uniform processor system with $m \geqslant 2$ processors is known to be NP-complete even for independent tasks and without resource consumption [57].

For independent tasks, Ji et al. [58] treated the problem of finding a schedule that minimizes the total resource consumption (carbon emission, electricity usage) while the makespan was limited by an upper bound. They showed that the problem is NP-complete, derived a tight lower bound, and developed a heuristic algorithm for the proposed problem. For an application composed of independent Unit Execution Time (UET) tasks, Lawler [59] proved that the scheduling problem $Qm|w_i = 1|Cmax$ can be solved in $\mathcal{O}(n \log m)$ time. By allowing tasks preemption, Gonzalez and Sahni [60] proposed an $\mathcal{O}(n)$ algorithm for the problem $Qm|pmtn|Cmax$ with n independent tasks on m uniform processors.

To the best of our knowledge, Yeh et al. [52] are the first to treat the parallel machine makespan problem with a constraint B on the resource consumption (ex: energy). They proposed three heuristics to solve the problem of scheduling independent tasks on m uniform processing elements ($Qm|B|C_{max}$) where some tasks might be processed only on certain machines.

By considering precedence constraints without communication delays, Brucker et al. [61] demonstrated that the problem is solvable in polynomial time for graphs of linear chains of tasks (the precedence constraints are chain-type where every task has at most one direct predecessor and at most one direct successor). They studied the problem of scheduling identical execution time tasks on two uniform processing elements ($Q2|chains, w_i = w|C_{max}$). Kubiak [51] demonstrated that the problem is also solvable in polynomial time for tree-graph ($Q2|tree, w_i = w|C_{max}$). Using a platform composed of m processors with K different speeds, a $2(K + 1)$-approximation algorithm has been developed by Chudak and Shmoys [62] for DAG scheduling problem without communication delays $Qm|prec|C_{max}$.

Kubiak et al. [63] considered a set of k independent chains Ch_1, Ch_2, $\cdots$, Ch_k, $k \geqslant 1$, made up by UET tasks. They take into consideration the precedence constraints, with a unitary communication delay, such that the communication time between each two successive tasks is equal to 1 if they are executed on two different processing elements. The problem is represented as $Qm|w_i = 1, chains, com = 1|Cmax$. They have shown that the problem of scheduling chains of UET tasks on uniform processors with communication delays to minimize makespan is NP-hard in the strong sense. Finally, they presented a heuristic for $Qm|w_i = 1, chains, com = 1|Cmax$ and proved that it generates solutions within $2m - 1$ units from optimum.

We consider here a particular case of applications composed of a set of dependent tasks and represented by a linear chain of tasks (see figure 2.1). The particularity of the structure of these applications gives us the possibility to find a performance guarantee algorithm using the preemptive scheduling.

Hu [64] and Timkovsky [65] studied the preemptive scheduling problem without a resource constraint and without communication delays, which is polynomially solvable for the particular case of tree-graph scheduling on parallel identical processing elements ($Pm|pmtn, tree|C_{max}$). In this chapter, we propose a polynomial-time Preemptive Scheduling algorithm (PS) for the problem of scheduling on m uniform machines for graphs represented by a linear chain of tasks respecting an energy constraint $\mathbb{E}$. This problem can be represented by $Qm|pmtn, chain, \mathbb{E}|C_{max}$.

Chekuriy and Benderz [66] gave a polynomial-time 6-approximation algorithm for a graph of chains $Qm|chains|C_{max}$ without communication delays. A polynomial-time 2-approximation algorithm was then proposed for the same problem by Woeginger [67]. Jansen and Solis-Oba [68] have presented a polynomial time approximation scheme for $Qm|chain|C_{max}$ problem with a performance guarantee ratio related to the number of processing elements and the ratio between the highest and lowest processor frequency. By considering communication delays, we use the solution of the preemptive scheduling (PS) to provide a new algorithm for the non-preemptive scheduling $Qm|chain, com, \mathbb{E}|C_{max}$ by respecting an energy constraint.

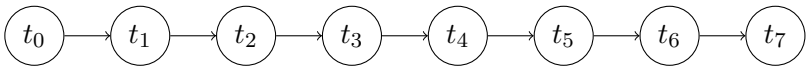


Figure 2.1: *Linear chain graph.*

Remark 1. We consider in this chapter sequential applications represented by a linear chain of tasks. The applications are composed of n successive tasks, numbered from 1 to n. The tasks are sequential, we switch from one processor to another only to increase or decrease the frequency of execution. It is therefore unnecessary to use two processing elements with the same frequency. Thus, we assume that all processors have different frequencies, *i.e.*, $f_1 < f_2 < \cdots < f_m$.

2.3 Mathematical model

The problem of scheduling a linear chain of tasks on uniform parallel machines respecting an energy constraint ($Qm|chain, \mathbb{E}, com|C_{max}$) can be modelled by a mixed integer constrained program (P_{ch}) as follows:

Data:

The data used in this model are:

- $w_{i,k}$ represents the execution time of t_i on the processing element p_k, with $w_{i,k} = \frac{w_i}{f_k}$. w_i is the weight of the task t_i and f_k represents the frequency of the processing element p_k.
- $cm_{i,l,i+1,k}$ represents the communication time between t_i and t_{i+1} if t_i is executed on the processing element p_l and t_{i+1} on p_k and $l \neq k$.
- $e_{i,k}$ is the energy consumption of the task t_i on the processing element p_k, with $e_{i,k} = w_i * f_k^2$.
- $\mathbb{E}$ represents energy bound on the total energy consumption of the application.

Variables:

We consider the following decision variables:

- $x_{i,k}$ equal to 1 if the task t_i is executed on the processing element p_k, 0 otherwise, $i = \overline{1..n}$ and $k = \overline{1..m}$.
- S_i is the starting execution time of the task t_i, $i = \overline{1..n}$.

Model:

The following model (P_{ch}) provides the optimal solution for $Qm|chain, \mathbb{E}, com|C_{max}$ problem.

$$(P_{ch}) \begin{cases} \sum_{k=1}^{m} x_{i,k} = 1, \forall i = \overline{1..n} & (1) \\ \sum_{i=1}^{n} \sum_{k=1}^{m} x_{i,k} * e_{i,k} \leq \mathbb{E} & (2) \\ S_i + x_{i,k_1} * w_{i,k_1} + (x_{i,k_1} + x_{i+1,k_2} - 1) cm_{i,k_1,i+1,k_2} \leqslant S_{i+1} & (3) \\ \qquad \forall k_1 = \overline{1..m}, \quad \forall k_2 = \overline{1..m} \qquad \forall i = \overline{1..n-1} \quad k_1 \neq k_2 & \\ Z(min) = S_n + \sum_{k=1}^{m} x_{n,k} * w_{n,k} & \end{cases}$$

Constraints:

- The first constraint simply expresses that each task must be executed only once and on a single processing element.
- Constraint (2) keeps the energy consumption during execution less than $\mathbb{E}$.
- The third constraint describes that the task t_{i+1} must be executed after the completion time of the task t_i ($i = \overline{1..n-1}$). The communication cost $cm_{i,k_1,i+1,k_2}$ is added if both tasks are executed on two different processing elements (p_{k1} and p_{k2}). If $x_{i,k1} = 1$ and $x_{i+1,k2} = 1$, then $x_{i,k_1} + x_{i+1,k_2} - 1 = 1$. Otherwise, $x_{i,k_1} + x_{i+1,k_2} - 1 \leqslant 0$.

The model (P_{ch}) can be solved by a solver like *CPLEX* but only for small instances. In the following, a polynomial method is proposed to solve the problem for large instances. First, we propose a polynomial-time Preemptive Scheduling algorithm (PS) for the problem $Qm|pmtn, chain, \mathbb{E}|C_{max}$ of scheduling a linear chain of preemptive tasks on m uniform machines respecting an energy constraint $\mathbb{E}$. Then, by considering communication delays, we use the solution of the preemptive scheduling (PS) to provide a new algorithm for the non-preemptive scheduling $Qm|chain, com, \mathbb{E}|C_{max}$. Finally, its performance will be compared to the optimal solution obtained by the model (P_{ch}) for small instances.

2.4 Optimal scheduling algorithm for a chain of preemptive tasks

In this section, we propose an algorithm to find the optimal solution of the preemptive scheduling without communication cost for a chain of n preemptive tasks on a set of m processing elements. The objective is to minimize the makespan while respecting an energy constraint. The problem is represented by $Qm|pmtn, chain, \mathbb{E}|C_{max}$.

Lemma 1. *A feasible solution for the problem $Qm|pmtn, chain, \mathbb{E}|C_{max}$ can be defined by the vector $Pw = \{P_1, P_2, \cdots, P_m\}$, where $P_k \geqslant 0$ is the workload putted on the processing element p_k, $k = \overline{1..m}$. Thus, $\sum_{k=1}^{m} P_k = \sum_{i=1}^{n} w_i = W$, i.e., the processing time of the workload P_k putted the processor p_k is given by $\frac{P_k}{f_k}$.*

Proof. The problem addressed here is the optimization of the preemptive scheduling of an application without considering communication delays, the objective is to minimize the end of its execution time by respecting an energy constraint. Thus, $P_k \geqslant 0$ can be defined as the sum of the execution times (or a fraction) of the tasks assigned to the processing element p_k.

Finally, a feasible solution for the problem $Qm|pmtn, chain, \mathbb{E}|C_{max}$ can be defined by the vector $Pw = \{P_1, P_2, \cdots, P_m\}$, where $P_k \geqslant 0$ is the workload putted on the processing element p_k, $k = \overline{1..m}$. $\sum_{k=1}^{m} P_k = \sum_{i=1}^{n} w_i = W$. The tasks are assigned to the vector Pw in ascending index order, i.e., $P_1 = \sum_{i=1}^{r_1} w_i + w'_{r_1+1}$, where w'_{r_1+1} represents a portion of the weight of the task t_{r_1+1}, $P_2 = (w_{r_1+1} - w'_{r_1+1}) + \sum_{i=r_1+2}^{r_2} w_i + w'_{r_2+1}, \cdots$, $P_m = (w_{r_{m-1}+1} - w'_{r_{m-1}+1}) + \sum_{i=r_{m-1}+2}^{n} w_i$. An example of a preemptive scheduling solution representation is presented below in Example 1. □

Example 1. Figure 2.2 presents an example of scheduling a chain of 5 tasks on 4 processing elements. The vector Pw is given by $Pw = \{P_1, P_2, P_3, P_4\}$ where:

$$P_1 = w_1 + \frac{w_2}{2}$$
$$P_2 = \frac{w_2}{2} + w_3$$
$$P_3 = \frac{w_4}{2}$$
$$P_4 = \frac{w_4}{2} + w_5$$

The finish execution time (makespan) of this instance is then given by $\widehat{C}_{max} = \frac{P_1}{f_1} + \frac{P_2}{f_2} + \frac{P_3}{f_3} + \frac{P_4}{f_4}$.

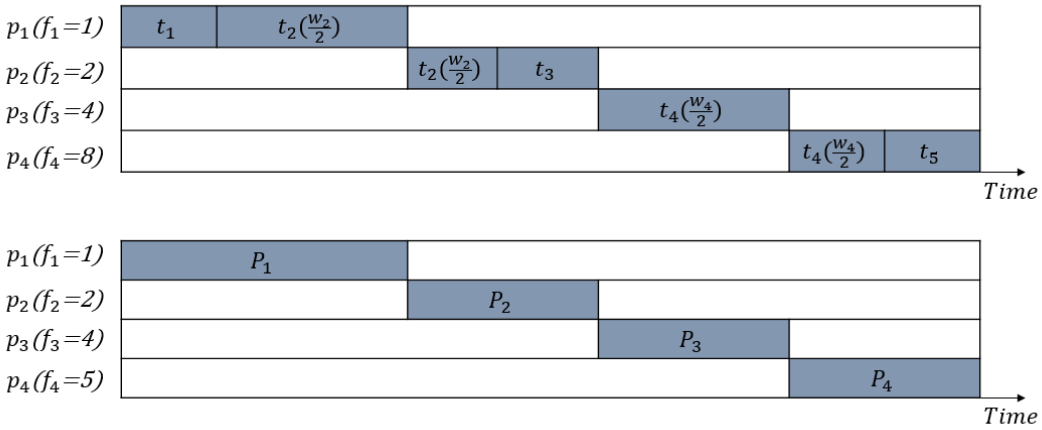


Figure 2.2: *Scheduling a chain of* 5 *tasks on* 4 *processing elements.*

In what follows, the notion of workload is used to define a feasible solution. The next lemma shows that the optimal solution is obtained by saturating the energy constraint $\mathbb{E}$.

Lemma 2. *The set of schedules that saturate energy constraint is dominant.*

Proof. Let $Pw = \{P_1, P_2, \cdots, P_m\}$ be the vector of a solution obtained for an instance I. Let $\widehat{C}_{max}$ be its makespan. Then, $\widehat{C}_{max} = \frac{P_1}{f_1} + \frac{P_2}{f_2} + \cdots + \frac{P_m}{f_m}$.

We assume that this solution does not saturate the energy constraint, i.e., $\sum_{k=1}^{m} P_k * f_k^2 < \mathbb{E}$. We construct another solution $Pw' = \{P'_1, P'_2, \cdots, P'_m\}$, $P'_k \geqslant 0$, such that $\sum_{k=1}^{m} P'_k * f_k^2 = \mathbb{E}$ as follows:

1. First, we look, in ascending order, for the set of tasks that can be moved from the processing elements $\{p_1, p_2, \cdots, p_l\}$, with $l \leqslant m$, to the fastest processing element p_m, while respecting the energy constraint $\mathbb{E}$, i.e., $l = max\{r \in \{1, \cdots, m\}, (\sum_{k=1}^{r} P_k) f_m^2 + \sum_{k=r+1}^{m} P_k f_k^2 < \mathbb{E}\}$. If no tasks can be moved, then $l = 0$.

2. We move all what is executed on $\{p_1, p_2, \cdots, p_l\}$ to p_m if $l > 0$, i.e., $P_1^{'} = 0$, $P_2^{'} = 0$, $\cdots$, $P_l^{'} = 0$, and $P_m^{'} = P_m + \sum_{k=1}^{l} P_k$.
Furthermore, to saturate the energy constraint, we move a part of P_{l+1} to P_m if $\sum_{k=1}^{l} P_k f_m^2 + \sum_{k=l+1}^{m} P_k f_k^2 < \mathbb{E}$. Finally, we set $P_{l+2}^{'} = P_{l+2}$, $P_{l+3}^{'} = P_{l+3}$, $\cdots$, $P_{m-1}^{'} = P_{m-1}$.

   Thus, the energy consumption of the new solution is given by $\mathbb{E} = P_{l+1}^{'} f_{l+1}^2 + (P_{l+1} - P_{l+1}^{'}) f_m^2 + \sum_{k=1}^{l} P_k f_m^2 + \sum_{k=l+2}^{m} P_k f_k^2$. Then, the value of $P_{l+1}^{'}$ can be calculated by $P_{l+1}^{'} = \frac{\mathbb{E} - \sum_{k=1}^{l+1} P_k f_m^2 - \sum_{k=l+2}^{m} P_k f_k^2}{f_{l+1}^2 - f_m^2}$, and then $P_m^{'} = P_m + \sum_{k=1}^{l} P_k + (P_{l+1} - P_{l+1}^{'})$. We resume this solution as follows:

$$
\begin{aligned}
& l = max\{r \in \{1, \cdots, m\},\ \sum_{k=1}^{r} P_k f_m^2 + \sum_{k=r+1}^{m} P_k f_k^2 < \mathbb{E}\} \\
& P_1^{'} = 0 \\
& P_2^{'} = 0 \\
& . \\
& . \\
& . \\
& P_l^{'} = 0 \\
& P_{l+1}^{'} = \frac{\mathbb{E} - \sum_{k=1}^{l+1} P_k f_m^2 - \sum_{k=l+2}^{m} P_k f_k^2}{f_{l+1}^2 - f_m^2} \\
& P_{l+2}^{'} = P_{l+2} \\
& P_{l+3}^{'} = P_{l+3} \\
& . \\
& . \\
& . \\
& P_m^{'} = P_m + \sum_{k=1}^{l} P_k + (P_{l+1} - P_{l+1}^{'})
\end{aligned}
$$

The makespan of the new solution is given by:

$$\widehat{C}_{max}^{'} = \sum_{k=1}^{m} \frac{P_k^{'}}{f_k} = \frac{P_{l+1}^{'}}{f_{l+1}} + \sum_{k=l+2}^{m} \frac{P_k}{f_k} + \frac{\sum_{k=1}^{l} P_k + (P_{l+1} - P_{l+1}^{'})}{f_m}$$

- This solution covers all tasks:

$$
\begin{aligned}
\sum_{k=1}^{m} P_k^{'} &= P_{l+1}^{'} + \sum_{k=1}^{l} P_k + \sum_{k=l+2}^{m} P_k + (P_{l+1} - P_{l+1}^{'}) \\
&= \sum_{k=1}^{l} P_k + \sum_{k=l+2}^{m} P_k + P_{l+1} = \sum_{k=1}^{m} P_k = W
\end{aligned}
$$

- This solution saturates the energy constraint:

$$\sum_{k=1}^{m} P_k^{'} f_k^2 = \sum_{k=1}^{l} P_k f_m^2 + \sum_{k=l+2}^{m} P_k f_k^2 + (P_{l+1} - P_{l+1}^{'}) f_m^2 + P_{l+1}^{'} f_{l+1}^2$$
$$= \sum_{k=1}^{l} P_k f_m^2 + \sum_{k=l+2}^{m} P_k f_k^2 + P_{l+1} f_m^2 + P_{l+1}^{'}(f_{l+1}^2 - f_m^2)$$

Since $P_{l+1}^{'} = \frac{\mathbb{E} - \sum_{k=1}^{l+1} P_k f_m^2 - \sum_{k=l+2}^{m} P_k f_k^2}{f_{l+1}^2 - f_m^2}$, we obtain:

$$P_{l+1}^{'}(f_{l+1}^2 - f_m^2) = \mathbb{E} - \sum_{k=1}^{l+1} P_k f_m^2 - \sum_{k=l+2}^{m} P_k f_k^2$$

Thus, we obtain $\sum_{k=1}^{m} P_k^{'} f_k^2 = \sum_{k=1}^{l} P_k f_m^2 + \sum_{k=l+2}^{m} P_k f_k^2 + P_{l+1} f_m^2 + \mathbb{E} - \sum_{k=1}^{l+1} P_k f_m^2 - \sum_{k=l+2}^{m} P_k f_k^2$

$$= \sum_{k=l+2}^{m} P_k f_k^2 - \sum_{k=l+2}^{m} P_k f_k^2 + P_{l+1} f_m^2 - P_{l+1} f_m^2 + \mathbb{E} = \mathbb{E}$$

- This solution provides better makespan, i.e., $\widehat{C}_{max}^{'} \leqslant \widehat{C}_{max}$. Indeed, since $f_m > f_k$ for $k = \overline{1..l+1}$, we obtain $\frac{\sum_{k=1}^{l} P_k^{'}}{f_m} = \frac{\sum_{k=1}^{l} P_k}{f_m} \leqslant \sum_{k=1}^{l} \frac{P_k}{f_k}$, $\frac{P_{l+1}^{'}}{f_{l+1}} + \frac{(P_{l+1} - P_{l+1}^{'})}{f_m} \leqslant \frac{P_{l+1}}{f_{l+1}}$ and $\sum_{k=l+2}^{m} \frac{P_k^{'}}{f_k} = \sum_{k=l+2}^{m} \frac{P_k}{f_k}$. Finally, $\widehat{C}_{max}^{'} = \sum_{k=1}^{m} \frac{P_k^{'}}{f_k} \leqslant \sum_{k=1}^{m} \frac{P_k}{f_k} = \widehat{C}_{max}$.

□

In the following, we will use this result to propose a polynomial algorithm for the scheduling problem $Qm|pmtn, chain, \mathbb{E}|C_{max}$, and we prove that it provides the optimal solution.

Theorem 1. *The following Algorithm 1 provides the optimal solution for preemptive scheduling without communication cost.*

Algorithm 1: Preemptive scheduling (PS).

Data: Set of processing elements $M = \{p_1, p_2, \cdots, p_m\}$ with $f_1 < f_2 < \cdots < f_m$; weights of the tasks $w_1, w_2, \cdots, w_n$; energy bound $\mathbb{E}$.
Result: Optimal preemptive scheduling.
begin
  $W = \sum_{i=1}^{n} w_i$; $k = max\{r \in \{1, \cdots, m\}, W * f_r^2 \leqslant \mathbb{E}\}$
  if $W * f_k^2 < \mathbb{E}$ **then**
    $W_k = \frac{\mathbb{E} - W * f_{k+1}^2}{f_k^2 - f_{k+1}^2}$
    $W_{k+1} = W - W_k$
  else
    $W_k = W$, $W_{k+1} = 0$
  $s = max\{u \in \{1, \cdots, n\}, \sum_{i=1}^{u} w_i < W_k\}$; if $s = \emptyset$ then $s = 0$;
  $w_{s+1}^{'} = W_k - \sum_{i=1}^{s} w_i$;
  Put $t_1 \cdots t_s$ and a part $w_{s+1}^{'}$ of t_{s+1} on p_k;
  Put $t_{s+2} \cdots t_n$ and the rest $(w_{s+1} - w_{s+1}^{'})$ of t_{s+1} on p_{k+1}.

We start by finding the fastest processing element p_k, on which we can execute all the tasks respecting the energy constraint. Then we look for the weight of tasks that can be assigned to the next processing element (p_{k+1}) in order to saturate the energy constraint. We denote by W_k the workload assigned to the processing element p_k, W_{k+1} on p_{k+1}.

From Lemma 2, the best solution is obtained when the energy constraint is saturated, i.e., $W_k f_k^2 + W_{k+1} f_{k+1}^2 = \mathbb{E}$, with $W_k + W_{k+1} = W$. The solution of the system of two equations with two unknowns is: $W_k = \frac{\mathbb{E} - W * f_{k+1}^2}{f_k^2 - f_{k+1}^2}$ and $W_{k+1} = W - W_k$. This keeps the feasibility of the solution:

- $\mathbb{E} - W * f_{k+1}^2 < 0$ because $W * f_{k+1}^2 > \mathbb{E}$, and $f_k^2 - f_{k+1}^2 < 0$ because $f_k < f_{k+1}$. Thus, $W_k > 0$.
- Furthermore, $W \geqslant W_k > 0$, induces $W_{k+1} = W - W_k \geqslant 0$.

Complexity:

The index k is calculated with a complexity of $\mathcal{O}(m)$, and the workloads are assigned to the two processors p_k and p_{k+1} with a complexity of $\mathcal{O}(n)$. Thus, for an instance of n tasks and m machines, the complexity of Algorithm 1 is $\mathcal{O}(n+m)$. Example 2 below shows the application of Algorithm 1 on an instance of the problem $Qm|pmtn, chain, \mathbb{E}|C_{max}$.

Example 2. Consider a heterogeneous platform with 3 processing elements. Their frequencies are given in Table 2.1. Consider an application represented by the task graph given by Figure 2.3. It contains ten tasks ($n = 10$) labelled from t_1 to t_{10}. The nodes are labelled with the weight of each task. The maximum energy consumption is $\mathbb{E} = 1350$.

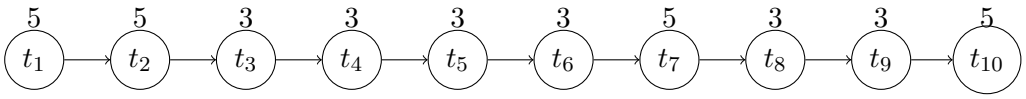


Figure 2.3: *Linear chain graph.*

Table 2.1: *Frequencies of the processing elements.*

p_k	p_1	p_2	p_3
f_k	1	2	6

The progress of algorithm 1 for this instance is given as follows:

1. Find the index k, with $k = max\{r \in \{1, \cdots, m\}, W * f_r^2 \leqslant \mathbb{E}\}$: $W = \sum_{i=1}^{n} w_i = 38$, and $W * f_1^2 = 38 * 1 = 38$, $W * f_2^2 = 38 * 4 = 152$, $W * f_3^2 = 38 * 36 = 1368$. Thus, $k = 2$.
2. Since $W * f_k^2 = W * f_2^2 = 152 < \mathbb{E}$, we obtain: $W_k = W_2 = \frac{\mathbb{E} - W * f_3^2}{f_2^2 - f_3^2} = 0.5625$ and $W_3 = W - W_2 = 37.4375$.
3. $s = max\{u \in \{1, \cdots, n\}, \sum_{i=1}^{u} w_i < W_k\} = max\{u \in \{1, \cdots, n\}, \sum_{i=1}^{u} w_i < 0.5625\} = \emptyset$; and thus, $s = 0$;
4. $w'_{s+1} = w'_1 = W_k - \sum_{i=1}^{s} w_i = W_2 - \sum_{i=1}^{0} w_i = W_2 = 0.5625$;
5. Put w'_1 on p_2;
6. Put $t_2 \cdots t_{10}$ and the rest $(w_1 - w'_1)$ of t_1 on p_3.

Figure 2.4 presents a Gantt illustration of the obtained scheduling.

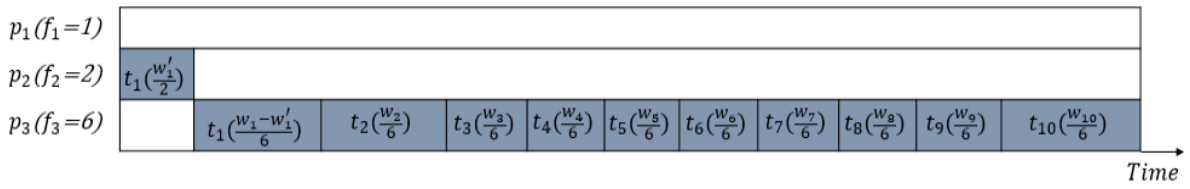


Figure 2.4: *A Gantt illustration of the obtained scheduling.*

We prove in the following that algorithm 1 provides the optimal solution for the scheduling problem $Qm|pmtn, chain, \mathbb{E}|C_{max}$.

Proof of Theorem 1. Let $\widehat{C}_{max}$ be the makespan of the solution obtained by Algorithm 1: $\widehat{C}_{max} = \frac{W_k}{f_k} + \frac{W_{k+1}}{f_{k+1}}$ due to the precedence constraint. Let $\widehat{C}'_{max} = \frac{P_1}{f_1} + \frac{P_2}{f_2} + \cdots + \frac{P_r}{f_r}$ be another solution on a set of $r > 2$ processing elements, $\sum_{l=1}^{r} P_l = W$. We distinguish three possible cases:

1. The first case corresponds to the one where all frequencies are lower than f_k such that $f_1 < f_2 < \cdots < f_r < f_k$.

   Hence, $\frac{1}{f_l} > \frac{1}{f_k}$, induces $\frac{P_l}{f_l} > \frac{P_l}{f_k}$, $\forall\ l = \overline{1..r}$. Follows, $\sum_{l=1}^{r} \frac{P_l}{f_l} > \frac{\sum_{l=1}^{r} P_l}{f_k} = \frac{W}{f_k}$.
   Finally, since $f_k < f_{k+1}$ induces $\sum_{l=1}^{r} \frac{P_l}{f_l} > \frac{W}{f_k} > \frac{W_k}{f_k} + \frac{W_{k+1}}{f_{k+1}}$. Then, $\widehat{C}'_{max} > \widehat{C}_{max}$.

2. The second case corresponds to the one where all frequencies are greater than f_{k+1} such that $f_{k+1} < f_1 < f_2 < \cdots < f_r$. Hence, $\sum_{l=1}^{r} P_l * f_l^2 > \sum_{l=1}^{r} P_l * f_{k+1}^2 = W * f_{k+1}^2 \geqslant \mathbb{E}$. Thus, $\sum_{l=1}^{r} P_l * f_l^2 > \mathbb{E}$, this solution is not feasible.

3. The last case corresponds to the one where $f_1 \leqslant f_k$ and $f_{k+1} \leqslant f_r$, i.e., $f_1 < \cdots < f_k < f_{k+1} < \cdots < f_r$. To study this case, we start with the following technical lemmas 3 and 4.

   Lemma 3. *Let A, B, C be three positive numbers such as $0 < A < B < C$ and W_1,W_2 be two positive numbers such as $W_1 + W_2 = W$.*

   $$\textit{If } W_1 * A^2 + W_2 * C^2 = W * B^2 \textit{ then } \frac{W_1}{A} + \frac{W_2}{C} > \frac{W}{B}$$

   Proof. By replacing W_2 by $(W - W_1)$ in $W_1 * A^2 + W_2 * C^2 = W * B^2$, we obtain $W_1 = W(\frac{C^2 - B^2}{C^2 - A^2})$. Then, by replacing W_1 by $(W - W_2)$, we obtain $W_2 = W(\frac{B^2 - A^2}{C^2 - A^2})$.

   Let $\Delta = \frac{W_1}{A} + \frac{W_2}{C} - \frac{W}{B}$. We prove in the following that $\Delta > 0$, and thus $\frac{W_1}{A} + \frac{W_2}{C} > \frac{W}{B}$.

   $$\Delta = \frac{W(C^2 - B^2)}{A(C^2 - A^2)} + \frac{W(B^2 - A^2)}{C(C^2 - A^2)} - \frac{W}{B} = \frac{W}{C^2 - A^2}\left(\frac{C^2 - B^2}{A} + \frac{B^2 - A^2}{C} - \frac{(C^2 - A^2)}{B}\right)$$

   We set $X = \frac{B}{A}$ and $Y = \frac{C}{A}$. Observe that $X > 1$ because $B > A$, and $Y > X$ because

$C > B$. Then, by setting $B = XA$ and $C = YA$, we obtain:

$$\begin{aligned}
\Delta &= \frac{W}{Y^2A^2 - A^2}\left(\frac{Y^2A^2 - X^2A^2}{A} + \frac{X^2A^2 - A^2}{YA} - \frac{(Y^2A^2 - A^2)}{XA}\right) \\
&= \frac{W}{Y^2A^2 - A^2}\left(Y^2A - X^2A + \frac{X^2A - A}{Y} - \frac{(Y^2A - A)}{X}\right) \\
&= \frac{W}{Y^2A - A}\left(Y^2 - X^2 + \frac{X^2 - 1}{Y} - \frac{(Y^2 - 1)}{X}\right) \\
&= \frac{W}{Y^2A - A}\left(\frac{XY^3 - X^3Y + X^3 - X - Y^3 + Y}{XY}\right) \\
&= \frac{W}{Y^2A - A}\left(\frac{-(X-1)(Y-1)(X-Y)(X+Y+1)}{XY}\right)
\end{aligned}$$

$$\text{Since } Y > X > 1, \text{ we have } \begin{cases} (Y-1) > 0 \\ (X-1) > 0 \\ (X-Y) < 0 \end{cases}$$

Therefore, $\dfrac{-(X-1)(Y-1)(X-Y)(X+Y+1)}{XY} > 0$ and $\dfrac{W}{Y^2A - A} > 0$ because $Y > 1$.

Finally, $\Delta > 0$ induces $\dfrac{W_1}{A} + \dfrac{W_2}{C} > \dfrac{W}{B}$.

□

In the following lemma, we use the lemma 3 to prove that we can construct another solution $\widehat{C}_{max} = \frac{W_k}{f_k} + \frac{W_{k+1}}{f_{k+1}}$ with a better makespan than $\widehat{C}'_{max} = \frac{P_1}{f_1} + \frac{P_2}{f_2} + \cdots + \frac{P_r}{f_r}$ (with $f_1 < \cdots < f_k < f_{k+1} < \cdots < f_r$), using only the processors p_k and p_{k+1}.

Lemma 4. *Let $\widehat{C}_{max}$ be the makespan of the solution obtained by Algorithm 1: $\widehat{C}_{max} = \frac{W_k}{f_k} + \frac{W_{k+1}}{f_{k+1}}$. Let $\widehat{C}'_{max} = \frac{P_1}{f_1} + \frac{P_2}{f_2} + \cdots + \frac{P_r}{f_r}$ be another solution on a set of $r > 2$ processing elements, where $\sum_{l=1}^{r} P_l = W$ and $f_1 < \cdots < f_k < f_{k+1} < \cdots < f_r$. If $\sum_{l=1}^{r} P_l f_l^2 = W_k f_k^2 + W_{k+1} f_{k+1}^2$ and $\sum_{l=1}^{r} P_l = W_k + W_{k+1} = W$, then $\sum_{l=1}^{r} \frac{P_l}{f_l} > \frac{W_k}{f_k} + \frac{W_{k+1}}{f_{k+1}}$, i.e., $\widehat{C}'_{max} \geqslant \widehat{C}_{max}$.*

Proof. The proof of this lemma is divided into three steps as follows:

(a) First, we prove that it exists a frequency $0 < \varphi_k < f_k$, such that $\sum_{l=1}^{k} \frac{P_l}{f_l} > \frac{\sum_{l=1}^{k} P_l}{\varphi_k}$. For this purpose, let φ_l, $l = \overline{1..k}$, be a sequence of real numbers such as:

$$\begin{cases} \varphi_1 = f_1 \\ \varphi_l = \sqrt{\frac{\sum_{\alpha=1}^{l-1} P_\alpha \varphi_{l-1}^2 + P_l f_l^2}{\sum_{\alpha=1}^{l} P_\alpha}}, \quad l = \overline{2..k} \end{cases}$$

This sequence guarantees that $\varphi_{l-1} < \varphi_l < f_l$, $\forall\ l = \overline{2..k}$. Indeed:

$$\text{Since } \varphi_1 = f_1,\ \varphi_2^2 = \frac{P_1\varphi_1^2 + P_2 f_2^2}{P_1 + P_2} = \frac{P_1 f_1^2 + P_2 f_2^2}{P_1 + P_2}.$$

$$\text{Since } f_1 < f_2,\ \varphi_2^2 > \frac{P_1 f_1^2 + P_2 f_1^2}{P_1 + P_2} = f_1^2. \text{ Then, } \varphi_2^2 > f_1^2 = \varphi_1^2, \text{ induces } \varphi_2 > \varphi_1.$$

$$\text{Furthermore, } \varphi_2^2 = \frac{P_1 f_1^2 + P_2 f_2^2}{P_1 + P_2} < \frac{P_1 f_2^2 + P_2 f_2^2}{P_1 + P_2} < f_2^2, \text{ induces } \varphi_1 < \varphi_2 < f_2.$$

We assume that this is true for $l = k-1$, i.e., $\varphi_{k-2} < \varphi_{k-1} < f_{k-1}$, and we prove that the propriety remains true for $l = k$, i.e., $\varphi_{k-1} < \varphi_k < f_k$.

First, $\varphi_k^2 = \frac{\sum_{\alpha=1}^{k-1} P_\alpha \varphi_{k-1}^2 + P_k f_k^2}{\sum_{\alpha=1}^{k} P_\alpha}$. And since $\varphi_{k-1} < f_{k-1}$, $\varphi_k^2 < \frac{\sum_{\alpha=1}^{k-1} P_\alpha f_{k-1}^2 + P_k f_k^2}{\sum_{\alpha=1}^{k} P_\alpha}$. Follows, since $f_{k-1} < f_k$, $\varphi_k^2 < \frac{\sum_{\alpha=1}^{k-1} P_\alpha f_k^2 + P_k f_k^2}{\sum_{\alpha=1}^{k} P_\alpha} = f_k^2$. Furthermore, $\varphi_k^2 = \frac{\sum_{\alpha=1}^{k-1} P_\alpha \varphi_{k-1}^2 + P_k f_k^2}{\sum_{\alpha=1}^{k} P_\alpha}$.

Then, since $\varphi_{k-1} < f_{k-1} < f_k$, $\varphi_k^2 > \frac{\sum_{\alpha=1}^{k-1} P_\alpha \varphi_{k-1}^2 + P_k \varphi_{k-1}^2}{\sum_{\alpha=1}^{k} P_\alpha} = \varphi_{k-1}^2$. Thus, $\varphi_k^2 > \varphi_{k-1}^2$. Finally, $\varphi_{k-1} < \varphi_k < f_k$. Now, from Lemma 3, we have:

Since $f_1 < \varphi_2 < f_2$, $\frac{P_1}{f_1} + \frac{P_2}{f_2} > \frac{P_1+P_2}{\varphi_2}$, and since $\varphi_2 < \varphi_3 < f_3$, $\frac{P_1+P_2}{\varphi_2} + \frac{P_3}{f_3} > \frac{\sum_{l=1}^{3} P_l}{\varphi_3}$. Finally, since $\varphi_{l-1} < \varphi_l < f_l$, $\forall\ l \in \{2, \cdots, k\}$, we obtain $\frac{\sum_{l=1}^{l-1} P_l}{\varphi_{l-1}} + \frac{P_l}{f_l} > \frac{\sum_{l=1}^{l} P_l}{\varphi_l}$. Follows, $\sum_{l=1}^{k} \frac{P_l}{f_l} > \frac{\sum_{l=1}^{k} P_l}{\varphi_k}$.

(b) Second, we prove that it exists a frequency $f_{k+1} < \phi_{k+1}$, such that $\sum_{l=k+1}^{r} \frac{P_l}{f_l} > \frac{\sum_{l=k+1}^{r} P_l}{\phi_{k+1}}$. For this purpose, let ϕ_l, $l = \overline{k+1..r}$, be another sequence of real such as:

$$\begin{cases} \phi_r = f_r \\ \phi_l = \sqrt{\frac{\sum_{\alpha=l+1}^{r} P_\alpha \phi_{l+1}^2 + P_l f_l^2}{\sum_{\alpha=l}^{r} P_\alpha}}, \quad l = \overline{k+1..r-1} \end{cases}$$

Using the same previous analysis, this sequence guarantees that $f_l < \phi_l < \phi_{l+1}$, $\forall\ l = \overline{k+1..r-1}$. Thus, according to Lemma 3, we obtain:

$$\sum_{l=k+1}^{r} \frac{P_l}{f_l} > \frac{\sum_{l=k+1}^{r} P_l}{\phi_{k+1}} \tag{2.13a}$$

$$\text{It results that, } \sum_{l=1}^{r} \frac{P_l}{f_l} > \frac{\sum_{l=1}^{k} P_l}{\varphi_k} + \frac{\sum_{l=k+1}^{r} P_l}{\phi_{k+1}} \tag{2.13b}$$

(c) Finally, using the two results obtained before, we prove that $\sum_{l=1}^{r} \frac{P_l}{f_l} > \frac{W_k}{f_k} + \frac{W_{k+1}}{f_{k+1}}$. In order to apply once again Lemma 3, we decompose $\sum_{l=1}^{k} P_l$ and $\sum_{l=k+1}^{r} P_l$ into 4 values $W_{L1}, W_{L2}, W_{R1}, W_{R2}$ such that:

$$\begin{cases} W_{L1} + W_{L2} = \sum_{l=1}^{k} P_l \\ W_{R1} + W_{R2} = \sum_{l=k+1}^{r} P_l \\ W_{L1} + W_{R1} = W_k \\ W_{L2} + W_{R2} = W_{k+1} \\ W_{L1}\varphi_k^2 + W_{R1}\phi_{k+1}^2 = W_k f_k^2 \\ W_{L2}\varphi_k^2 + W_{R2}\phi_{k+1}^2 = W_{k+1} f_{k+1}^2 \end{cases} \Longrightarrow \begin{cases} W_{L1} = W_k \frac{(\phi_{k+1}^2 - f_k^2)}{(\phi_{k+1}^2 - \varphi_k^2)} \\ W_{L2} = W_{k+1} \frac{(\phi_{k+1}^2 - f_{k+1}^2)}{(\phi_{k+1}^2 - \varphi_k^2)} \\ W_{R1} = W_k \frac{(f_k^2 - \varphi_k^2)}{(\phi_{k+1}^2 - \varphi_k^2)} \\ W_{R2} = W_{k+1} \frac{(f_{k+1}^2 - \varphi_k^2)}{(\phi_{k+1}^2 - \varphi_k^2)} \end{cases}$$

Since $\varphi_{l-1} < \varphi_l < f_l$, $\forall\ l = \overline{2..k}$ and $f_l < \phi_l < \phi_{l+1}$, $\forall\ l = \overline{k+1..r-1}$, we have:

- $\phi_{k+1}^2 > f_{k+1}^2 > f_k^2$, then $\phi_{k+1}^2 - f_k^2 > 0$. $\phi_{k+1}^2 > f_{k+1}^2 > f_k^2 > \varphi_k^2$, then $\phi_{k+1}^2 - \varphi_k^2 > 0$. Thus, $W_{L1} = W_k \frac{(\phi_{k+1}^2 - f_k^2)}{(\phi_{k+1}^2 - \varphi_k^2)} > 0$.
- $\phi_{k+1}^2 > f_{k+1}^2$, then $\phi_{k+1}^2 - f_{k+1}^2 > 0$. $\phi_{k+1}^2 > f_{k+1}^2 > f_k^2 > \varphi_k^2$, then $\phi_{k+1}^2 - \varphi_k^2 > 0$. Thus, $W_{L2} = W_{k+1} \frac{(\phi_{k+1}^2 - f_{k+1}^2)}{(\phi_{k+1}^2 - \varphi_k^2)} > 0$.

- $f_k^2 > \varphi_k^2$, then $f_k^2 - \varphi_k^2 > 0$. $\phi_{k+1}^2 > f_{k+1}^2 > f_k^2 > \varphi_k^2$, then $\phi_{k+1}^2 - \varphi_k^2 > 0$. Thus, $W_{R1} = W_k \frac{(f_k^2 - \varphi_k^2)}{(\phi_{k+1}^2 - \varphi_k^2)} > 0$.
- $f_{k+1}^2 > f_k^2 > \varphi_k^2$, then $f_{k+1}^2 - \varphi_k^2 > 0$. $\phi_{k+1}^2 > f_{k+1}^2 > f_k^2 > \varphi_k^2$, then $\phi_{k+1}^2 - \varphi_k^2 > 0$. Thus, $W_{R2} = W_{k+1} \frac{(f_{k+1}^2 - \varphi_k^2)}{(\phi_{k+1}^2 - \varphi_k^2)} > 0$.

Observe that the result values are all positive. From Lemma 3, we obtain:

$$\frac{W_{L1}}{\varphi_k} + \frac{W_{R1}}{\phi_{k+1}} > \frac{W_k}{f_k} \text{ and } \frac{W_{L2}}{\varphi_k} + \frac{W_{R2}}{\phi_{k+1}} > \frac{W_{k+1}}{f_{k+1}}$$

$$\text{Thus, } \frac{W_L}{\varphi_k} + \frac{W_R}{\phi_{k+1}} = \frac{W_{L1}}{\varphi_k} + \frac{W_{R1}}{\phi_{k+1}} + \frac{W_{L2}}{\varphi_k} + \frac{W_{R2}}{\phi_{k+1}} > \frac{W_k}{f_k} + \frac{W_{k+1}}{f_{k+1}}$$

From (2.13b), we have:

$$\begin{aligned}\sum_{l=1}^{r} \frac{P_l}{f_l} > \frac{\sum_{l=1}^{k} P_l}{\varphi_k} + \frac{\sum_{l=k+1}^{r} P_l}{\phi_{k+1}} &= \frac{W_{L1} + W_{L2}}{\varphi_k} + \frac{W_{R1} + W_{R2}}{\phi_{k+1}} \\ &= \frac{W_{L1}}{\varphi_k} + \frac{W_{R1}}{\phi_{k+1}} + \frac{W_{L2}}{\varphi_k} + \frac{W_{R2}}{\phi_{k+1}} > \frac{W_k}{f_k} + \frac{W_{k+1}}{f_{k+1}}\end{aligned}$$

$$\text{Follows, } \sum_{l=1}^{r} \frac{P_l}{f_l} > \frac{W_k}{f_k} + \frac{W_{k+1}}{f_{k+1}}$$

The three steps of this lemma proof are illustrated by Figure 2.5. First, we show how to introduce φ_l, $l = \overline{1..k}$, and ϕ_l, $l = \overline{k+1..r}$, to prove that $\sum_{l=1}^{k} \frac{P_l}{f_l} > \frac{\sum_{l=1}^{k} P_l}{\varphi_k}$ and $\sum_{l=k+1}^{r} \frac{P_l}{f_l} > \frac{\sum_{l=k+1}^{r} P_l}{\phi_{k+1}}$. Then, we show how we use W_{L1} and W_{R1} to prove that $\frac{W_{L1}}{\varphi_k} + \frac{W_{R1}}{\phi_{k+1}} > \frac{W_k}{f_k}$, then W_{L2} and W_{R2} to prove that $\frac{W_{L2}}{\varphi_k} + \frac{W_{R2}}{\phi_{k+1}} > \frac{W_{k+1}}{f_{k+1}}$. □

Thus, the third case of the proof of theorem 1 is illustrated by the lemma 4, i.e., $\widehat{C}'_{max} = \sum_{l=1}^{r} \frac{P_l}{f_l} > \widehat{C}_{max} = \frac{W_k}{f_k} + \frac{W_{k+1}}{f_{k+1}}$. Finally, Algorithm 1 provides the optimal solution for the preemptive scheduling problem $Qm|pmtn, chain, \mathbb{E}|C_{max}$.

□

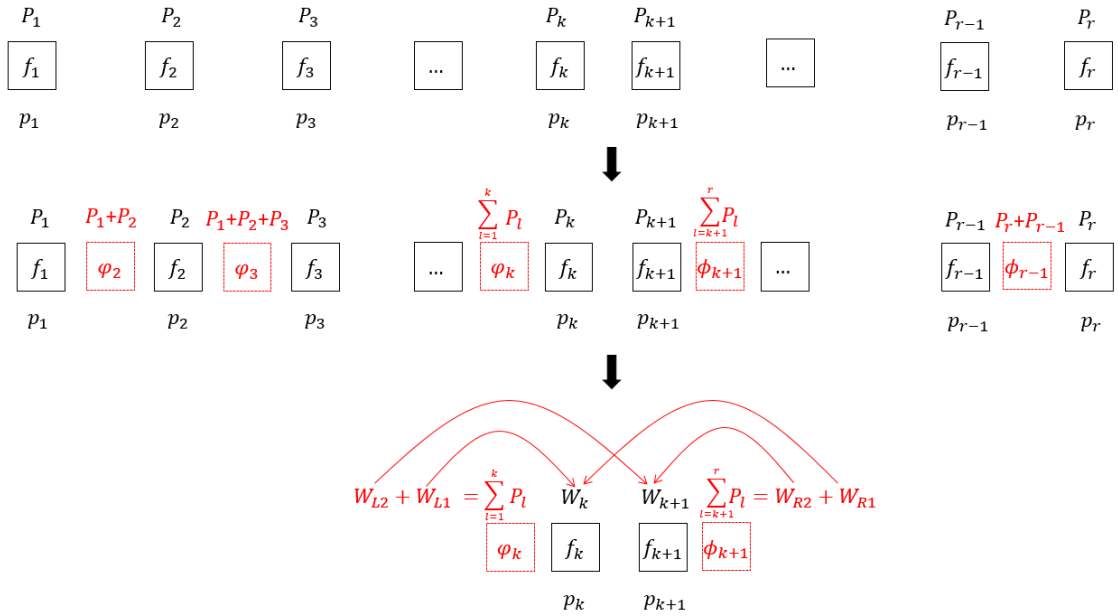


Figure 2.5: *Summary of the first part of the proof.*

Remark 2. The proof remains valid if $\sum_{l=1}^{r} P_l f_l^2 \leqslant W_k f_k^2 + W_{k+1} f_{k+1}^2$. Indeed, from Lemma 2, we can obtain another solution with $P_1', P_2', \cdots, P_r'$, such as: $\sum_{l=1}^{r} P_l' = \sum_{l=1}^{r} P_l$ and $\sum_{l=1}^{r} P_l' f_l^2 = W_k f_k^2 + W_{k+1} f_{k+1}^2$. Thus, we obtain $\frac{W_k}{f_k} + \frac{W_{k+1}}{f_{k+1}} < \sum_{l=1}^{r} \frac{P_l'}{f_l} < \sum_{l=1}^{r} \frac{P_l}{f_l}$.

2.5 An approximation scheduling algorithm for chain of non-preemptive tasks with communication costs

In this section, we solve the problem of scheduling application tasks on m machines that minimizes the completion time (makespan) by respecting precedence constraints with communication costs and respecting an energy constraint. Taking into account the classification of scheduling problems previously defined in section 1.5.2, this problem can be represented by $Qm|chain, \mathbb{E}, com|C_{max}$.

For this purpose, we transform the previous solution obtained for the preemptive scheduling, using only the two processing elements p_k and p_{k+1} used for the preemptive solution. Compared to the preemptive scheduling, in this section we do not allow preemption of tasks and we consider communication delays.

Theorem 2. *The following Algorithm 2 provides a solution for the non-preemptive scheduling starting from the preemptive scheduling solution obtained by algorithm 1.*

The two variables α and β are used to determine the assignment of tasks. In the case $W_{k+1} = 0$, we put all the tasks on p_k. Otherwise, algorithm 2 compares three scheduling policies and takes the best one (which gives the smallest makespan). The three policies are calculated as follows:

1. Starting by executing some tasks on p_k, then the rest on p_{k+1}: let $Cost_1(v)$ be the makespan obtained by executing the first tasks (t_1 to t_v) on p_k with $\sum_{i=1}^{v} w_i \geqslant W_k$, then the rest on p_{k+1}.

$$Cost_1(v) = \{\frac{\sum_{i=1}^{v} w_i}{f_k} + cm_{v,k,v+1,k+1} + \frac{\sum_{i=v+1}^{n} w_i}{f_{k+1}}\}$$

This function depends on the workload of tasks assigned to each processor, and the communication cost between tasks. The algorithm chooses the best index v_1, such that $\sum_{i=1}^{v_1} w_i \geqslant W_k$ and $Cost_1(v_1)$ is minimal.

2. Starting by executing some tasks on p_{k+1}, then the rest on p_k: Let $Cost_2(v)$ be the makespan obtained by executing the first tasks (t_1 to t_v) on p_{k+1}, then the rest on p_k with $\sum_{i=v+1}^{n} w_i \geqslant W_k$.

$$Cost_2(v) = \{\frac{\sum_{i=1}^{v} w_i}{f_{k+1}} + cm_{v,k,v+1,k+1} + \frac{\sum_{i=v+1}^{n} w_i}{f_k}\}$$

This function also depends on the workload of tasks assigned to each processor, and the communication cost between tasks. The algorithm chooses the best index v_2, such that $\sum_{i=v_2+1}^{n} w_i \geqslant W_k$ and $Cost_2(v_2)$ is minimal.

3. Finally, we check if the makespan generated by using both processing elements p_k and p_{k+1} is smaller than the scheduling makespan obtained by executing all tasks on p_k. Thus, if $\frac{\sum_{i=1}^{n} w_i}{f_k} \leqslant Cost_1(v_1)$ and $\frac{\sum_{i=1}^{n} w_i}{f_k} \leqslant Cost_2(v_2)$, then the best makespan is obtained by executing all tasks on the processing element p_k.

Algorithm 2: Non-Preemptive Scheduling (NPS).

Data: Weights of the tasks $w_1, w_2, \cdots, w_n$; communication costs between tasks between p_k and p_{k+1} ($cm_{i,k,i+1,k+1}$), $i = \overline{1..n-1}$; energy bound $\mathbb{E}$.
Result: A feasible solution for the non-preemptive scheduling.
begin
  Find p_k, p_{k+1} and W_k, W_{k+1} using the Preemptive Scheduling (PS) solution obtained by Algorithm 1;
  if $W_k = W$ **then**
    $\beta = n$, $\alpha = 1$
  else
    $Cost_1 = min\{Cost_1(v), v = \overline{1..n-1}, \sum_{i=1}^{v} w_i \geqslant W_k\}$; let v_1 be the best index, i.e., $Cost_1(v_1) = Cost_1$;
    $Cost_2 = min\{Cost_2(v), v = \overline{1..n-1}, \sum_{i=v+1}^{n} w_i \geqslant W_k\}$; let v_2 be the best index, i.e., $Cost_2(v_2) = Cost_2$;
    if $Cost_1 < Cost_2$ **then**
      $Cost = Cost_1$, $\beta = v_1$, $\alpha = 1$
    else
      $Cost = Cost_2$, $\beta = n$, $\alpha = v_2 + 1$
    if $Cost \geqslant \frac{W}{f_k}$ **then**
      $Cost = \frac{W}{f_k}$, $\beta = n$, $\alpha = 1$
  Put tasks between t_α and t_β on the processing element p_k;
  Order the rest on processing element p_{k+1}, $\widehat{C}_{max} = Cost$.

Lemma 5. *The energy consumption of the scheduling generated by NPS Algorithm 2 respects the energy constraint for the three scheduling policies.*

Proof. 1. The makespan of the first scheduling policy is given by $Cost_1(v) = \{\frac{\sum_{i=1}^{v} w_i}{f_k} + cm_{v,k,v+1,k+1} + \frac{\sum_{i=v+1}^{n} w_i}{f_{k+1}}\}$, obtained by executing the first tasks (t_1 to t_v) on p_k with $\sum_{i=1}^{v} w_i \geqslant W_k$, then the rest on p_{k+1}.

Since $\sum_{i=1}^{v_1} w_i \geqslant W_k$, we have $\sum_{i=v_1+1}^{n} w_i = W - \sum_{i=1}^{v_1} w_i \leqslant W - W_k = W_{k+1}$, and thus, $\sum_{i=v_1+1}^{n} w_i \leqslant W_{k+1}$. The energy consumption of this solution is given by $\sum_{i=1}^{v_1} w_i * f_k^2 + \sum_{i=v_1+1}^{n} w_i * f_{k+1}^2 \leqslant W_k * f_k^2 + (\sum_{i=v_1+1}^{n} w_i + (\sum_{i=1}^{v_1} w_i - W_k)) * f_{k+1}^2$ since $\sum_{i=1}^{v_1} w_i \geqslant W_k$ and $f_k^2 < f_{k+1}^2$. Then, $W_k * f_k^2 + (\sum_{i=v_1+1}^{n} w_i + (\sum_{i=1}^{v_1} w_i - W_k)) * f_{k+1}^2 = W_k * f_k^2 + (W - W_k) * f_{k+1}^2 = W_k * f_k^2 + W_{k+1} * f_{k+1}^2 = \mathbb{E}$.

Finally, $\sum_{i=1}^{v_1} w_i * f_k^2 + \sum_{i=v_1+1}^{n} w_i * f_{k+1}^2 \leqslant \mathbb{E}$, thus, the energy constraint is respected.

2. The makespan of the second scheduling policy is given by $Cost_2(v) = \{\frac{\sum_{i=1}^{v} w_i}{f_{k+1}} + cm_{v,k,v+1,k+1} + \frac{\sum_{i=v+1}^{n} w_i}{f_k}\}$, obtained by executing the first tasks (t_1 to t_v) on p_{k+1}, then the rest on p_k with $\sum_{i=v+1}^{n} w_i \geqslant W_k$.

Since $\sum_{i=v_2+1}^{n} w_i \geqslant W_k$, we have $\sum_{i=1}^{v_2} w_i = W - \sum_{i=v_2+1}^{n} w_i \leqslant W - W_k = W_{k+1}$, and thus, $\sum_{i=1}^{v_2} w_i \leqslant W_k$. The energy consumption of this solution is given by $\sum_{i=v_2+1}^{n} w_i * f_k^2 + \sum_{i=1}^{v_2} w_i * f_{k+1}^2 \leqslant W_k * f_k^2 + (\sum_{i=1}^{v_2} w_i + (\sum_{i=v_2+1}^{n} w_i - W_k)) * f_{k+1}^2$ since $\sum_{i=v_2+1}^{n} w_i \geqslant W_k$ and $f_k^2 < f_{k+1}^2$. Then, $W_k * f_k^2 + (\sum_{i=1}^{v_2} w_i + (\sum_{i=v_2+1}^{n} w_i - W_k)) * f_{k+1}^2 = W_k * f_k^2 + (W - W_k) * f_{k+1}^2 = W_k * f_k^2 + W_{k+1} * f_{k+1}^2 = \mathbb{E}$. Finally, $\sum_{i=v_2+1}^{n} w_i * f_k^2 + \sum_{i=1}^{v_2} w_i * f_{k+1}^2 \leqslant \mathbb{E}$, and thus, the energy constraint is also respected.

3. The makespan of the last scheduling policy is obtained by executing all tasks on p_k. The energy constraint is also respected in this case, since $W * f_k^2 \leqslant \mathbb{E}$.

□

Complexity

Algorithm 1 provides the optimal solution for preemptive scheduling with a complexity of $\mathcal{O}(n+m)$, and we look for v_1 and v_2 in Algorithm 2 with a complexity of $\mathcal{O}(n)$. Thus, for an instance of n tasks and m machines, the complexity of Algorithm 2 is $\mathcal{O}(n+m)$.

Example 3 shows the application of Algorithm 2 on an instance of the problem $Qm|chain, \mathbb{E}, com|C_{max}$.

Example 3. We take the same instance used in the example 2 for PS Algorithm 1. We consider a heterogeneous platform with 3 processing elements. Their frequencies are given in Table 2.2. We consider an application represented by the task graph given by Figure 2.6. It contains ten tasks ($n = 10$) labelled from t_1 to t_{10}. The nodes are labelled with the weight of each task. For this example, we suppose that $cm_{i,1,i+1,2} = cm_{i,1,i+1,3} = cm_{i,2,i+1,3}$ for $i = \overline{1..9}$. The edges are labelled with the communication cost between tasks. The maximum energy consumption is $\mathbb{E}$=1350.

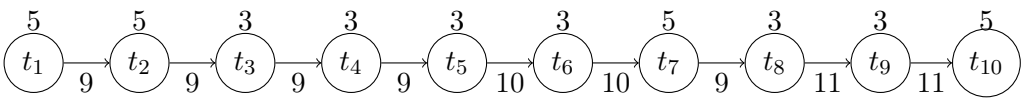


Figure 2.6: *Linear chain graph.*

Table 2.2: *Frequencies of the processing elements.*

p_k	p_1	p_2	p_3
f_k	1	2	6

The progress of Algorithm 2 for this instance is given as follows:

1. The application of preemptive scheduling Algorithm 1 (example 2) gives: $p_k = p_2$ with $W_2 = 0.5625$ and $p_{k+1} = p_3$ with $W_3 = 37.4375$.
2. Since $W_3 > 0$, we obtain: $\begin{cases} Cost_1 = \frac{w_1}{f_2} + \frac{\sum_{i=2}^{10} w_i}{f_3} + cm_{1,2,2,3} = 17 & \text{with } v_1 = 1 \\ Cost_2 = \frac{\sum_{i=1}^{7} w_i}{f_2} + \frac{\sum_{i=8}^{10} w_i}{f_3} + cm_{7,2,8,3} = 19 & \text{with } v_2 = 7 \end{cases}$
3. We check if the makespan generated by using both processing elements p_k and p_{k+1} is smaller than the scheduling makespan obtained by executing all tasks on p_k. Since $Cost_1 < Cost_2$, we obtain $Cost = Cost_1 = 17$ with $\beta = 1$ and $\alpha = 1$. Finally, $\frac{W}{f_2} = \frac{38}{2} = 19 > Cost$. Thus, the best makespan is $Cost_1$ obtained with the first scheduling policy.
4. We put the task t_1 on the processing element p_2 and tasks t_2 to t_{10} on p_3.

We obtain a solution with $\widehat{C}_{max} = Cost = 17$. For this instance, our approach provides an optimal solution. Figure 2.7 presents a Gantt illustration of the obtained scheduling.

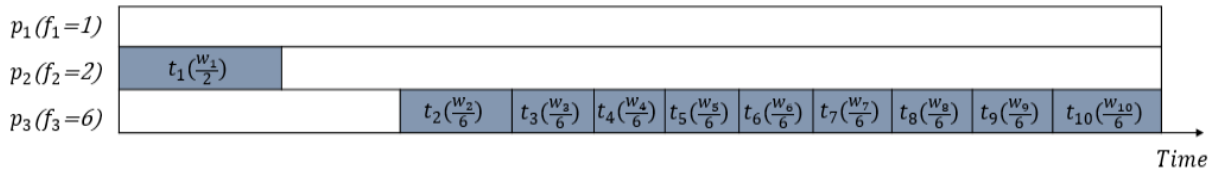


Figure 2.7: *A Gantt illustration of the obtained scheduling.*

Algorithm 2 is a generic algorithm, it does not always guarantee to obtain the optimal solution. To study the theoretical worst-case performance of our method, we prove in what follows that Algorithm 2 solves the scheduling problem $Qm|chain, \mathbb{E}, com|C_{max}$ with a performance guarantee bounded by $\frac{f_{k+1}}{f_k}$ compared to the optimal solution.

Algorithm analysis

Let $C^{\star}_{max}$ be the optimal solution of our problem (best makespan for $Qm|chain, \mathbb{E}, com|C_{max}$) and $\widehat{C}_{max}$ be the solution obtained by NPS Algorithm 2. To show the performance guarantee of NPS Algorithm 2, we compare its solution $\widehat{C}_{max}$ to the optimal solution $C^{\star}_{max}$ in the worst case as follows.

Proposition 1. *The ratio between $\widehat{C}_{max}$ the solution obtained by using NPS Algorithm 2 and the optimal solution $C^{\star}_{max}$ is* $\frac{\widehat{C}_{max}}{C^{\star}_{max}} \leqslant \frac{W}{W_k + \frac{f_k W_{k+1}}{f_{k+1}}}$.

Proof. The optimal solution C'_{max} of the preemptive scheduling is given by $C'_{max} = \frac{W_k}{f_k} + \frac{W_{k+1}}{f_{k+1}}$ due to precedence constraints. In the worst case, NPS Algorithm 2 assigns all tasks to the

processing element p_k, i.e., $\widehat{C}_{max} \leqslant \frac{W}{f_k}$. Follows, $\frac{\widehat{C}_{max}}{C'_{max}} \leqslant \frac{\frac{W}{f_k}}{\frac{W_k}{f_k} + \frac{W_{k+1}}{f_{k+1}}} \leqslant \frac{W}{W_k + \frac{f_k W_{k+1}}{f_{k+1}}}$. Furthermore, the optimal solution of PS Algorithm 1 represents a lower bound for the scheduling problem $Qm|chain, \mathbb{E}, com|C_{max}$, i.e., $C'_{max} \leqslant C^{\star}_{max}$. Finally, $\frac{\widehat{C}_{max}}{C^{\star}_{max}} \leqslant \frac{\widehat{C}_{max}}{C'_{max}} \leqslant \frac{W}{W_k + \frac{f_k W_{k+1}}{f_{k+1}}}$.

□

Proposition 2. *The ratio between $\widehat{C}_{max}$ the solution obtained by using NPS Algorithm 2 and the optimal solution $C^{\star}_{max}$ can be bounded by the ratio between the frequencies of the two processing elements p_k et p_{k+1}, i.e., $\frac{\widehat{C}_{max}}{C^{\star}_{max}} < \frac{f_{k+1}}{f_k}$.*

Proof.

Since, $\frac{f_k}{f_{k+1}} < 1$, $\frac{\widehat{C}_{max}}{C^{\star}_{max}} \leqslant \frac{W}{W_k + \frac{f_k W_{k+1}}{f_{k+1}}} < \frac{W}{\frac{f_k}{f_{k+1}}(W_k + W_{k+1})} = \frac{f_{k+1}}{f_k}$. Finally, $\frac{\widehat{C}_{max}}{C^{\star}_{max}} < \frac{f_{k+1}}{f_k}$.

□

Remark 3. In general, the ratio $\frac{\widehat{C}_{max}}{C^{\star}_{max}}$ can be bounded by the ratio between the frequencies of two successive processing elements. Let Υ be the value given by $\Upsilon = max\{\frac{f_{h+1}}{f_h}, h = \overline{1..m-1}\}$. Then, $\frac{\widehat{C}_{max}}{C^{\star}_{max}} < \frac{f_{k+1}}{f_k} \leqslant \Upsilon$. Finally, $\frac{\widehat{C}_{max}}{C^{\star}_{max}} < \Upsilon$.

We have shown that the ratio between $\widehat{C}_{max}$ the solution obtained using NPS Algorithm 2 and the optimal solution $C^{\star}_{max}$ for the non-preemptive scheduling in the worst case depends on the ratio between the frequencies of two successive processing element, i.e., $\frac{\widehat{C}_{max}}{C^{\star}_{max}} < \frac{f_{k+1}}{f_k}$. In what follows, NPS Algorithm 2 is compared to the optimal solution $C^{\star}_{max}$ and to the lower bound C'_{max} using several instances randomly generated.

2.6 Experimental results

In order to measure the efficiency of our algorithm, we performed several tests on randomly generated instances with different dimensions. For this purpose, we developed a random instance generator using *C++* with several input parameters. General settings are the number of tasks n and the number of processing elements m.

We denote by *test_n_m* the instance defined by n tasks and m processing elements. The weights of the tasks are generated randomly over an interval $[w_{min}, w_{max}]$, with $w_{min} = 4$ and $w_{max} = 100$. The frequencies of the processing elements are also randomly generated over an interval $[f_{min}, f_{max}]$ while ensuring the heterogeneity of the system by generating different values, where $f_{min} = 2$ and $f_{max} = 2 * m$. The communication costs between tasks and between different machines are also generated randomly over an interval $[cm_{min}, cm_{max}]$, with $cm_{min} = 4$ and $cm_{max} = 100$. The bound $\mathbb{E}$ is randomly generated with $W * f_1^2 < \mathbb{E} < W * f_m^2$, which is a sufficient condition to the existence of a feasible solution.

Both Algorithms 1 and 2 were implemented using *C++*. The exact solution is obtained by solving the model (P_{ch}) with *CPLEX* 12.5.0 [69] and the OPL script language. All tests have been performed on a 48-core AMD Opteron 6172 server with 64 GB RAM and Debian GNU/Linux 8.2 operating system.

The following Table 2.3 shows the results of tests on different instance sizes. We have generated 30 instances for the first eight rows (from instance *test_8_3* to *test_100_6*), then ten instances for the other ones (from instance *test_200_9* to *test_10000_11*). We compared them

only to the lower bound (PS solution) due to the large running time (> 60 minutes) needed to obtain the optimal solution using *CPLEX*.

Column ***PS*** (Preemptive Scheduling) presents the average makespan obtained by Algorithm 1. Columns ***CPLEX*** present the average optimal solution (**Sol**) obtained by the resolution of the model (P_{ch}) with *CPLEX* and the average running time required to get it (**Time**). In column **Opt**, "✓" means that the optimal solution is obtained using model (P_{ch}) for all instances, otherwise we put "/". The next columns of the table concern the results of NPS (Non-Preemptive Scheduling) algorithm. Columns **Sol** presents the average makespan obtained by NPS Algorithm and columns **Time** its average running time. Column **Opt** gives the number of optimal solution obtained by NPS algorithm. Column $\boldsymbol{GAP_1}$ (resp. $\boldsymbol{GAP_2}$) presents the average ratio between NPS algorithm and the solution obtained by *CPLEX* (resp. *PS* algorithm). Let *CPLEX*-solution (I_k) be the optimal non-preemptive solution of an instance I_k. Let PS-solution (I_k) be the optimal preemptive solution of an instance I_k. The values of $\boldsymbol{GAP_1}$ and $\boldsymbol{GAP_2}$ are calculated for N instances I_k, ($k = \overline{1..N}$), as follows:

$$\boldsymbol{GAP_1} = \frac{\sum_{k=1}^{N}(\frac{NPS\ \text{solution}(I_k) - \text{PS-solution}(I_k)}{\text{PS-solution}(I_k)} * 100)}{N}$$

$$\boldsymbol{GAP_2} = \frac{\sum_{k=1}^{N}(\frac{NPS\ \text{solution}(I_k) - CPLEX\text{-solution}(I_k)}{CPLEX\text{-solution}(I_k)} * 100)}{N}$$

Finally, column $\boldsymbol{\sigma_1}$ (resp. $\boldsymbol{\sigma_2}$) presents the standard deviation between the solutions obtained by NPS algorithm and the solution obtained by *CPLEX* (resp. *PS* algorithm). The values of $\boldsymbol{\sigma_1}$ and $\boldsymbol{\sigma_2}$ are calculated for N instances I_k ($k = \overline{1..N}$) as follows:

$$\boldsymbol{\sigma_1} = \sqrt{\frac{\sum_{k=1}^{N}(\frac{NPS\ \text{solution}(I_k) - \text{PS-solution}(I_k)}{\text{PS-solution}(I_k)} * 100)^2}{N}}$$

$$\boldsymbol{\sigma_2} = \sqrt{\frac{\sum_{k=1}^{N}(\frac{NPS\ \text{solution}(I_k) - CPLEX\text{-solution}(I_k)}{CPLEX\text{-solution}(I_k)} * 100)^2}{N}}$$

Obtaining the optimal solution for large instances using *CPLEX* is very expensive in running time. We have therefore limited the running time for *CPLEX* to 60 minutes.

Table 2.3: *Evaluation of the NPS heuristic compared to CPLEX.*

Instances	***PS***	***CPLEX***			**NPS**						
		Sol	**Time**	**Opt**	**Sol**	**Time**	**opt**	$\boldsymbol{GAP_1}$	$\boldsymbol{\sigma_1}$	$\boldsymbol{GAP_2}$	$\boldsymbol{\sigma_2}$
test_8_3	37.312	51.06	0.03s	✓	51.54	0.000054s	29	40.43%	57.87%	0.71%	3.91%
test_12_3	54.40	77.82	0.059s	✓	78.16	0.000053s	28	44.94%	56.35%	0.22%	1.14%
test_15_3	58.30	78.38	0.096s	✓	78.62	0.000049s	29	34.26%	40.89%	0.22%	1.22%
test_20_4	48.71	63.37	0.441s	✓	63.98	0.000060s	25	31.29%	39.45%	0.69%	2.21%
test_30_6	37.46	42.54	7.80s	✓	42.54	0.000069s	30	14.59%	21.52%	0%	0%
test_50_6	65.08	77.27	15m 43s	✓	77.97	0.000034s	28	20 %	28.60%	0.61%	2.46%
test_70_9	43.72	46.98	1h 8min	✓	46.98	0.0000245s	30	7.66 %	9.74 %	0%	0%
test_100_9	56.06	181.52	60 min	/	60.15	0.0009s	/	7.59%	10.15%	/	/
test_200_9	134.688	/	/	/	148.15	0.000040s	/	9.84%	13.00%	/	/
test_1000_11	502.78	/	/	/	522.19	0.000076s	/	3.83%	4.57%	/	/
test_10000_11	5077.38	/	/	/	5117.61	0.000457s	/	0.82%	0.88%	/	/
Average	564.09	/	/	/	571.62	0.00016s	94.76 %	19.56%	25.72%	/	/

In Table 2.3, we can notice that for the most of the instances with less than 70 tasks, our algorithm provides the optimal solution (94.76%) with a smaller running time than *CPLEX*.

From the instances *test*_100_9 to *test*_10000_11, *CPLEX* can not provide the optimal solution after one hour, whereas NPS provides a solution in less than one second for an instance

with 10000 tasks. Furthermore, the average GAP (resp. standard deviation) of the solutions obtained by NPS algorithm for these instances is given by 5.52% (resp. $7,15\%$) compared to the optimal preemptive solution obtained by PS algorithm. This means that on average, the solution obtained by NPS algorithm is smaller than 1.1 the lower bound, and thus the optimal solution.

2.7 Conclusion

This chapter presents an efficient approximation algorithm to solve a task scheduling problem on heterogeneous platforms for the particular case of linear chains of tasks (sequential applications). Our objective is to minimize the total execution time (makespan) by respecting an energy constraint on the total energy consumed by the system. This work has shown that finding the optimal scheduling is not easy. Tests on large instances close to reality shows the limits of solving the problem with a solver such as *CPLEX*.

The main contribution of this work is an algorithm which provides a solution with small running time, and also guarantees the quality of the solution obtained compared to the optimal solution. The ratio obtained depends on the frequencies of two successive processing elements p_k and p_{k+1} used to provide the an optimal preemptive scheduling. We proved that the ratio between $\widehat{C}_{max}$ the solution obtained by using NPS Algorithm 2 and the optimal solution $C^{\star}_{max}$ can be bounded by the ratio between the frequencies of these two processing elements, i.e., $\frac{\widehat{C}_{max}}{C^{\star}_{max}} \leqslant \frac{f_{k+1}}{f_k}$. The results of this chapter were published in [70].

As part of the future, it could be interesting to focus on more general classes of graphs like DAG. The use of the preemptive scheduling has proven to be effective for an application represented by a linear chain of tasks. Using the same approach on all graph paths could provide an effective method. Furthermore, it could be interesting to test NPS algorithm on real sequential applications en real platforms.

In this chapter, we have seen that the scheduling of particular cases of applications (linear chain of tasks) on heterogeneous platforms can be handled effectively if they satisfy the properties of the consistent model. Unfortunately, this is not always the case, where the execution model of several applications is inconsistent. In the following chapter, we are interested in the more general case, which is the scheduling of applications under inconsistent model. We will focus on hybrid platforms, and will show that scheduling on this kind of platform can be effective if we use more specific methods designed for hybrid platforms.

Chapter

3 Scheduling parallel applications on hybrid platforms with an unlimited number of processors

Chapter content

3.1 Introduction

This chapter addresses the problem of scheduling parallel applications onto a particular case of HPC platforms composed of two different types of resources like CPU and GPU; these platforms are often called hybrid platforms.

We suppose that our hybrid platform is composed of an unlimited number of two types of processors. We also assume that there are no communication delays between the processors of the same type.

This assumption is motivated by tightly coupled multiprocessor systems which contain multiple processing elements that are connected and may have access to a central shared memory. This may make the communication cost negligible. If communication delays on the processing elements of the same type are taken into account, our results can be seen as a first phase of a new scheduling method. This method may be completed by a scheduling algorithm for identical machines applied to the processing elements of each type.

One direction where we believe our results can be used is the convergence Big Data-HPC. Several supercomputing centers have started to implement a convergence between Big-Data/Cloud and HPC, where numerous *small* applications are running on a supercomputer [71–73]. In this context, cloud applications are treated as second class citizen. They can use the computing power not being used by actual HPC applications. From a typical Cloud-Computing application, the number of nodes needed for each of its tasks is again many orders of magnitude below that of a supercomputer. Hence, one could expect that the main challenge for those jobs will

be to determine the type of node that they need (and its property), rather than focusing on the number of available nodes.

The problem treated in this chapter corresponds to solving a scheduling problem on two types of identical parallel processing elements $(P, P)|prec, com|C_{max}$ as defined in section 1.5.3.

The rest of this chapter is organized as follows. After presenting the used notations in the next section, we discuss in Section 3.3 the works related to the problem we address. Then, after studying the complexity of the problem in Section 3.4 and proving that the problem is NP-complete, we provide in Section 3.5, 3.6 and 3.7 some polynomial algorithms for special cases. Finally, we provide concluding remarks and future directions in Section 3.8

3.2 Notations

We suppose in this chapter that our hybrid platform is composed of an unlimited number of two types of processors denoted by $\mathcal{A}$ and $\mathcal{B}$. We assume that there are no communication delays between the processors of the same type: for all $(t_i, t_j) \in E$, for all $(p_1, p_2) \in \mathcal{A} \times \mathcal{A}$ or $(p_1, p_2) \in \mathcal{B} \times \mathcal{B}$, $cm_{i,1,j,2} = 0$.

We also assume that the communication cost between processors of different types are identical: for all $(t_i, t_j) \in E$, for all $(p_1, p_2) \in \mathcal{A} \times \mathcal{B}$ and $(p_3, p_4) \in \mathcal{B} \times \mathcal{A}$, $cm_{i,1,j,2} = cm_{i,2,j,1} = cm_{i,3,j,4} = cm_{i,4,j,3}$. To simplify these notations, in the rest of this thesis, the communication costs for hybrid platforms will be denoted by $cm_{i,j}$ for all $(t_i, t_j) \in E$.

For any value $u \in \{\mathcal{A}, \mathcal{B}\}$, $\bar{u} = \mathcal{A}$ if $u = \mathcal{B}$, and $\bar{u} = \mathcal{B}$ otherwise. For any task t_i, $w_{i,\mathcal{A}}$ (resp. $w_{i,\mathcal{B}}$) is the execution time of t_i on a processor of type $\mathcal{A}$ (resp. $\mathcal{B}$). We denote by C_i the completion time of the task t_i. For any arc $(t_i, t_j) \in E$, $C_i + w_{j,u} \leq C_j$ if both tasks are executed on the same type of processors $u \in \{\mathcal{A}, \mathcal{B}\}$. Otherwise, $C_i + w_{j,u} + cm_{i,j} \leq C_j$ where t_j is executed to a processor of type u and t_i to a processor of type $\bar{u}$.

The problem addressed here consists on allocating tasks to processors such that the overall makespan C_{max} is minimized. Since there is an unbounded number of processors of each type, it corresponds to finding an allocation $\theta : V \to \{\mathcal{A}, \mathcal{B}\}$ of all tasks on each type of processors. For an allocation θ and a path $p = t_1 \to t_2 \to \cdots \to t_p$ of the graph $\mathcal{G} = (V, E)$, we define the length of the path p according to θ by $\texttt{len}(p, \theta) = w_{1,\theta(t_1)} + \mathbb{1}_{\theta(t_1) \neq \theta(t_2)} cm_{1,2} + w_{2,\theta(t_2)} + \cdots + w_{p,\theta(t_p)}$. The makespan $MS(\mathcal{G}, \theta)$ is then obtained by computing the longest path of the graph $\mathcal{G}$ including the corresponding duration of the tasks and the communication costs: $MS(\mathcal{G}, \theta) = \max_{p \in \{\text{paths of } \mathcal{G}\}} \texttt{len}(p, \theta)$.

3.3 Related work

Recently, the problem of scheduling tasks on hybrid parallel platforms (2 types of homogeneous machines) has attracted a lot of attention. In the case where all processors have the same processing power and there is a cost for any communication $(P|prec, com|C_{max})$, the problem has been shown to be NP-hard [29] even for identical processing times. We discuss in this section the different works related to scheduling either with two types of machines, with an unbounded number of processors, or/and with communication costs.

Hybrid parallel platforms Different works have considered the problem of heterogeneous parallel platforms, where there are h-type of homogeneous machines, each with a limited number of processors in order to minimize the makespan. Even with no communication delays, the problem is NP-hard if the number of processors is limited [31].

Several works have studied the problem of scheduling independent tasks on ℓ (resp. k) processors of type $\mathcal{A}$ (resp. $\mathcal{B}$) which is represented by $(P\ell, Pk)||C_{max}$. Imreh [74] proves that the greedy algorithm provides a solution with a performance guarantee of $2 + \frac{\ell - 1}{k}$, where $k \leqslant \ell$.

Recently, a 2-approximation algorithm has been proposed by Marchal et al. [75]. For the same problem, Kedad-Sidhoum et al. [76] proposed two families of approximation algorithms that can achieve an approximation ratio smaller than $\frac{3}{2} + \epsilon$. By considering precedence constraints without communication delays $(P\ell, Pk)|prec|C_{max}$, Kedad-Sidhoum et al. [77] developed a tight 6-approximation algorithm for general structure graphs on hybrid parallel multi-core machines, composed of CPUs with additional accelerators (GPUs). This work was later revisited by Amaris et al.[78] who showed that by separating the allocation phase and the scheduling phase, they could obtain algorithms with a similar approximation ratio but that performs significantly better in practice.

Greedy algorithm:

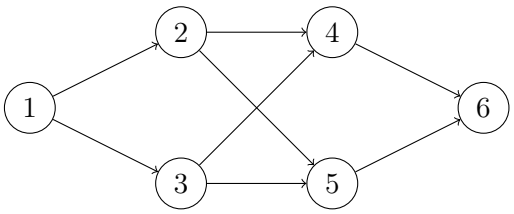
A greedy algorithm is an algorithmic paradigm that follows the problem solving heuristic of making the locally optimal choice at each step with the intent of finding a global optimum[79]. A greedy strategy does not guarantee an optimal solution, but may yield locally optimal solutions that approximate a global optimal solution in a reasonable amount of time. For scheduling problems, a greedy algorithm assign each task to one of its favorite machines, chosen in a greedy strategy (minimum completion time for example).

Duplications to reduce communications Duplication has been often used in the context of scheduling with communication costs. Indeed, executing a task on more than one processor allows for its successor to be ready sooner by not waiting for the data transfer. In the context of an unlimited number of processors, one needs to consider a polynomial number of *performance profile* (computation/communication costs) for the problem to be in NP. Bajaj and Agrawal [80] proposed a task duplication based scheduling algorithm for network of heterogeneous systems (m unrelated processors) considering precedence constraints with communication delays $Rm|prec, com|C_{max}$. This algorithm combines cluster based scheduling and duplication based scheduling to find the optimal solution. A set of conditions on task computation and communication time must be satisfied. Using the same model, Colin and Chrétienne [81] proposed an optimal method with a $\mathcal{O}(n^2)$ complexity time when the communication delays are not too large compared to the computation costs. Darbha and Agrawal [82] proposed another optimal solution strategy named TDS algorithm (Task Duplication based Scheduling) with the same complexity time $\mathcal{O}(n^2)$. Later, Park and Choe [83] extended this work when the communications are significantly larger than computations. For the general case (no assumptions on communication and computation costs), Wu et al. [84] proposed a genetic algorithm approach. Kwok and Ahmad [85] wrote a large survey of many algorithms for DAG scheduling in the presence of communication delays with duplication and a limited number of processors, $Rm|prec, com|C_{max}$.

The closest to this work is the paper of Barthou and Jeannot [86] who studied the problem of minimizing the makespan on unbounded platforms. They provide a polynomial-time algorithm of complexity $\mathcal{O}(k^2|E| + k|V|)$ for an unlimited number of processors and $k \geq 2$ types of processors. We briefly describe their algorithm for $k = 2$. The idea is simply to build the earliest schedule which is optimum. Let us denote by $C_j(\mathcal{A})$ (resp. $C_j(\mathcal{B})$) the completion time of the task $t_j \in V$ on a processor of type $\mathcal{A}$ (resp. $\mathcal{B}$) following the earliest schedule. These values may be computed as follows:

- For any task $t_j \in V$ with $\Gamma^-(t_j) = \emptyset$, $C_j(\mathcal{A}) = w_{j,\mathcal{A}}$ and $C_j(\mathcal{B}) = w_{j,\mathcal{B}}$. The tasks without predecessors are executed on the best processor since the number of processors is not limited.
- Otherwise, we get $C_j(\mathcal{A}) = w_{j,\mathcal{A}} + \max_{t_i \in \Gamma^-(t_j)}(\min(C_i(\mathcal{A}), C_i(\mathcal{B}) + cm_{i,j}))$ and $C_j(\mathcal{B}) = w_{j,\mathcal{B}} + \max_{t_i \in \Gamma^-(t_j)}(\min(C_i(\mathcal{B}), C_i(\mathcal{A}) + cm_{i,j}))$, since the duplication of tasks is allowed.

Let us consider as an example an instance of the problem pictured by Figure 3.1.



$t_i \in V$	$w_{i,\mathcal{A}}$	$w_{i,\mathcal{B}}$
t_1	1	1
t_2	5	2
t_3	1	7
t_4	2	6
t_5	4	2
t_6	1	2

Figure 3.1: *A graph $\mathcal{G} = (V, E)$ and the durations of the tasks on different types of processors. The communication delay is $cm_{i,j} = 3$ for any arc $(t_i, t_j) \in E$.*

The dates of the task earliest schedules are pictured by Figure 3.2. For example, $C_1(\mathcal{A}) = w_{1,\mathcal{A}} = 1$ and $C_1(\mathcal{B}) = w_{1,\mathcal{B}} = 1$. For the task t_2, $C_2(\mathcal{A}) = w_{2,\mathcal{A}} + \min(C_1(\mathcal{A}), C_1(\mathcal{B}) + cm_{1,1}) = 5 + \min(1, 4) = 6$ and $C_2(\mathcal{B}) = w_{2,\mathcal{B}} + \min(C_1(\mathcal{B}), C_1(\mathcal{A}) + cm_{1,2}) = 2 + \min(1, 4) = 3$.

$t_i \in V$	$C_i(\mathcal{A})$	$C_i(\mathcal{B})$
t_1	1	1
t_2	6	3
t_3	2	8
t_4	8	11
t_5	10	7
t_6	11	13

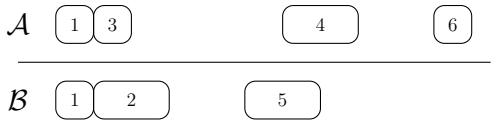


Figure 3.2: *The earliest schedule and a corresponding schedule with unnecessary duplicates removed. In grey, the values considered for the schedule.*

Some duplicates of tasks may be greedily removed without influence on the makespan. Let us denote $\Theta : V \times \{\mathcal{A}, \mathcal{B}\} \to \{0, 1\}$ such that, for any couple $(t_i, u) \in V \times \{\mathcal{A}, \mathcal{B}\}$, $\Theta(t_i, u) = 1$ if t_i is executed by u. We observe that $\Theta(t_i, u) \vee \Theta(t_i, \bar{u}) = 1$ since each task is allocated to $\mathcal{A}$, $\mathcal{B}$ or $\mathcal{A}$ and $\mathcal{B}$.

Let us suppose that the graph has a unique fictitious task t_n such that $C_{max} = \min\{C_n(\mathcal{A}), C_n(\mathcal{B})\}$. Then, let $u \in \{\mathcal{A}, \mathcal{B}\}$ such that $C_n(u) \leq C_n(\bar{u})$. We set $\Theta(t_n, u) = 1$ and $\Theta(t_n, \bar{u}) = 0$.

Now, let us consider a task $t_i \in V$ such that the allocation $\Theta(t_j, u), u \in \{\mathcal{A}, B\}$ is fixed for any successor $t_j \in \Gamma^+(t_i)$. Then, for $u \in \{\mathcal{A}, \mathcal{B}\}$, $\Theta(t_i, \bar{u}) = 0$ if the execution of t_j on u at time $C_j(u)$ is sufficient to get all the transfer properties true, that is:

- For any task $t_j \in \Gamma^+(t_i)$ with $\Theta(t_j, u) = 1$, $C_i(u) + w_{j,u} \leq C_j(u)$.
- For any task $t_j \in \Gamma^+(t_i)$ with $\Theta(t_j, \bar{u}) = 1$, $C_i(u) + w_{j,\bar{u}} + cm_{i,j} \leq C_j(\bar{u})$.

In the contrary case, we must set $\Theta(t_i, \bar{u}) = 1$.

Unnecessary duplicates of tasks are then removed using the reverse topological order. Right part of Figure 3.2 presents the earliest schedule with no additional duplicates. As an example, task t_3 has two successors t_4 and t_5 with $\Theta(t_4, \mathcal{A}) = 1$, $\Theta(t_4, \mathcal{B}) = 0$, $\Theta(t_5, \mathcal{A}) = 0$, $\Theta(t_5, \mathcal{B}) = 1$. Now, since $C_3(\mathcal{A}) + w_{4,\mathcal{A}} = 4 \leq C_4(\mathcal{A})$ and $C_3(\mathcal{A}) + w_{5,\mathcal{B}} + cm_{3,5} = 7 \leq C_3(\mathcal{B})$, we can set $\Theta(t_3, \mathcal{B}) = 1$. Note that the two duplicates of task t_1 are necessary. Indeed, observe that $C_1(\mathcal{A}) + w_{2,\mathcal{B}} + cm_{1,2} = 6 > C_2(\mathcal{B})$ thus $\Theta(t_1, \mathcal{B}) = 1$. On the same way, $C_1(\mathcal{B}) + w_{3,\mathcal{A}} + cm_{1,3} = 5 > C_3(\mathcal{A})$ thus $\Theta(t_1, \mathcal{A}) = 1$.

This small example illustrates that all the duplicates cannot necessarily be removed without increasing the makespan. Furthermore, duplication may lead to other problems, such as additional energy consumption and significant memory footprint. Thus, in this thesis, we consider the problem for which the duplication of tasks is not allowed.

Communication without duplication The most famous heuristic developed for the problem of DAG scheduling on heterogeneous platforms considering communication delays ($Rm|prec, com|C_{max}$) is Heterogeneous Earliest Finish Time algorithm (HEFT) [26]. It has no performance guarantee, but performs particularly well. Other heuristics for this problem can be roughly partitioned into two classes: clustering and list scheduling algorithms.

Clustering algorithms [87, 88] usually provide good solutions for communication-intensive graphs by scheduling heavily communicating tasks onto the same processor. After grouping tasks into a set of clusters using different clustering policies, clusters are mapped onto processors using communication sensitive or insensitive heuristics.

List scheduling algorithms [89] are often used to handle a limited number of processors. Most of them [90–93] can be decomposed in two main phases. The first one assigns priorities based on certain task properties, typically run time and/or communication delays. The second phase assigns tasks to processors following a priority list.

Experimentally, a comparison of different list scheduling algorithms can be found in the work of Kushwaha and Kumar [93]. Wang and Sinnen [94] provide also a wide comparison of clustering and list scheduling algorithms for a limited and unlimited number of processors.

Table 3.1: *List of works on different applications and platforms to minimize the execution time. We call hybrid a platform with k types of processors where there are no communication costs within a type.*

Application \ Platform	Unlimited processors		Limited processors	
	Homogeneous processors	**Hybrid platforms**	**Heterogeneous processors**	**Hybrid platforms**
Independent tasks	P (folklore)	P (folklore)	NP-c (folklore) [95]	2-approx [76**?**] ($k = 2$)
Dependant task without communication delays	P (folklore)	P (folklore)	NP-c (folklore) $2(K+1)-$approx [62]	NP-c (folklore) 6-approx [77, 78] ($k = 2$)
Dependant task with communication delays with duplication	NP-c (general) P (special cases): [81–83]	P [86] ($k = 2$)	NP-c [80, 84]	NP-c Heuristics [86]
Dependant task with communication delays without duplication	NP-c [88, 96] [97]	[This chapter]: NP-c P (special cases)	NP-c [87, 90, 93] [91, 92]	NP-c

Further models and results Considering an unlimited number of homogeneous processors, Giroudeau et al. [96] studied the problem of scheduling where all the tasks of the precedence graph have unit execution times and considered a fix large communication delay between each pair of successive tasks ($P|prec, com = c \geqslant 2|C_{max}$). They proposed a polynomial-time approximation algorithm with performance ratio $\frac{2(c+1)}{3}$, with $c \geqslant 2$. Using a platform composed of m processors with K different speeds, a $2(K+1)$-approximation algorithm has been developed

by Chudak and Shmoys [62] for the DAG scheduling problem without communication delays $Qm|prec|C_{max}$. For $K = 2$, the ratio is then given by 6 for $(P\ell, Pk)|prec|C_{max}$ problem. If there are no communication delays, the problem $R|prec|C_{max}$ is trivial, where each task is simply assigned to the fastest machine. For platforms with a limited number of processors, Lenstra et al. [95] provided a $\frac{3}{2}$-approximation algorithm for $Rm|prec|C_{max}$ problem. Recently, a survey was proposed by Beaumont et al. [98] for scheduling on Two Types of Resources. We summarize all references in Table 3.1 depending on the constraint models or platforms.

3.4 Complexity

We prove by the following theorem that the scheduling problem $(P, P)|prec, com|C_{max}$ is NP-Complete even for graphs of depth 3.

Theorem 3. *The problem of deciding whether an instance of our main problem has a schedule of length* 2 *is strongly NP-complete even for graphs of depth* 3*.*

3-SATISFIABILITY problem:

The boolean satisfiability problem is the problem of determining if there exists an interpretation that satisfies a given Boolean formula. N-SATISFIABILITY problem is a special case of satisfiability problem, represented by a boolean expression divided to clauses, such that every clause contains N terms. As an example, for $N = 3$, the following formula has 2 clauses and 3 variables x_1, x_2, x_3:

$$(\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee \bar{x}_3)$$

The question is to assign a True or False value to each variable x_i in order to make the whole formula true. The formula above is satisfiable: by choosing $x_1 = True$, $x_2 = True$ and $x_3 = True$, $(False \vee True \vee True) \wedge (True \vee False \vee False) = True$. The formula is not satisfiable for $x_1 = False$, $x_2 = True$ and $x_3 = True$.

Proof. We perform the reduction from the 3-SATISFIABILITY (3-SAT) problem which is known to be strongly NP-complete [57, 89]: given $\{C_1, \cdots, C_m\}$ be a set of disjunctive clauses where each clause contains exactly three literals over $X = \{x_1, \cdots, x_n\}$ a set of boolean variables. Is there a truth assignment to X such that each clause is satisfied?

We write each clause $C_i = \tilde{x}_{i_1} \vee \tilde{x}_{i_2} \vee \tilde{x}_{i_3}$ where $(x_{i_1}, x_{i_2}, x_{i_3}) \in X^3$, and $\tilde{x}_k = x_k$ or $\bar{x}_k$. We are looking for a truth assignment such that $\bigwedge_{i=1}^{m} C_i$ is true.

From an instance $\mathcal{I}_1$ of 3-SAT: $\{C_1, \cdots, C_m\}$ over $\{x_1, \cdots, x_n\}$, we construct the following instance $\mathcal{I}_2$ for our problem:

- For all $i \in \{1, \cdots, n\}$, we define 2 tasks $t_{0(i)}$ and $t_{\infty(i)}$, and an edge $(t_{0(i)}, t_{\infty(i)})$.
- Then for each clause $C_i = \tilde{x}_{i_1} \vee \tilde{x}_{i_2} \vee \tilde{x}_{i_3}$, 3 tasks $t_{i(i_1)}, t_{i(i_2)}, t_{i(i_3)}$ are created and the following set of edges: $\{(t_{i(i_1)}, t_{i(i_2)}), (t_{i(i_2)}, t_{i(i_3)}), (t_{i(i_1)}, t_{\infty(i_1)}), (t_{0(i_2)}, t_{i(i_2)}), (t_{0(i_3)}, t_{i(i_3)})\}$.
- For any $j \in \{1, \cdots, n\}$, $v_j^\star$ denotes the set of all the instantiations of x_j in $\mathcal{G}$.

Overall, the graph $\mathcal{G} = (V, E)$ of depth 3 has $2n + 3m$ vertices and $n + 5m$ edges.

We can verify that the depth of the graph is exactly three. Indeed, the paths of $\mathcal{G}$ are exactly:

$$\forall j \in \{1, \cdots, n\}, \quad (t_{0(j)} \to t_{\infty(j)}). \tag{3.1a}$$
$$\forall i \in \{1, \cdots, m\}, \quad \text{let } C_i = \tilde{x}_{i_1} \vee \tilde{x}_{i_2} \vee \tilde{x}_{i_3} \tag{3.1b}$$
$$\text{then:} \quad (t_{i(i_1)} \to t_{i(i_2)} \to t_{i(i_3)}); \tag{3.1c}$$
$$(t_{i(i_1)} \to t_{\infty(i_1)}); \tag{3.1d}$$
$$(t_{0(i_2)} \to t_{i(i_2)} \to t_{i(i_3)}); \tag{3.1e}$$
$$(t_{0(i_3)} \to t_{i(i_3)}). \tag{3.1f}$$

We then define the execution and communication costs that can be written as follows:

- $\forall j \in \{1, \cdots, n\},\ w_{\infty(j),\mathcal{A}} = w_{\infty(j),\mathcal{B}} = w_{0(j),\mathcal{A}} = w_{0(j),\mathcal{B}} = 0$;
- $\forall j \in \{1, \cdots, n\},\ cm_{0(j),\infty(j)} = 3$;
- for all edges $(t_{i(j)}, t_{\infty(j)}), (t_{0(j')}, t_{i'(j')}) \in E$, we add the communication costs:

$$cm_{i(j),\infty(j)} = cm_{0(j'),i'(j')} = 3.$$

Then for $C_i = \tilde{x}_{i_1} \vee \tilde{x}_{i_2} \vee \tilde{x}_{i_3}$, we define the execution time costs as follows:

$$w_{i(i_j),\mathcal{A}} = 1 - w_{i(i_j),\mathcal{B}} = \begin{cases} 1 & \text{if } \tilde{x}_{i_j} = \bar{x}_{i_j} \\ 0 & \text{if } \tilde{x}_{i_j} = x_{i_j} \end{cases} \tag{3.2}$$

Furthermore, we set $cm_{i(i_1),i(i_2)} = cm_{i(i_2),i(i_3)} = 0$. Finally, in the instance $\mathcal{I}_2$, we want to study whether there exists a schedule θ whose makespan is not greater than 2.

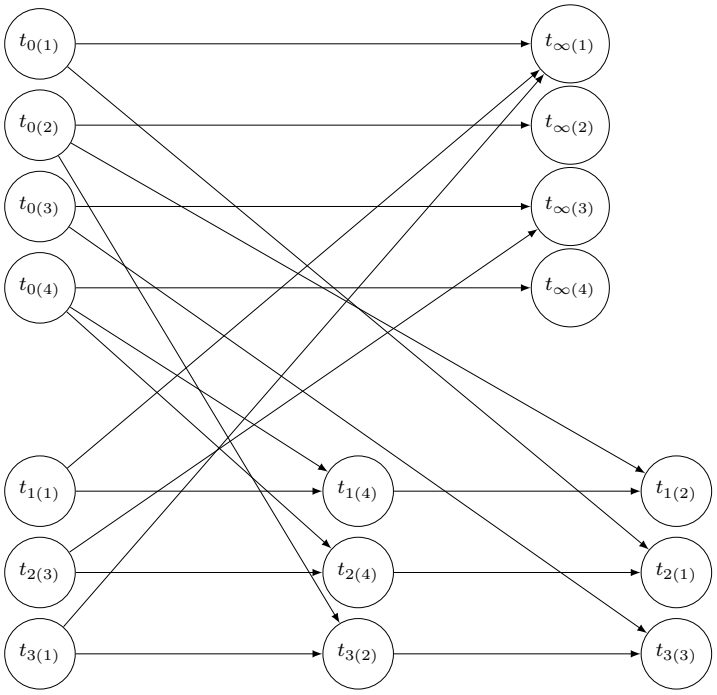


Figure 3.3: *Transformation of* $(x_1 \vee \bar{x}_4 \vee x_2) \bigwedge (\bar{x}_3 \vee \bar{x}_4 \vee x_1) \bigwedge (x_1 \vee x_2 \vee x_3)$ *(*$m = 3$ *clauses,* $n = 4$ *variables) into the associated graph* $\mathcal{G} = (V, E)$.

In Figure 3.3, we show an example of the construction of the associated graph for the transformation of $(x_1 \vee \bar{x}_4 \vee x_2) \bigwedge (\bar{x}_3 \vee \bar{x}_4 \vee x_1) \bigwedge (x_1 \vee x_2 \vee x_3)$ ($m = 3$ clauses, $n = 4$ variables):

- Here, the clause $C_1 = x_1 \vee \bar{x}_4 \vee x_2$ is associated with the vertices $t_{1(1)}$, $t_{1(4)}$ and $t_{1(2)}$ and the arcs set $\{(t_{1(1)}, t_{1(4)}), (t_{1(4)}, t_{1(2)}), (t_{1(1)}, t_{\infty(1)}), (t_{0(4)}, t_{1(4)}), (t_{0(2)}, t_{1(2)})\}$.
- Moreover, $w_{1(1),\mathcal{A}} = w_{1(2),\mathcal{A}} = 0$, $w_{1(4),\mathcal{A}} = 1$, $w_{1(1),\mathcal{B}} = w_{1(2),\mathcal{B}} = 1$ and $w_{1(4),\mathcal{B}} = 0$.

Note that the sets of all the instantiations of x_j in $\mathcal{G}$, $j = \overline{1..4}$, are given by:

$$v_1^\star = \{t_{0(1)}, t_{\infty(1)}, t_{1(1)}, t_{2(1)}, t_{3(1)}\}$$
$$v_2^\star = \{t_{0(2)}, t_{\infty(2)}, t_{1(2)}, t_{3(2)}\}$$
$$v_3^\star = \{t_{0(3)}, t_{\infty(3)}, t_{2(3)}, t_{3(3)}\}$$
$$v_4^\star = \{t_{0(4)}, t_{\infty(4)}, t_{1(4)}, t_{2(4)}\}$$

Let $\mathcal{S}$ be the set of schedules such that, $\forall \theta \in \mathcal{S}$, all tasks from $v_j^\star$ are scheduled by the same type of machines, *i.e*, for any couple $(t_{\alpha(j)}, t_{\beta(j)}) \in v_j^\star \times v_j^\star$, $\theta(t_{\alpha(j)}) = \theta(t_{\beta(j)})$. The next lemmas provide dominance properties on feasible schedules of $\mathcal{I}_2$.

Lemma 6. *Any feasible solution θ of $\mathcal{I}_2$ belongs to $\mathcal{S}$.*

Proof. Let us suppose by contradiction that a feasible solution $\theta \notin \mathcal{S}$. Two cases must then be considered:

- If there exists $j \in \{1, \cdots, n\}$ with $\theta(t_{0(j)}) \neq \theta(t_{\infty(j)})$, then there is a communication delay of 3 between them and $\texttt{len}(t_{0(j)} \to t_{\infty(j)}, \theta) = 3$.
- Otherwise, $\forall j \in \{1, \cdots, n\}, \theta(t_{0(j)}) = \theta(t_{\infty(j)})$. Thus, there exists a task $t_{i(j)}$ with $\theta(t_{i(j)}) \neq \theta(t_{0(j)})$. If $t_{i(j)}$ is associated to the first term of the clause C_i, then $(t_{0(j)}, t_{i(j)}) \in E$ and $\texttt{len}(t_{0(j)} \to t_{i(j)}, \theta) = 3$. Otherwise, $(t_{i(j)}, t_{\infty(j)}) \in E$ and $\texttt{len}(t_{i(j)} \to t_{\infty(j)}, \theta) = 3$.

The makespan of θ is at least 3 in both cases, the contradiction. □

Lemma 7. *For any schedule $\theta \in \mathcal{S}$, the makespan is obtained by computing the longest path $\texttt{len}(t_{i(i_1)} \to t_{i(i_2)} \to t_{i(i_3)}, \theta)$ of the graph $\mathcal{G}$, $i \in \{1, \cdots, m\}$, i.e., $MS(\mathcal{G}, \theta) = \max_{i \in \{1,\cdots,m\}} \texttt{len}(t_{i(i_1)} \to t_{i(i_2)} \to t_{i(i_3)}, \theta)$.*

Proof. To do this, we study the length of paths of $\mathcal{G}$.

- Let $j \in \{1, \cdots, n\}$, $\texttt{len}(t_{0(j)} \to t_{\infty(j)}, \theta) = 0$ since $\theta(t_{0(j)}) = \theta(t_{\infty(j)})$ by Lemma 6.
- Let $i \in \{1, \cdots, m\}$ associated with the clause $C_i = \tilde{x}_{i_1} \vee \tilde{x}_{i_2} \vee \tilde{x}_{i_3}$:

  1. Let us consider first the path $t_{i(i_1)} \to t_{\infty(i_1)}$. By Lemma 6, $\theta(t_{i(i_1)}) = \theta(t_{\infty(i_1)})$ and thus $cm_{i(i_1),\infty(i_1)} = 0$. Since $\texttt{len}(t_{\infty(i_1)}, \theta) = 0$,
  $$\texttt{len}(t_{i(i_1)} \to t_{\infty(i_1)}, \theta) = \texttt{len}(t_{i(i_1)}, \theta) \leq \texttt{len}(t_{i(i_1)} \to t_{i(i_2)} \to t_{i(i_3)}, \theta).$$
  2. Let us consider now the path $t_{0(i_2)} \to t_{i(i_2)} \to t_{i(i_3)}$. Similarly, $\theta(t_{0(i_2)}) = \theta(t_{i(i_2)})$ hence
  $$\texttt{len}(t_{0(i_2)} \to t_{i(i_2)} \to t_{i(i_3)}, \theta) = \texttt{len}(t_{i(i_2)} \to t_{i(i_3)}, \theta) \leq \texttt{len}(t_{i(i_1)} \to t_{i(i_2)} \to t_{i(i_3)}, \theta).$$
  3. Lastly, for the path $(t_{0(i_3)} \to t_{i(i_3)})$, since $\theta(t_{0(i_3)}) = \theta(t_{i(i_3)})$,
  $$\texttt{len}(t_{0(i_3)} \to t_{i(i_3)}, \theta) = \texttt{len}(t_{i(i_3)}, \theta) \leq \texttt{len}(t_{i(i_1)} \to t_{i(i_2)} \to t_{i(i_3)}, \theta),$$
  which concludes the lemma.

□

Assume that λ is a solution of $\mathcal{I}_1$, let us show that the schedule defined as follow, $\forall j \in \{1, \cdots, n\}$, $\forall t_{\alpha(j)} \in v_j^\star$,

$$\theta_\lambda : t_{\alpha(j)} \mapsto \begin{cases} \mathcal{A} & \text{if } \lambda(x_j) = 1 \\ \mathcal{B} & \text{if } \lambda(x_j) = 0 \end{cases}$$

has a makespan not greater than 2 and thus is a solution. Following Lemma 7, we must prove that $\forall i \in \{1, \cdots, n\}$, $\texttt{len}(t_{i(i_1)} \to t_{i(i_2)} \to t_{i(i_3)}, \theta_\lambda) \leq 2$.

For any clause $C_i = \tilde{x}_{i_1} \vee \tilde{x}_{i_2} \vee \tilde{x}_{i_3}$, since $\lambda(C_i) = 1$, there exists $j \in \{1, 2, 3\}$ such that $\lambda(\tilde{x}_{i_j}) = 1$. Two cases must be considered:

1. If $\tilde{x}_{i_j} = x_{i_j}$, then by definition, $w_{i(i_j),\mathcal{A}} = 0$. Since $\lambda(x_{i_j}) = 1$, $\theta_\lambda(t_{i(i_j)}) = \mathcal{A}$ and thus $\texttt{len}(t_{i(i_j)}, \theta_\lambda) = w_{i(i_j),\mathcal{A}} = 0$.

2. Otherwise, $\tilde{x}_{i_j} = \bar{x}_{i_j}$ and $w_{i(i_j),\mathcal{B}} = 0$. Now, as $\lambda(x_{i_j}) = 0$, $\theta_\lambda(t_{i(i_j)}) = \mathcal{B}$ and thus $\texttt{len}(t_{i(i_j)}, \theta_\lambda) = w_{i(i_j),\mathcal{B}} = 0$.

$\texttt{len}(t_{i(i_j)}, \theta_\lambda) = 0$ in both cases, so $\texttt{len}(t_{i(i_1)} \to t_{i(i_2)} \to t_{i(i_3)}, \theta_\lambda) \leq 2$.

Assume now that we have a solution θ of $\mathcal{I}_2$, let us show that $\lambda_\theta(x_j) = [\theta(t_{\infty(j)}) = \mathcal{A}]$ is a solution to $\mathcal{I}_1$. Following Lemma 6, $\theta \in \mathcal{S}$. Moreover, for any clause $C_i = \tilde{x}_{i_1} \vee \tilde{x}_{i_2} \vee \tilde{x}_{i_3}$, the corresponding path of $\mathcal{G}$ verifies $\texttt{len}(t_{i(i_1)} \to t_{i(i_2)} \to t_{i(i_3)}, \theta) \leq 2$. Thus, there is $j \in \{1, 2, 3\}$ with $\texttt{len}(t_{i(i_j)}, \theta) = 0$. Two cases must be considered:

1. If $\tilde{x}_{i_j} = x_{i_j}$, then by definition, $w_{i(i_j),\mathcal{A}} = 0$ and $w_{i(i_j),\mathcal{B}} = 1$. So, $\theta(t_{i(i_j)}) = \mathcal{A}$ and thus $\lambda_\theta(x_{i_j}) = 1$.

2. Else, $\tilde{x}_{i_j} = \bar{x}_{i_j}$ and thus $w_{i(i_j),\mathcal{A}} = 1$ and $w_{i(i_j),\mathcal{B}} = 0$. So, $\theta(t_{i(i_j)}) = \mathcal{B}$ and thus $\lambda_\theta(\bar{x}_{i_j}) = 1$.

So, at least one term of C_i is true following λ_θ, λ_θ is then a solution to $\mathcal{I}_1$. This concludes the proof that the problem is strongly NP-complete. □

Corollary 1. *There is no polynomial-time algorithm for the problem with a performance bound smaller than $\frac{3}{2}$ unless $\mathcal{P} = \mathcal{NP}$.*

Proof. By contradiction, let us suppose that there exists a polynomial-time algorithm with a performance ratio $\rho < \frac{3}{2}$. This algorithm can be used to decide the existence of a schedule with a length at most 2 for any instance $\mathcal{I}$. We deduce that there exists a polynomial time algorithm to decide the existence of a schedule of length strictly less than 3, which contradicts Theorem 3. □

We have shown that the problem is NP-complete if the graph has depth 3. The natural question that arises is whether it is already NP-complete for lower-depth graphs or particular classes of graph. In the following, we provide some polynomial time algorithms for special cases of graphs: bi-partite graphs, trees and series-parallel graphs.

3.5 Bi-partite graphs

Bi-partite graph:

In the mathematical field of graph theory, a Bi-partite graph (or bigraph) is a connected graph whose vertices can be divided into two disjoint and independent sets V_1 and V_2. Every arc connects a vertex in V_1 to one in V_2.

We consider here a bi-partite graph $\mathcal{G} = (V, E)$ (e.g. see Figure 3.4). The idea of the algorithm is first to compute, for any couple of tasks $(t_i, t_j) \in E$ and classes $(u, v) \in \{\mathcal{A}, \mathcal{B}\}^2$, the value $D_{i,j}(u, v)$ defined as

$$D_{i,j}(u,v) = \begin{cases} w_{i,u} + w_{j,v} & \text{if } u = v \\ w_{i,u} + w_{j,v} + cm_{i,j} & \text{otherwise.} \end{cases}$$

Each value $D_{i,j}(u, v)$ corresponds to a lower bound of the schedule if $\theta(t_i) = u$ and $\theta(t_i) = v$. Thus, if there exists a schedule of length $C < D_{i,j}(u, v)$, then $\theta(t_i) \neq u$ or $\theta(t_j) \neq v$. The idea is then to remove successively highest values of $D_{i,j}(u, v)$ whereas the corresponding 2-Satisfiability (2-SAT) system remains feasible.

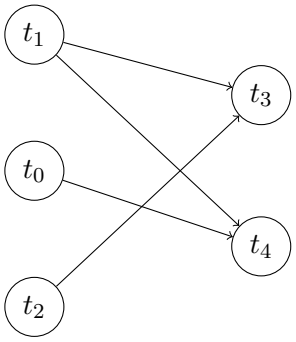


Figure 3.4: *A bi-partite graph*

Theorem 4. BiPartAlgo($\mathcal{G}$) *described below provides an optimal solution in polynomial time with a complexity of* $\Theta(n^4)$ *when* $\mathcal{G}$ *has depth 2.*

Observe that in the case of a bipartite graph $\mathcal{G} = (V, E)$, the paths are exactly the edges of $\mathcal{G}$. The intuition of the algorithm is then to compute first the makespan of all possible task allocations for all edges, and then to remove pairs associated to forbidden allocations.

For any edge $(t_i, t_j) \in E$, 4 allocations are possible: $(\theta(t_i), \theta(t_j)) \in \{\mathcal{A}, \mathcal{B}\}^2 = \{(\mathcal{A}, \mathcal{A}), (\mathcal{A}, \mathcal{B}), (\mathcal{B}, \mathcal{A}), (\mathcal{B}, \mathcal{B})\}$. We define the set of quintuplet of all these allocations:

$$\texttt{WgPaths} = \Big\{ (D_{i,j}(\theta(t_i), \theta(t_j)), t_i, t_j, \theta(t_i), \theta(t_j)) \big| (t_i, t_j) \in E, (\theta(t_i), \theta(t_j)) \in \{\mathcal{A}, \mathcal{B}\}^2 \Big\}.$$

This set can be constructed in linear time by a simple iteration through all the edges of the graph by a procedure that we call MkWgPaths(V, E).

Finally to minimize the makespan, we iteratively remove from `WgPaths` the allocations that would maximize the makespan and check that there still exists a possible schedule (each task t_i is affected to only one processor).

In the rest, we use the following notation for a schedule θ and a time D:

$$\begin{aligned} \mathrm{WP}(D) =& \big\{ (t_i, t_j, \sigma_i, \sigma_j) \text{ s.t. } (\ell, t_i, t_j, \sigma_i, \sigma_j) \in \texttt{WgPaths} \quad \text{and } \ell > D \big\} \\ P_D(\theta) =& \bigwedge_{(t_i,t_j,\sigma_i,\sigma_j) \in \mathrm{WP}(D)} [(\theta(t_i) \neq \sigma_i) \vee (\theta(t_j) \neq \sigma_j)] \end{aligned}$$

Intuitively, WP(D) is the set of paths and allocations of length greater than D, and the scheduling θ ignores all these paths.

Lemma 8. *Let* θ *be a schedule of makespan* D*, then* $P_D(\theta)$ *is satisfied.*

This result is a direct consequence of the fact that there should be no path of length greater than D. Hence, for $(t_i, t_j, \sigma_i, \sigma_j) \in \mathrm{WP}(D)$, we know that we do not have simultaneously in the schedule $(\theta(t_i) = \sigma_i)$ and $(\theta(t_j) = \sigma_j)$. Hence,

$$\neg \bigvee_{(t_i,t_j,\sigma_i,\sigma_j)\in \mathrm{WP}(D)} [(\theta(t_i) = \sigma_i) \wedge (\theta(t_j) = \sigma_j)] = \bigwedge_{(t_i,t_j,\sigma_i,\sigma_j)\in \mathrm{WP}(D)} [(\theta(t_i) \neq \sigma_i) \vee (\theta(t_j) \neq \sigma_j)] = P_D(\theta) \quad (3.4)$$

Algorithm 3: BiPartAlgo($\mathcal{G}$): polynomial algorithm for $\mathcal{G} = (V, E)$ a bipartite graph

```
1 begin
2     WgPaths ← MkWgPaths(G);
3     P_alg ← True; P_tmp ← True                    /* Two clauses with n variables */
4     for (ℓ, t_i, t_j, σ_i, σ_j) ∈ WgPaths, by decreasing value of ℓ do
5         P_tmp ← P_alg ∧ ((θ(t_i) ≠ σ_i) ∨ (θ(t_j) ≠ σ_j));
6         if P_tmp is not satisfiable then
7             Break
8         P_alg ← P_tmp;
9     θ(t_1), ··· , θ(t_n) ←Solve(P_alg)            /* Using a 2-SAT solver*/
```

2-Satisfiability:

2-Satisfiability is a computational problem of assigning values to variables, each of which has two possible values, in order to satisfy a system of constraints on pairs of variables. It may be solved with a complexity of $\mathcal{O}(n^3)$ [99], where n is the number of variables in the instance.

For example, let $x1 = (\theta(t_i) = \sigma_i)$ and $x2 = (\theta(t_j) = \sigma_j)$.The expression $F = (x1 \vee x2) \wedge (\bar{x1} \vee \bar{x2}) = ((\theta(t_i) = \sigma_i) \vee (\theta(t_j) = \sigma_j)) \wedge ((\theta(t_i) \neq \sigma_i) \vee (\theta(t_j) \neq \sigma_j))$ is satisfiable for $x1 = False$, $x2 = True$.

After calculating the decreasing order of `WgPaths` paths, Algorithm 3 eliminates the longest allocations $(\ell, t_i, t_j, \sigma_i, \sigma_j) \in$ `WgPaths`, by adding clauses $\big((\theta(t_i) \neq \sigma_i) \vee (\theta(t_j) \neq \sigma_j)\big)$ to P_{alg} while it remains satisfiable. Then, the obtained system P_{alg} is solved to get a feasible schedule.

Proof of Theorem 4. Consider an instance $\mathcal{G}$ of the problem. Let D_{alg} be the duration of the schedule returned by BiPartAlgo($\mathcal{G}$). Clearly, $D_{\mathrm{alg}} = \max_{(t_i,t_j)\in E}(w_{i,\theta(t_i)} + \mathbb{1}_{\theta(i)\neq\theta(j)} cm_{i,j} + w_{j,\theta(t_j)}$ (the paths of $\mathcal{G}$ are exactly the edges of $\mathcal{G}$). Let P_{alg} be the set of clauses computed by it (line 9). Let $W_{\mathrm{alg}} = \{(t_i, t_j, \sigma_i, \sigma_j) | (D_{i,j}(\sigma_i, \sigma_j), t_i, t_j, \sigma_i, \sigma_j) \in$ `WgPaths`$\}$ s.t. $P_{\mathrm{alg}} = \bigwedge_{(t_i,t_j,\sigma_i,\sigma_j)\in W_{\mathrm{alg}}} [(\theta(t_i) \neq \sigma_i) \vee (\theta(t_j) \neq \sigma_j)]$. Then by construction of P_{alg}, we have the following properties:

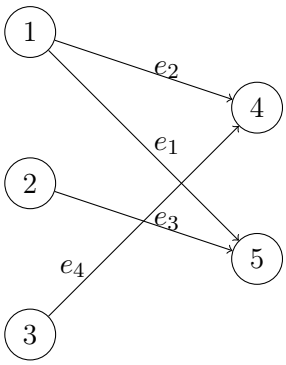
1. For all $\varepsilon > 0$, $\mathrm{WP}(D_{\mathrm{alg}}) \subset W_{\mathrm{alg}} \subset \mathrm{WP}(D_{\mathrm{alg}-\varepsilon})$, because we add paths by decreasing value of makespan (line 4).

2. There exists $(D_{\mathrm{alg}}, t_{i_0}, t_{j_0}, \sigma_{i_0}, \sigma_{j_0}) \in$ `WgPaths` such that P_{alg} is satisfiable and $P_{\mathrm{alg}} \bigwedge [(\theta(t_{i_0}) \neq \sigma_{i_0}) \vee (\theta(t_{j_0}) \neq \sigma_{j_0})]$ is not satisfiable. This is the stopping condition on line 7.

We show the optimality of Algorithm 3 by contradiction. If it is not optimal, then $D_{\text{opt}} < D_{\text{alg}}$, and $W_{\text{alg}} \cup (t_{i_0}, t_{j_0}, \sigma_{i_0}, \sigma_{j_0}) \subset \text{WP}(D_{\text{opt}})$. Furthermore, according to Lemma 8, $P_{D_{\text{opt}}}(\theta_{\text{opt}})$ is satisfied, hence σ_{opt} is also a solution to $P_{\text{alg}} \bigwedge [(\theta(t_{i_0}) \neq \sigma_{i_0}) \vee (\theta(t_{j_0}) \neq \sigma_{j_0})]$. This contradicts the fact that it does not admit a solution, hence contradicting the non-optimality.

□

Example 4. Figure 3.5 presents a bi-partite graph and the durations of the tasks on processors of both types $\mathcal{A}$ and $\mathcal{B}$.



$t_i \in V$	$w_{i,\mathcal{A}}$	$w_{i,\mathcal{B}}$
t_1	3	2
t_2	5	1
t_3	2	7
t_4	5	4
t_5	2	7

$D_{i,j}(u,v)$	e_1	e_2	e_3	e_4
$D_{i,j}(\mathcal{A},\mathcal{A})$	5	8	7	7
$D_{i,j}(\mathcal{A},\mathcal{B})$	13	10	15	9
$D_{i,j}(\mathcal{B},\mathcal{B})$	9	6	8	11
$D_{i,j}(\mathcal{B},\mathcal{A})$	7	9	6	15

Figure 3.5: *A bi-partite graph $\mathcal{G} = (V, E)$ on the left. Durations of the tasks on different type of processors in the middle. The communication cost is $cm_{i,j} = 3$ for any arc $(t_i, t_j) \in E$. On the right, computation of bounds $D_{i,j}(u, v)$, for $(t_i, t_j) \in E$ and classes $(u, v) \in \{\mathcal{A}, \mathcal{B}\}^2$.*

If we remove $D_{3,4}(\mathcal{B}, \mathcal{A}) = 15$, then we cannot have any more $\theta(t_3) = \mathcal{B}$ and $\theta(t_4) = \mathcal{A}$, thus we add the clause $\theta(t_3) = \mathcal{B} \vee \theta(t_4) = \mathcal{A}$. By removing $D_{3,4}(\mathcal{B}, \mathcal{A}) = 15$, $D_{2,5}(\mathcal{A}, \mathcal{B}) = 15$, $D_{1,5}(\mathcal{A}, \mathcal{B}) = 13$, $D_{3,4}(\mathcal{B}, \mathcal{B}) = 11$,$D_{1,4}(\mathcal{A}, \mathcal{B}) = 10$, $D_{3,4}(\mathcal{A}, \mathcal{B}) = 9$, $D_{1,5}(\mathcal{B}, \mathcal{B}) = 9$ and $D_{1,4}(\mathcal{B}, \mathcal{A}) = 9$, we get the following set of clauses:

$$\begin{cases} \theta(t_3) \neq \mathcal{B} \vee \theta(t_4) \neq \mathcal{A} \\ \theta(t_2) \neq \mathcal{A} \vee \theta(t_5) \neq \mathcal{B} \\ \theta(t_1) \neq \mathcal{A} \vee \theta(t_5) \neq \mathcal{B} \\ \theta(t_3) \neq \mathcal{B} \vee \theta(t_4) \neq \mathcal{B} \\ \theta(t_1) \neq \mathcal{A} \vee \theta(t_4) \neq \mathcal{B} \\ \theta(t_3) \neq \mathcal{A} \vee \theta(t_4) \neq \mathcal{B} \\ \theta(t_1) \neq \mathcal{B} \vee \theta(t_5) \neq \mathcal{B} \\ \theta(t_1) \neq \mathcal{B} \vee \theta(t_4) \neq \mathcal{A} \end{cases}$$

Setting $\theta(t_i) \neq u$ to $\theta(t_i) = \bar{u}$ for any couple $(t_i, u) \in V \times \{\mathcal{A}, \mathcal{B}\}$, we can transform the precedent system to get:

$$\begin{cases} \theta(t_3) = \mathcal{A} \vee \theta(t_4) \neq \mathcal{A} \\ \theta(t_2) \neq \mathcal{A} \vee \theta(t_5) = \mathcal{A} \\ \theta(t_1) \neq \mathcal{A} \vee \theta(t_5) = \mathcal{A} \\ \theta(t_3) = \mathcal{A} \vee \theta(t_4) = \mathcal{A} \\ \theta(t_1) \neq \mathcal{A} \vee \theta(t_4) = \mathcal{A} \\ \theta(t_3) \neq \mathcal{A} \vee \theta(t_4) = \mathcal{A} \\ \theta(t_1) = \mathcal{A} \vee \theta(t_5) = \mathcal{A} \\ \theta(t_1) = \mathcal{A} \vee \theta(t_4) \neq \mathcal{A} \end{cases}$$

A solution to this system is given by $\theta(t_1) = \mathcal{A}$, $\theta(t_2) = \mathcal{B}$, $\theta(t_3) = \mathcal{A}$, $\theta(t_4) = \mathcal{A}$ and $\theta(t_5) = \mathcal{B}$. The corresponding schedule has thus a length equal to $D_{1,4}(\mathcal{B}, \mathcal{A}) = 9$.

Now, if we remove $D_{1,4}(\mathcal{A}, \mathcal{A}) = 8$, we add to this system the clause $\theta(t_1) \neq \mathcal{A} \vee \theta(t_4) \neq \mathcal{A}$. Observe that, if $\theta(t_4) = \mathcal{A}$, we must set $\theta(t_1) \neq \mathcal{A}$ and we cannot verify the last clause

$\theta(t_1) = \mathcal{A} \vee \theta(t_4) \neq \mathcal{A}$. Now, if $\theta(t_4) \neq \mathcal{A}$, then the 6th clause $\theta(t_3) \neq \mathcal{A} \vee \theta(t_4) = \mathcal{A}$ implies that $\theta(t_3) \neq \mathcal{A}$ whereas the 4th one $\theta(t_3) = \mathcal{A} \vee \theta(t_4) = \mathcal{A}$ implies that $\theta(t_3) = \mathcal{A}$. Thus, the new system has no solution and the previous allocation is optimum, i.e, $\theta(t_1) = \mathcal{A}$, $\theta(t_2) = \mathcal{B}$, $\theta(t_3) = \mathcal{A}$, $\theta(t_4) = \mathcal{A}$ and $\theta(t_5) = \mathcal{B}$.

Complexity:

The complexity of MkWgPaths(V, E) is $\mathcal{O}(|E|)$. In Algorithm 3, the decreasing order of `WgPaths` paths is calculated with a $\mathcal{O}(n\log_2(n))$ time complexity. The loop for (line 4) is unwound at most $3|E|$ times, and we verify if P_{tmp} is satisfiable in line 6 with a complexity of $\mathcal{O}(k)$, where $k \leqslant 4|E|$ is the number of clauses is P_{tmp}. Then, since $|E| \leqslant n^2$, then $|E|^2 \leqslant n^4$, and the complexity of the loop for is $\mathcal{O}(n^4)$. Finally, Since the complexity of solving P_{alg} is $\mathcal{O}(n^3)$, the complexity of Algorithm 3 is $\mathcal{O}(n^4)$.

3.6 Trees

Out-tree graph:

In graph theory, an out-tree $\mathcal{G} = (V, E)$ is a directed graph in which, for a vertex $u \in V$ called the root and any other vertex $v \in V$, there is exactly one directed path from u to v [100].

Let us suppose now the DAG $\mathcal{G} = (V, E)$ is an out-tree rooted by $t_1 \in V$. For any task $t_i \in V$, the sub-tree rooted by t_i is the sub-graph $\mathcal{G}_i$ of $\mathcal{G} = (V, E)$ which vertices are t_i and the successors of t_i. Figure 3.6 represents an example of an out-tree graph. Figure 3.7 (resp. 3.8) represents the sub-tree rooted by t_2 (resp. t_3).

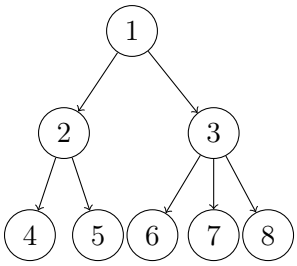


Figure 3.6: *An out-tree graph $\mathcal{G}$*

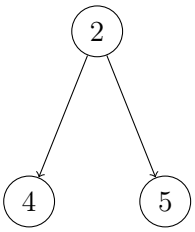


Figure 3.7: *$\mathcal{G}_2$ is the sub-tree rooted by t_2.*

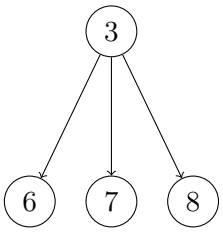


Figure 3.8: *$\mathcal{G}_3$ is the sub-tree rooted by t_3.*

For any task $u \in V$, let us denote by $D^{\mathcal{A}}(u)$ (resp. $D^{\mathcal{B}}(u)$) the lower bound of the minimal makespan of $\mathcal{G}_u$ assuming that $\theta(u) = \mathcal{A}$ (resp. $\theta(u) = \mathcal{B}$). For any allocation function θ, let $\bar{\theta}(u) = \mathcal{A}$ if $\theta(u) = \mathcal{B}$, $\bar{\theta}(u) = \mathcal{B}$ otherwise. Then, for any task $u \in V$, we get $D^{\theta(u)}(u) = w_{u,\theta(u)} + \max_{v \in \Gamma^+(u)} \min(D^{\theta(u)}(v), cm_{u,v} + D^{\bar{\theta}(u)}(v))$.

Let us suppose that the arc $(u, v) \in E$. Observe that, if $D^{\mathcal{A}}(v) \leq cm_{u,v} + D^{\mathcal{B}}(v)$, then $D^{\mathcal{A}}(u) \geq w_{u,\mathcal{A}} + D^{\mathcal{A}}(v)$. In the opposite, $D^{\mathcal{A}}(u) \geq w_{u,\mathcal{A}} + cm_{u,v} + D^{\mathcal{B}}(v)$ and thus $D^{\mathcal{A}}(u) \geq w_{u,\mathcal{A}} + \min(D^{\mathcal{A}}(v), cm_{u,v} + D^{\mathcal{B}}(v))$. Similarly, $D^{\mathcal{B}}(u) \geq w_{u,\mathcal{B}} + \min(D^{\mathcal{B}}(v), cm_{u,v} + D^{\mathcal{A}}(v))$.

Theorem 5. *For an out-tree graph $\mathcal{G} = (V, E)$ rooted by $r \in V$, an allocation θ may be built such that the corresponding schedule of length $D(r)$ verifies $D(r) = \min(D^{\mathcal{A}}(r), D^{\mathcal{B}}(r))$ and thus is optimal.*

Proof. Let us suppose that lower bounds $D^{\mathcal{A}}(u)$ and $D^{\mathcal{B}}(u)$ for $u \in V$ are given. Let us define the allocation θ as $\theta(r) = \mathcal{A}$ if $D^{\mathcal{A}}(r) \leq D^{\mathcal{B}}(r)$ and $\theta(r) = \mathcal{B}$ in the opposite. For any task

$v \neq r$ with $(u, v) \in E$, and an already fixed value $\theta(u)$ (equal to A or B), we set $\theta(v) = \theta(u)$ if $D^{\theta(u)}(v) < D^{\bar{\theta}(u)}(v) + cm_{u,v}$, and $\theta(v) = \bar{\theta}(u)$ otherwise.

For any task u, we prove that the length $D(u)$ of the schedule of $\mathcal{G}_u$ for the allocation θ verifies $D(u) = D^{\theta(u)}(u)$. If u is a leaf, $D(u) = w_{u,\theta(u)} = D^{\theta(u)}(u)$.

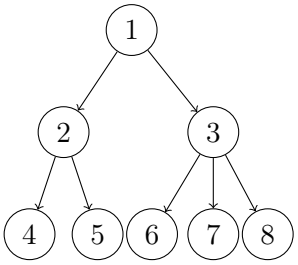
Now, let us suppose that $\Gamma^+(u) \neq \emptyset$. By definition, for any arc $(u, v) \in E$, if $\theta(u) = \theta(v)$, $cm_{u,v} = 0$. Then, if we set $\Delta^\theta(u, v) = D(v) + cm_{u,v}$, we get by induction $\Delta^\theta(u, v) = D^{\theta(v)}(v) + cm_{u,v}$ and by definition of θ, $\Delta^\theta(u, v) = \min(D^{\theta(u)}(v), D^{\bar{\theta}(u)}(v) + cm_{u,v})$. Now, $D(u) = w_{u,\theta(u)} + \max_{v \in \Gamma^+(u)} \Delta^\theta(u, v)$ and thus by definition of $D^{\theta(u)}$, $D(u) = D^{\theta(u)}$, which concludes the proof.

□

Corollary 2. *By Theorem 5, a polynomial time algorithm of time complexity $\mathcal{O}(n)$ can be deduced by computing first $D^{\mathcal{A}}$, $D^{\mathcal{B}}$ and then θ.*

Example 5. Let us consider as an example an instance of our problem pictured by Figure 3.9. Figure 3.10 presents the lower bound $D^{\theta(u)}(u)$ for $(u, \theta(u)) \in V \times \{\mathcal{A}, \mathcal{B}\}$. It may easily be computed from the leaves to the root t_1. For our example, let us consider t_3:

$$\begin{aligned} D^{\mathcal{A}}(t_3) &= 4 + \max_{t_j \in \{t_6, t_7, t_8\}} (\min(D^{\mathcal{A}}(t_j), D^{\mathcal{B}}(t_j) + cm_{3,j}) \\ &= 4 + \max(\min(5, 4), \min(2, 8), \min(3, 3)) = 4 + \max(4, 2, 3) = 8. \end{aligned}$$



$t_i \in V$	$w_{i,\mathcal{A}}$	$w_{i,\mathcal{B}}$
t_1	2	3
t_2	3	2
t_3	4	2
t_4	5	2
t_5	3	3
t_6	5	1
t_7	2	5
t_8	3	1

$(t_i, t_j) \in E$	$cm_{i,j}$
(t_3, t_6)	3
(t_3, t_7)	3
(t_3, t_8)	2
(t_2, t_4)	2
(t_2, t_5)	3
(t_1, t_2)	1
(t_1, t_3)	4

Figure 3.9: *An out-tree graph $\mathcal{G}$, with the durations of the tasks and communication costs.*

Let $D^\star(i)$ be the duration of $\mathcal{G}_i$ for an optimal schedule of $\mathcal{G}$. It is defined by its corresponding allocation function θ as follows:

- If $D^{\mathcal{A}}(t_1) \leq D^{\mathcal{B}}(t_1)$, we set $\theta(t_1) = \mathcal{A}$ and $D^\star(1) = D^{\mathcal{A}}(t_1)$, otherwise $\theta(t_1) = \mathcal{B}$ and $D^\star(1) = D^{\mathcal{B}}(t_1)$.

- Let us consider now a task t_j with $j > 1$ and the (unique) task t_i with $(t_i, t_j) \in E$. Let us consider $\sigma \in \{\mathcal{A}, \mathcal{B}\}$ such that $\theta(t_i) = \sigma$. If $D^\sigma(j) < D^{\bar{\sigma}}(j) + cm_{i,j}$, then t_j must be allocated to σ, thus $\theta(t_j) = \sigma$ and then $D^\star(j) = D^\sigma(j)$. Otherwise, t_j must be allocated to $\bar{\sigma}$, thus $\theta(t_j) = \bar{\sigma}$ and then $D^\star(j) = D^{\bar{\sigma}}(j)$.

$t_i \in V$	$D^{\mathcal{A}}(t_i)$	$D^{\mathcal{B}}(t_i)$
t_1	10	10
t_2	7	5
t_3	8	7
t_4	5	2
t_5	3	3
t_6	5	1
t_7	2	5
t_8	3	1

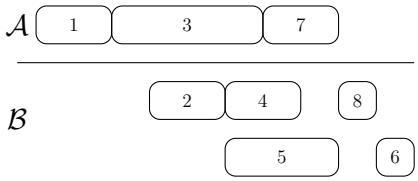


Figure 3.10: *Lower bounds $D^{\sigma}(i)$ for $(t_i, \sigma) \in V \times \{\mathcal{A}, \mathcal{B}\}$. An optimal allocation is associated with the values in gray. $\theta(t_1) = \mathcal{A}$, $\theta(t_2) = \mathcal{B}$, $\theta(t_3) = \mathcal{A}$, $\theta(t_4) = \mathcal{B}$, $\theta(t_5) = \mathcal{B}$, $\theta(t_6) = \mathcal{B}$, $\theta(t_7) = \mathcal{A}$ and $\theta(t_8) = \mathcal{B}$.*

For our example, $D^{\mathcal{A}}(t_1) \leq D^{\mathcal{B}}(t_1)$. Thus, we set for the root $\theta(t_1) = \mathcal{A}$ and $D^{\star}(t_1) = D^{\mathcal{A}}(t_1) = 10$. For t_2, $D^{\mathcal{A}}(t_2) = 7$ and $D^{\mathcal{B}}(t_2)+cm_{1,2} = 5+1 = 6$. Thus, $D^{\mathcal{A}}(t_2) > D^{\mathcal{B}}(t_2)+cm_{1,2}$ and t_2 must be allocated to $\mathcal{B}$. We set $\theta(t_2) = \mathcal{B}$ and $D^{\star}(t_2) = D^{\mathcal{B}}(t_2) = 5$. For t_3, $D^{\mathcal{A}}(t_3) = 8$ and $D^{\mathcal{B}}(t_3) + cm_{1,3} = 7 + 4 = 11$. Then, $D^{\mathcal{A}}(t_3) \leq D^{\mathcal{B}}(t_3) + cm_{1,3}$ and t_3 must be allocated to $\mathcal{A}$. We set $\theta(t_3) = \mathcal{A}$ and $D^{\star}(t_3) = D^{\mathcal{A}}(t_3) = 8$. Figure 3.10 presents the allocation of the tasks and a corresponding schedule of minimal length equal to $D^{\star}(t_1) = 10$.

3.7 Series-Parallel graphs

Let us consider a two terminal Series Parallel digraph ($2SP$ in short) as defined in [101, 102]. Each element of this class has a unique source s and a unique sink t with $s \neq t$.

It is formally defined as follows where $\mathcal{G}$ and $\mathcal{H}$ are two $2SP$ graphs.

- The arc $(s, t) \in 2SP$ as presented in Figure 3.11;

Figure 3.11: *A $2SP$ graph with only two tasks s and t.*

- The series composition of $\mathcal{G}$ (e.g. see Figure 3.12) and $\mathcal{H}$ (e.g. see Figure 3.13) is denoted by $\mathcal{G}.\mathcal{H}$ (e.g. see Figure 3.14) and is built by identifying the sink of $\mathcal{G}$ with the source of $\mathcal{H}$;

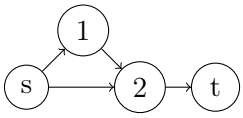


Figure 3.12: *The graph $\mathcal{G}$.*

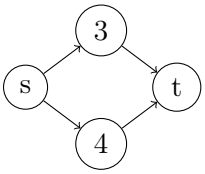


Figure 3.13: *The graph $\mathcal{H}$.*

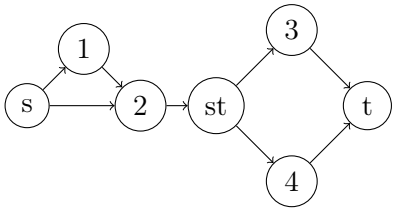


Figure 3.14: *$\mathcal{G}.\mathcal{H}$: A $2SP$ graph (series composition).*

- The parallel composition of $\mathcal{G}$ (e.g. see Figure 3.12) and $\mathcal{H}$ (e.g. see Figure 3.13) is denoted by $\mathcal{G} + \mathcal{H}$ (e.g. see Figure 3.15) and identifies respectively the sinks and the sources of the two digraphs.

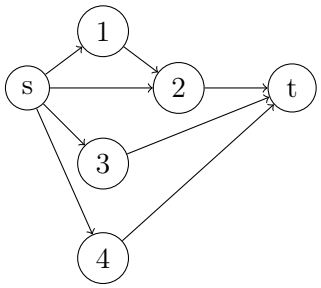


Figure 3.15: *$\mathcal{G} + \mathcal{H}$: A 2SP graph (parallel composition).*

Definition 1 (Lower bound)**.** *For any element $\mathcal{G} \in 2SP$ with a source s and a sink t and for any couple $(\alpha, \beta) \in \{\mathcal{A}, \mathcal{B}\}^2$, let us denote by $D^{\alpha\beta}(\mathcal{G})$ a lower bound defined as follows of the minimum length of a schedule of $\mathcal{G}$ with $\theta(s) = \alpha$ and $\theta(t) = \beta$.*

- *For any graph $\mathcal{G}$ with a unique arc $e = (s, t)$, for any couple $(\alpha, \beta) \in \{\mathcal{A}, \mathcal{B}\}^2$,*

$$D^{\alpha\beta}(\mathcal{G}) = \begin{cases} w_{s,\alpha} + w_{t,\beta} + cm_{s,t} & \text{if } \alpha \neq \beta \\ w_{s,\alpha} + w_{t,\beta} & \text{otherwise.} \end{cases}$$

- *Now, if $\mathcal{G}$ and $\mathcal{H}$ are two 2SP, then for the series composition, we set $D^{\alpha\beta}(\mathcal{G}.\mathcal{H}) = \min_{\gamma \in \{\mathcal{A},\mathcal{B}\}}(D^{\alpha\gamma}(\mathcal{G}) + D^{\gamma\beta}(\mathcal{H}) - w_{t,\gamma})$ where t is the sink of $\mathcal{G}$ ($w_{t,\gamma}$ computed in $D^{\alpha\gamma}(\mathcal{G})$ and $D^{\gamma\beta}(\mathcal{H})$).*

- *Similarly, for the parallel composition, we set $D^{\alpha\beta}(\mathcal{G} + \mathcal{H}) = \max(D^{\alpha\beta}(\mathcal{G}), D^{\alpha\beta}(\mathcal{H}))$.*

Definition 2 (Allocation function)**.** *We define the allocation function θ associated with a 2SP graph $\mathcal{G}$ and the corresponding length $D(\mathcal{G})$ as follows. We set $D(\mathcal{G}) = \min_{(\alpha,\beta)\in\{\mathcal{A},\mathcal{B}\}^2}(D^{\alpha\beta}(\mathcal{G}))$. We also set $\theta(s)$ and $\theta(t)$ the allocation function of the source and the sink of $\mathcal{G}$ as $D(\mathcal{G}) = D^{\theta(s)\theta(t)}(\mathcal{G})$.*

Now, for any series composition, let us suppose that s and t (resp. s' and t') are the source and the sink of $\mathcal{G}$ (resp. $\mathcal{H}$). We also suppose that $\theta(s)$ and $\theta(t')$ are fixed. Then, for $\mathcal{G}.\mathcal{H}$, $t = s'$ and we get $\theta(t) = \gamma \in \{\mathcal{A}, \mathcal{B}\}$ such that $D(\mathcal{G}.\mathcal{H}) = D^{\theta(s)\theta(t)}(\mathcal{G}) + D^{\theta(s')\theta(t')}(\mathcal{H}) - w_{t,\theta(t)}$.

If $\mathcal{G}$ is a $2SP$ graph of source s and sink t, any vertex $v \in V - \{s, t\}$ is involved in a series composition, and thus θ is completely defined.

Theorem 6. *For any $2SP$ graph $\mathcal{G}$ of source s and sink t, $D(\mathcal{G}) = D^{\theta(s)\theta(t)}(\mathcal{G})$.*

Proof. The equality is clearly true if $\mathcal{G}$ is an arc (s, t). Indeed, we get in this case $D(\mathcal{G}) = \min_{(\alpha,\beta)\in\{\mathcal{A},\mathcal{B}\}^2}(D^{\alpha\beta}(\mathcal{G})) = D^{\theta(s)\theta(t)}(\mathcal{G})$.

Now, let us suppose that s and t (resp. s' and t') are the source and the sink of $\mathcal{G}$ (resp. $\mathcal{H}$) and that $D(\mathcal{G}) = D^{\theta(s)\theta(t)}(\mathcal{G})$ and $D(\mathcal{H}) = D^{\theta(s')\theta(t')}(\mathcal{H})$. For a parallel composition, $D(\mathcal{G}+\mathcal{H}) = \max(D^{\theta(s)\theta(t)}(\mathcal{G}), D^{\theta(s')\theta(t')}(\mathcal{H})) = D^{\theta(s)\theta(t)}(\mathcal{G} + \mathcal{H})$ as $s = s'$ and $t = t'$.

For the series composition, $D(\mathcal{G}.\mathcal{H}) = D(\mathcal{G}) + D(\mathcal{H}) - w_{t,\theta(t)} = D^{\theta(s)\theta(t)}(\mathcal{G}.\mathcal{H})$, since $t = s'$, which concludes the proof. □

Corollary 3. *A polynomial-time algorithm of time complexity $\mathcal{O}(|E|)$ can be deduced by computing lower bounds $D^{\alpha\beta}$, $(\alpha, \beta) \in \{A, B\}^2$ for each graph issued from the decomposition of G and a corresponding allocation θ.*

Example 6. Let us consider as an example an instance of our problem pictured by Figure 3.16, which represents a $2SP$ graph and its associated decomposition tree. The tables presented in Figure 3.17 show the duration of tasks on each type of processors and the communication delays.

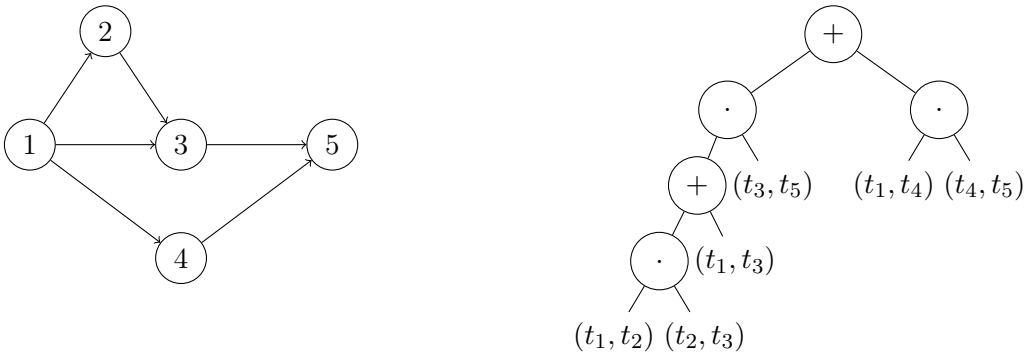


Figure 3.16: *A 2SP graph and its associated decomposition tree. Leaves correspond to arcs, while internal nodes represent series (.) or parallel (+) compositions.*

$t_i \in V$	$w_{i,\mathcal{A}}$	$w_{i,\mathcal{B}}$
t_1	5	3
t_2	4	2
t_3	1	5
t_4	2	5
t_5	3	3

$(t_i, t_j) \in E$	$cm_{i,j}$
(t_1, t_2)	3
(t_2, t_3)	2
(t_1, t_3)	1
(t_3, t_5)	4
(t_1, t_4)	3
(t_4, t_5)	5

Figure 3.17: *Duration of tasks following processors class and communication delays.*

Table 3.2 reports the computation of the lower bound $D_{\mathcal{G}}(\alpha, \beta)$, for any 2SP subgraph of $\mathcal{G} = (V, E)$ and any couple $(\alpha, \beta) \in \{\mathcal{A}, \mathcal{B}\}^2$. For example, for $\mathcal{G} = (V, E) = \mathcal{G}_1 = \mathcal{G}_2 + \mathcal{G}_3$,

$$D_{\mathcal{G}_1}(\mathcal{A}, \mathcal{B}) = \max(D_{\mathcal{G}_2}(\mathcal{A}, \mathcal{B}), D_{\mathcal{G}_3}(\mathcal{A}, \mathcal{B})) = \max(10, 15) = 15.$$

On the same way, for $\mathcal{G}_2 = \mathcal{G}_4.(\{t_3, t_5\}, \{(t_3, t_5)\})$, setting $\mathcal{H} = (\{t_3, t_5\}, \{(t_3, t_5)\})$, we get

$$\begin{aligned} D_{\mathcal{G}_2}(\mathcal{A}, \mathcal{B}) &= \min(D_{\mathcal{G}_4}(\mathcal{A}, \mathcal{A}) + D_{\mathcal{H}}(\mathcal{A}, \mathcal{B}) - w_{3,\mathcal{A}}, D_{\mathcal{G}_4}(\mathcal{A}, \mathcal{B}) + D_{\mathcal{H}}(\mathcal{B}, \mathcal{B}) - w_{3,\mathcal{B}} \\ &= \min(10 + 1 - 1, 15 + 8 - 5) = 10. \end{aligned}$$

$\mathcal{H}$	$D_{\mathcal{H}}(\mathcal{A}, \mathcal{A})$	$D_{\mathcal{H}}(\mathcal{A}, \mathcal{B})$	$D_{\mathcal{H}}(\mathcal{B}, \mathcal{A})$	$D_{\mathcal{H}}(\mathcal{B}, \mathcal{B})$
$(\{t_1, t_2\}, \{(t_1, t_2)\})$	9	10	10	5
$(\{t_2, t_3\}, \{(t_2, t_3)\})$	5	11	5	7
$(\{t_1, t_3\}, \{(t_1, t_3)\})$	6	11	5	8
$(\{t_3, t_5\}, \{(t_3, t_5)\})$	4	1	10	8
$(\{t_1, t_4\}, \{(t_1, t_4)\})$	7	13	8	8
$(\{t_4, t_5\}, \{(t_4, t_5)\})$	5	10	13	8
$\mathcal{G}_5$	10	15	8	10
$\mathcal{G}_4$	10	15	8	10
$\mathcal{G}_3$	10	15	11	11
$\mathcal{G}_2$	13	10	11	8
$\mathcal{G}_1$	13	15	11	11

Table 3.2: *Computation of the lower bounds $D_{\mathcal{G}}(\alpha, \beta)$, for any 2SP subgraph of $\mathcal{G} = (V, E)$ and any couple $(\alpha, \beta) \in \{\mathcal{A}, \mathcal{B}\}^2$. $\mathcal{G} = (V, E) = \mathcal{G}_1 = \mathcal{G}_2 + \mathcal{G}_3$, $\mathcal{G}_2 = \mathcal{G}_4.(\{t_3, t_5\}, \{(t_3, t_5)\})$, $\mathcal{G}_3 = (\{t_1, t_4\}, \{(t_1, t_4)\}).(\{t_4, t_5\}, \{(t_4, t_5)\})$, $\mathcal{G}_5 = (\{t_1, t_2\}, \{(t_1, t_2)\}).(\{t_2, t_3\}, \{(t_2, t_3)\})$ and $\mathcal{G}_4 = \mathcal{G}_5 + (\{t_1, t_3\}, \{(t_1, t_3)\})$.*

For our example, possible values for $D^\star_{\mathcal{G}}$ are presented in grey. Clearly, we have $D^\star_G = \min(13, 15, 11) = 11$, and this value is achieved for $D_G(\mathcal{B}, \mathcal{A})$, thus we set $\theta(t_1) = \mathcal{B}$, $\theta(t_5) = \mathcal{A}$ and $D^\star_G = D_G(\mathcal{B}, \mathcal{A})$. On the same way, $D^\star_{\mathcal{G}_2} = D_{\mathcal{G}_4}(\mathcal{B}, \mathcal{A}) + D_{\mathcal{H}}(\mathcal{B}, \mathcal{A}) - w_{3,\mathcal{A}} = 8 + 4 - 1$ with $\mathcal{H} = (\{t_3, t_5\}, \{(t_3, t_5)\})$. Thus, $\theta(t_3) = \mathcal{A}$, $D^\star_{\mathcal{G}_4} = D_{\mathcal{G}_4}(\mathcal{B}, \mathcal{A})$ and $D^\star_{\mathcal{H}} = D_{\mathcal{H}}(\mathcal{A}, \mathcal{A})$. One may also verify that $\theta(t_2) = \mathcal{B}$ and $\theta(t_3) = \mathcal{A}$.

3.8 Conclusion

In this chapter we have studied the problem of scheduling a DAG on an unbounded hybrid platform. Specifically, our platform consists of two types of processing elements, each with an unbounded number of resources. Moving data from one type to the other one has a communication cost. We have shown the intractability of the problem by reducing this problem to the 3-satisfiability problem. We have proved that there does not exist 3/2-approximation algorithms unless P=NP. We further provide some polynomial time algorithms for special cases of graphs. The results of this chapter were published in [103, 104]. While this model seems very theoretical, we can see several applications, both in High-Performance Computing (*In-Situ analysis*) and in Big Data analytics in the cloud (Geo-distributed data-Centers).

There are several extensions that we can see to this work. One direction we are interested by, is a version of this problem where only one type has an unbounded number of resources, and where the data is located on the other one. For example, in the context of smartphone applications, we can model the frontend/backend context where the phone (Machine 1) has a limited number of available processors, but can rely on sending some of the computations on a backend machine (cloud-based), with an unbounded number of processors. Similarly to here, the problem is a data and communication problem: given the cost to transfer data from one machine to the other one, what is the most efficient strategy?.

In the context of two unbounded platforms, it would be interesting to find some polynomial time algorithms with proven bounds to the optimal solution. We do not expect to be able to find one in the general case, but we hope that with some constraints between the communication costs and computation cost (*as is often done in the context of scheduling DAGs with communication delays*), one may be able to find such algorithms. In the following chapter, we present two efficient algorithms to solve the problem of scheduling DAG applications on a bounded hybrid platform with communication delays.

Chapter

4 Hybrid platform with a limited number of processors

Chapter content

4.1 Introduction

In the previous chapter, we have studied the problem of scheduling a DAG on an unbounded hybrid platform. Specifically, our platform consists of two types of processing elements, each with an unbounded number of resources.

In this chapter, we suppose that our hybrid platform is composed of a limited number of two types of processing elements denoted by $\mathcal{A}$ and $\mathcal{B}$ (NVIDIA Drive PX2 platform for example as presented in 1.2.2). We denote by ℓ (resp. k) the number of processing elements of type $\mathcal{A}$ (resp. $\mathcal{B}$). Let m be the total number of processing elements, $m = \ell + k$. For any value $u \in \{\mathcal{A}, \mathcal{B}\}$, $\bar{u} = \mathcal{A}$ if $u = \mathcal{B}$, and $\bar{u} = \mathcal{B}$ otherwise. For any task t_i, $w_{i,\mathcal{A}}$ (resp. $w_{i,\mathcal{B}}$) is the execution time of t_i on a processor of type $\mathcal{A}$ (resp. $\mathcal{B}$).

As defined in Chapters 1 and 3, we assume that there are no communication delays between the processors of the same type, i.e., for all $(t_i, t_j) \in E$, for all $(p_1, p_2) \in \mathcal{A} \times \mathcal{A}$ and $(p_3, p_4) \in \mathcal{B} \times \mathcal{B}$, $cm_{i,1,j,2} = cm_{i,2,j,1} = 0$ and $cm_{i,3,j,4} = cm_{i,4,j,3} = 0$. Furthermore, we assume that the communication cost between processors of different types are identical: for all $(t_i, t_j) \in E$, for all $(p_1, p_2) \in \mathcal{A} \times \mathcal{B}$ and $(p_3, p_4) \in \mathcal{B} \times \mathcal{A}$, $cm_{i,1,j,2} = cm_{i,2,j,1} = cm_{i,3,j,4} = cm_{i,4,j,3}$. To simplify these notations, in the rest of this thesis, the communication costs for hybrid platforms will be denoted by $cm_{i,j}$ for all $(t_i, t_j) \in E$.

Let C_i be the end date of the task t_i. For any arc $(t_i, t_j) \in E$, $C_i + w_{j,u} \leq C_j$ if both tasks are executed to a processor of type $u \in \{\mathcal{A}, \mathcal{B}\}$. Otherwise, $C_i + w_{j,u} + cm_{i,j} \leq C_j$ where t_j is executed to a processor of type u and t_j to a processor of type $\bar{u}$.

The problem addressed here consists on allocating tasks to processors such that the overall makespan C_{max} is minimized. Contrary to a platform with an unlimited number of processing elements, the number of tasks that can be executed in parallel here is limited. Even if several tasks can be executed in parallel, it is necessary to take into account the number of available processing elements and choose which tasks to execute in priority. Thus, it is important to properly use the number of each type of processing elements to exploit the parallelism effectively.

For this purpose, we propose two algorithms: a non polynomial-time algorithm with a performance ratio of 6 and a polynomial time algorithm with a relative performance guarantee. Both methods are based on a new scheduling technique that define the pre-allocation of tasks before the task scheduling phase. This technique allows us to have an extended view of the entire graph that represents the application. Thus, we can make the best decisions for the maximum number of tasks. Tests on large instances demonstrate that the proposed algorithms achieve close-to-optimal performances. The problem treated in this chapter corresponds to solving a scheduling problem on two types of parallel processing elements $(P\ell, Pk)|prec, com|C_{max}$ as defined in Section 1.5.3.

The rest of this chapter is organized as follows. The next section presents the mathematical model used to obtain the optimal solution for small instances. Section 4.3 presents some classical resolution methods based on list algorithms. Then, we detail in Section 4.4 the two proposed scheduling algorithms with a performance guaranty analysis. Finally, after testing the algorithms on several instances in Section 4.5, we provide concluding remarks and future directions in Section 4.6

4.2 Mathematical model

This section aims at providing a modelling of the scheduling problem using a Mixed Integer Program (MIP) for which the number of variables and constraints is polynomial.

Data:

A hybrid platform is composed of m resources $P = \{p_1, p_2, \cdots, p_m\}$ of two types: $\mathcal{A}$ and $\mathcal{B}$ (e.g. GPU+CPU). ℓ represents the number of processing elements of type $\mathcal{A}$ and k the number of processing elements of type $\mathcal{B}$, $m = \ell + k$. Let $P(\mathcal{A})$ (resp. $P(\mathcal{B})$) be the set of processors of type $\mathcal{A}$ (resp. $\mathcal{B}$). Clearly, $P(\mathcal{A}) = \{p_1, p_2, \cdots, p_\ell\}$ and $P(\mathcal{B}) = \{p_{\ell+1}, p_{\ell+2}, \cdots, p_m\}$. We denote by $\tau(r)$ the type of the processing element p_r, such that $\tau(r) = \mathcal{A}$ if $r \leqslant \ell$ and $\tau(r) = \mathcal{B}$ if $r > \ell$. We also use the two following data:

- $w_{i,\mathcal{A}}$ (resp. $w_{i,\mathcal{B}}$) is the execution time of t_i on a processor of type $\mathcal{A}$ (resp. $\mathcal{B}$).
- $cm_{i,j}$ is the communication delay between t_i and t_j if they are executed on two different types of processing element.

Variables:

Let consider the following decision variables:

- $x_{i,r}$ equal to 1 if the task t_i is assigned to the processor p_r, 0 otherwise, $i = \overline{1..n}$ and $r = \overline{1..m}$.
- S_i is the starting time of the task t_i, $i = \overline{1..n}$.
- $\theta(t_i)$ represents the processor assignment of t_i (e.g. $\theta(t_i) = p_1$ means that t_i is assigned to p_1).
- $o_{i,j}$ is an intermediary variable to manage overlapping tasks on the same processing element. For each couple of tasks $t_i \neq t_j$, $i = \overline{1..n}$ and $j = \overline{1..n}$, if t_i and t_j are executed by the same processing element, i.e., $\theta(t_i) = \theta(t_j)$, then:

$$o_{i,j} = \begin{cases} 1 & \text{if } t_j \text{ is executed after the completion time of } t_i \\ 0 & \text{if } t_i \text{ is executed after the completion time of } t_j. \end{cases}$$

Model:

We thus obtain the following model (Opt) composed of 4 constraints:

$$(Opt) \begin{cases} \sum_{r=1}^{m} x_{i,r} = 1, \forall i = \overline{1..n} & (1) \\ \\ S_i + x_{i,r} w_{i,\tau(r)} + (x_{i,r} + x_{j,u} - 1) cm_{i,j} \leqslant S_j & (2) \\ \forall \ (t_i, t_j) \in E, \ \ \forall r = \overline{1..m}, \ \ \forall u = \overline{1..m}, \ \tau(r) \neq \tau(u) & \\ \\ S_i + x_{i,r} w_{i,\tau(r)} \leqslant S_j + B \times (3 - x_{ir} - x_{jr} - o_{i,j}) & (3) \\ S_j + x_{j,r} w_{j,\tau(r)} \leqslant S_i + B \times (2 - x_{ir} - x_{jr} + o_{i,j}) & \\ \forall i, j = \overline{1..n} \quad i \neq j, \ (t_i, t_j) \notin E, \ \forall r = \overline{1..m}, B = Cte & \\ \\ S_i + \sum_{r=1}^{m} x_{i,r} w_{i,\tau(r)} \leqslant C_{max}, \forall i, \Gamma^+(t_i) = \emptyset & (4) \\ \\ x_{i,r} \text{ and } o_{i,j} \in \{0, 1\}, \ \forall i, j = \overline{1..n}, \ r = \overline{1..m} & \\ Z(min) = C_{max} & \end{cases}$$

Constraints:

- First constraint (1) simply expresses that each task must be executed only once on one processor.
- Constraint (2) describes that the task t_j must be executed after the completion time of the task t_i for each couple of tasks $(t_i, t_j) \in E$, and the communication cost $cm_{i,j}$ is added if they are executed on two different types of processing elements. If t_i is executed on p_r, and t_j on p_u and $\tau(r) \neq \tau(u)$, then $x_{i,r} + x_{j,u} - 1 = 1$. Otherwise, $x_{i,r} + x_{j,u} - 1 \leqslant 0$.
- Overlapping tasks on each processor are avoided by the constraint (3) using a large constant B, such that if couple of tasks t_i and t_j are executed on the same processor, then either t_j starts after the completion time of the task t_i or t_i starts after the completion time of the task t_j. For each couple of tasks t_i and t_j, $(t_i, t_j) \notin E$ (treated by constraint 2), if t_i and t_j are executed on the same processing element p_r, then $B \times (3 - x_{ir} - x_{jr} - o_{i,j}) = B \times (1 - o_{i,j})$ and $B \times (2 - x_{ir} - x_{jr} + o_{i,j}) = B \times (o_{i,j})$. Depending on the value of $o_{i,j}$, if $B \times (1 - o_{i,j}) = 0$ (because $o_{i,j} = 1$) then $B \times (o_{i,j}) = B$, and if $B \times (1 - o_{i,j}) = B$ (because $o_{i,j} = 0$) then $B \times (o_{i,j}) = 0$. Thus, only one constraint becomes important, the other one will always be feasible, such that:

– Either t_j starts after the completion of the task t_i, where $\begin{cases} S_i + x_{i,r} w_{i,\tau(r)} \leqslant S_j \\ S_j + x_{j,r} w_{j,\tau(r)} \leqslant S_i + B \end{cases}$

– Or t_i starts after the completion of the task t_j, where $\begin{cases} S_i + x_{i,r} w_{i,\tau(r)} \leqslant S_j + B \\ S_j + x_{j,r} w_{j,\tau(r)} \leqslant S_i \end{cases}$

- Constraint (4) describes that C_{max} is bigger than the completion time of the tasks without successors. In other words, C_{max} represents an upper bound for all paths of the graph, and thus it bounds the end of the last task completion time. By minimizing C_{max}, we minimize the completion time of the application.

Solvers like *CPLEX* can be used to find the optimal solution for small instances. However, in a practical way, generic approaches are used to solve large instances with a reasonable running time. A state of the art for the scheduling problem on platforms with limited and unlimited resources is provided in the previous chapter (Chapter 3). Nevertheless, in the following, we will detail some methods that give an evaluation overview of the scheduling methods on limited resource platforms.

4.3 Basic List Scheduling algorithm (without pre-allocation)

List scheduling algorithms are often used to handle a limited number of processors. For our general problem, the first known algorithm named Heterogeneous Earliest Finish Time (HEFT) [26] is decomposed into two main phases: task prioritizing phase, then processor selection phase. The first one assigns priorities based on certain task properties, typically run time and/or communication delays. The second phase assigns the tasks to the processors using a basic list scheduling algorithm. Several other authors improved the priority list without changing the global algorithm. This section aims at presenting HEFT algorithm and several more recent improvements.

4.3.1 The Heterogeneous-Earliest-Finish-Time (HEFT) Algorithm

HEFT algorithm [26] is considered as a reference method in the literature for its lower time complexity. It is based on two main phases:

1. **Task prioritizing:** this phase uses execution times and communication costs to calculate the ranks. Task priorities are defined as $rank(T) = \{rank(t_1), rank(t_2), \cdots, rank(t_n)\}$. $rank(t_i)$ represents a length evaluation of the longest path from the task t_i to the exit node, including the computational cost of t_i and is given by:

$$rank(t_i) = \frac{w_{i,\mathcal{A}} + w_{i,\mathcal{B}}}{2} + \max_{t_j \in \Gamma^+(t_i)} (cm_{i,j} + rank(t_j))$$

   The priority list $rank(T)$ is ordered by decreasing the values of $rank(t_i)$, $i = \overline{1..n}$.

2. **Processor selection phase:** in this phase, the first unscheduled task from the task priority list is selected and assigned to the processing element p_r that provides its Earliest Finish Time (EFT). However, HEFT algorithm uses an insertion policy that tries to insert a task at the earliest idle time between two already scheduled tasks on the processing element, if the slot is large enough to execute the task. Additionally, scheduling on this idle time slot should preserve precedence constraints. The search of an appropriate idle

time slot of a task t_i on a processing element p_r starts at the time equal to the ready time of t_i on p_r. Algorithm 4 provides the detailed description of the HEFT Algorithm.

Algorithm 4: HEFT Algorithm

Data: $rank(T) = \{rank(t_1), rank(t_2), \cdots, rank(t_n)\}$.
Result: Feasible scheduling.
begin
  Create an empty list ready-list;
  ready-list= $\{t_j,\ \Gamma^-(t_j) = \emptyset,\ j = \overline{1..n}\}$;
  while *ready-list* $\neq \emptyset$ **do**
    $t_i \longleftarrow$ task with highest $rank(t_i)$ from the ready-list;
    for *all* $p_r \in P$ **do**
      Compute $EFT(t_i, p_r)$ value using an insertion-based scheduling policy;
    Assign task t_i to the processor p_r that minimize EFT of task t_i;
    Update ready-list with unscheduled tasks which their predecessors were already executed;

Complexity:

Task priorities table is calculated for n tasks on two types of processing elements. For each task t_i (n in total), we calculate for all its successors $t_j \in \Gamma^+(t_i)$ ($n-1$ at most) the value $(cm_{i,j} + rank(t_j))$ and we take the maximum value. Thus, the complexity of the first phase is $\mathcal{O}(n^2)$. Furthermore, for each task (n in total), we calculate its EFT by taking into account all its predecessors ($n-1$ at most) on each available processing element (m at most). The insertion policy is verified on a processing element by checking the non-overlapping with at most $n-1$ tasks. Thus, the complexity of the second phase is $\mathcal{O}(n^2m)$. Finally, for our model, HEFT algorithm has a $\mathcal{O}(n^2m)$ time complexity.

4.3.2 Lookahead scheduling

HEFT algorithm does not consider more than one task during processor assignments. The assignment policy does not take into account the remaining unscheduled tasks. For a task t_i, HEFT algorithm chooses the best processor to minimize the EFT of t_i, but this assignment may be bad for the successors of t_i. Furthermore, it was observed that there are significant differences between the performance of HEFT depending on the scheme used for computing the task priorities list $rank(T)$ [105]. To enhance this strategy, Lookahead algorithm [90] presents an interesting improvement of HEFT, where the locally optimal decisions made by the heuristic do not rely on estimates of a single task only, but also look ahead in the schedule and take into account some information about the impact of this decision on the direct successors of the task being allocated. Therefore, it increases the temporal complexity of HEFT by the average number of direct successors per task.

Based on HEFT algorithm, its main characteristic is its processor selection policy. To select a processor for the current task t_i, it iterates on all available processors and calculates the EFT for all tasks $t_j \in \Gamma^+(t_i)$ on all processors. The processor selected for the task t_i is the one that minimizes the maximum EFT of all successors from t_i on all resources where t_i is tested. However, direct successors of a task t_i do not depend only on this task, and may not become ready immediately after scheduling t_i (as a result of a dependence to another unscheduled predecessors). Thus, to calculate the estimated finish time of the successors of t_i, Lookahead only consider their already scheduled predecessors and t_i, ignoring further delays that may arise due to any unscheduled predecessors.

Complexity:

Lookahead algorithm has the same structure as HEFT algorithm, but calculates in addition the EFT for each task $t_j \in \Gamma^+(t_i)$ ($n-1$ at most) of the current task t_i. The EFT of each task $t_j \in \Gamma^+(t_i)$ is calculated by taking into account all its predecessors $t_v \in \Gamma^-(t_j)$ ($n-1$ at most) on m processing elements. This increases the total complexity of the method to $\mathcal{O}(n^4m^2)$ for a single level of forecasting (only direct successors). The authors reported that additional levels of forecasting do not result in significant improvements for the makespan objective.

4.3.3 Predict Earliest Finish Time algorithm (PEFT)

Lookahead can significantly improve the schedule returned by HEFT, especially in cases where the communication delays are significant. However, one of the weaknesses of this method is that it selects a resource for a task t_i based on the estimated finish time of all its direct successors, even if they are not all on the critical path. Furthermore, the estimated finish times computed for the successors of a task t_i is rather optimistic, since they depend only on t_i and already scheduled predecessors.

PEFT algorithm [92] has tried to fix this problem by proposing the Optimistic Cost Table (OCT). This table is computed before scheduling and represents for each pair (task, processor) the minimum processing time of the longest path from the current task to the exit node by assigning the best processors to each of those tasks. The table is optimistic because it does not consider processor availability at a given time. The values stored in the cost table are used in the processor selection phase. Rather than considering only the EFT for the task that is being scheduled, PEFT adds to EFT the processing time stored in the table for the pair (task, processor). All processors are tested, and the one that gives the minimum value is selected. Thus, PEFT algorithm introduces the look ahead feature while maintaining quadratic complexity. The detailed description of the PEFT Algorithm for our model is given as follows:

1. Compute OCT table: the OCT value of a task t_i on a processor type $r \in \{\mathcal{A}, \mathcal{B}\}$ is recursively defined as follows by treating the DAG from the exit task to the entry one:

$$OCT(t_i, r) = \max_{t_j \in \Gamma^+(t_i)} [\min_{u \in \{\mathcal{A},\mathcal{B}\}} \{OCT(t_j, u) + w_{j,u} + \zeta_{i,j}\}]$$

   The communication cost between two tasks t_i and $t_j \in \Gamma^+(t_i)$ is given by $\zeta_{i,j} = cm_{i,j}$ if $u = r$, $\zeta_{i,j} = 0$ otherwise.

2. Compute $rank_{oct}$ for all tasks (Task Prioritizing): to define task priorities, the average OCT is computed for each task as follows:

$$rank_{oct}(t_i) = \frac{OCT(t_i, \mathcal{A}) + OCT(t_i, \mathcal{B})}{2}$$

   The priority list $rank_{oct}(T)$ is ordered by decreasing the values of $rank_{oct}(t_i)$, $i = \overline{1..n}$.

3. The following list algorithm is then applied:

Algorithm 5: PEFT Algorithm

Data: OCT table and $rank_{oct}(T)$.
Result: Feasible scheduling.
begin
  Create an empty list ready-list;
  ready-list= $\{t_j,\ \Gamma^-(t_j) = \emptyset,\ j = \overline{1..n}\}$;
  while *ready-list* $\neq \emptyset$ **do**
    $t_i \longleftarrow$ task with highest $rank_{oct}(t_i)$ from the ready-list;
    for *All* $p_r \in P$ **do**
      Compute $EFT(t_i, p_r)$ value using an insertion-based scheduling policy;
      $O_{EFT}(t_i, p_r) = EFT(t_i, p_r) + OCT(t_i, \tau(r))$;
    Assign task t_i to the processor p_r that minimize O_{EFT} of the task t_i;
    Update ready-list with unscheduled tasks;

Complexity:

OCT and task priorities table $rank_{oct}(T)$ are calculated for n tasks on the two types of processing elements with a complexity of $\mathcal{O}(n)$. Furthermore, the EFT value is calculated for each task (n in total) by taking into account all its predecessors ($n-1$ at most) on each available processing element (m at most). The insertion policy is verified on a processing element by checking the non-overlapping with at most $(n-1)$ tasks. Thus, the O_{EFT} value is calculated for each task with a complexity of $\mathcal{O}(n^2m)$. Finally, PEFT algorithm improves the scheduling provided by HEFT algorithm while maintaining the same time complexity of $\mathcal{O}(n^2m)$.

4.3.4 INCremental Subgraph Earliest Finish Time algorithm (INCSEFT)

The common point of the three methods defined above is that they assign a single task at each step. HEFT algorithm assigns task by task based only on task priorities. Lookahead algorithm tried to improve it by considering the direct successors of each task before its assignment. To do better, PEFT algorithm uses the optimistic cost table OCT in the processor selection phase. This table contains the minimum longest path from each task to the exit node. The table is optimistic because it does not consider processor availability at a given time. Thus, the initial OCT value of a task can change after the assignment of some tasks that precede it. These methods can be effective locally. But assigning only one task at each step can negatively influence overall scheduling. INCSEFT algorithm [106] integrates an extended view of the task graph and uses the notion of sub-graphs when assigning processors. It uses the notion of dynamically calculated critical paths using ranks based on average task execution costs.

To construct a sub-graph, ranks are assigned to the tasks which are processed in order of their ranks. These ranks are computed recursively in a bottom-up approach using the average execution cost of the current node and the accumulated average execution cost of its successors. At each step, a critical path is found by starting with the unscheduled task t_i having the highest rank. Then, an unscheduled task $t_j \in \Gamma^+(t_i)$ with the highest rank is added to the critical path. The critical path is formed when the last added task has no unscheduled successors.

The tasks on a critical path are added to a sub-graph $\widehat{\mathcal{G}}$ that grows gradually in an incremental way and then processed for the minimization of its completion time. At each step, the scheduling of the sub-graph $\widehat{\mathcal{G}}$ is calculated without taking into account the (other) tasks that are not in $\widehat{\mathcal{G}}$. For the first critical path, the best processor is assigned. For the subsequent critical paths, the processor assignment proceeds under the following two cases: all tasks in a critical path are assigned to the most appropriate processor if the new sub-graph scheduling does not exceed the

length of the previously calculated scheduling. Otherwise, a single task is assigned to the most appropriate processor, and the other tasks are removed from the sub-graph $\widehat{\mathcal{G}}$ to form another critical path in the next step. The algorithm ends when all the tasks are assigned, i.e., $\widehat{\mathcal{G}} = \mathcal{G}$. The detailed description of INCSEFT Algorithm for our model is given as follows:

1. Compute ranks of the tasks:

$$rank_{INCSEFT}(t_i) = \frac{w_{i,\mathcal{A}} + w_{i,\mathcal{B}}}{2}$$

   The list $rank_{INCSEFT}(T)$ is ordered by decreasing the values of $rank_{INCSEFT}(t_i)$, $i = \overline{1..n}$.

2. The following list algorithm is then applied:

Algorithm 6: INCSEFT Algorithm

Data: $rank_{INCSEFT}(T)$ table; $\mathcal{G} = (V, E)$.
Result: Feasible scheduling.
begin
- Create an empty list ready-list;
- ready-list= $\{t_j,\ \Gamma^-(t_j) = \emptyset,\ j = \overline{1..n}\}$;
- $t_i \longleftarrow$ task with highest $rank_{INCSEFT}(t_i)$ from the ready-list;
- $\widehat{\mathcal{G}}$ = the critical path of the graph $\mathcal{G}$ built from the tasks t_i to the end of the graph by adding highest rank tasks in the incremental way;
- All tasks of $\widehat{\mathcal{G}}$ are assigned to the best processor, let $\widehat{C}_{max}$ be its completion time;
- Update ready-list with unscheduled tasks;
- Create a sub-graph $\mathcal{G}'$ which does not contain the scheduled tasks, i.e., $\mathcal{G}' = \mathcal{G} \backslash \widehat{\mathcal{G}}$;
- **while** *ready-list* $\neq \emptyset$ **do**
  - $t_i \longleftarrow$ task with highest $rank_{INCSEFT}(t_i)$ from the ready-list;
  - CP = the critical path of the graph $\mathcal{G}'$ built from the tasks t_i to the end of the graph by adding highest rank tasks in the incremental way;
  - $\widehat{C}'_{max}$= makespan obtained by executing all the tasks $t_v \in CP$ on the best processor taking into account the scheduled tasks from $\widehat{\mathcal{G}}$;
  - **if** $\widehat{C}'_{max} \leqslant \widehat{C}_{max}$ **then**
    - Execute all tasks $t_v \in CP$ on the best processor;
    - $\mathcal{G}' \longleftarrow \mathcal{G}' \backslash CP$, $\widehat{\mathcal{G}} \longleftarrow \widehat{\mathcal{G}} \cup CP$;
  - **else**
    - Execute only the first task t_i of CP;
    - $\mathcal{G}' \longleftarrow \mathcal{G}' \backslash \{t_i\}$, $\widehat{\mathcal{G}} \longleftarrow \widehat{\mathcal{G}} \cup \{t_i\}$;
  - $\widehat{C}_{max}$ = completion time of $\widehat{\mathcal{G}}$;
  - Update ready-list with unscheduled tasks;

Complexity:

$rank_{INCSEFT}(T)$ table is calculated for n tasks on the two types of processing elements with a complexity of $\mathcal{O}(n)$. Furthermore, for each task t_i (n in total), the Critical Path CP starting by t_i is calculated by finding the tasks with the highest rank among the successors of t_i ($n-1$ at most). Then, a new makespan is calculated by executing all tasks $t_v \in CP$ (n at most) on each processing element (m in total) by taking into account the scheduled tasks ($n-1$ at most). Thus, the complexity time of INCSEFT algorithm is given by $\mathcal{O}(n^3 m)$.

INCSEFT algorithm produces very effective near-optimal schedules. Experiments were performed with a large number of task graphs using different topologies [106]. The INCSEFT strategy performs better by producing schedules with smaller length than HEFT and Lookahead scheduling strategies. Using an incremental approach minimizes the length of scheduling at each step while incorporating an extended view of the graph at each scheduling step. Its weakness is that it puts all the path on the same processor, which can be locally efficient since it avoids communication costs. However, this method treats paths separately, and assigning tasks from one path to the same processor may not arrange the tasks of another path.

The best way to do this is to deal with all the paths together, and make the decisions that arrange the greatest number of paths. However, doing this for any platform (with more than two types of processors) seems complicated. The particularity of a hybrid platform gives us the possibility to use efficient techniques as linear programming. In the following, a new scheduling strategy will be defined, which, unlike the previous methods, makes a pre-allocation before starting scheduling.

4.4 Basic List Scheduling algorithm with pre-allocation

The common point of the methods mentioned above is that they decide during the scheduling phase of the processor that will execute each task. Thus, the assignment of a task t_i (or the entire path that starts with t_i as in INCSEFT algorithm) to a processor is calculated locally without taking into account the entire graph. These methods try to make the best choice to minimize the completion time of some tasks, without taking into account the whole graph, which may not arrange the scheduling of the remaining tasks.

The pre-allocation technique decides in advance on which type of processors to execute each task. Then, the scheduling algorithm decides on which processor among the processors of the type chosen by the pre-allocation to execute each task. Forcing a task to run on one of the two types may not locally arrange the duration of scheduling, but may arrange the scheduling of the remaining tasks, and thus the final scheduling of all tasks.

Using the pre-allocation technique, a two-phase approach has been first developed by Kedad-Sidhoum et al. [77] for the problem without communications delays. The first phase consists in solving an assignment problem to find the type of processor assigned to execute each task ($\mathcal{A}$ or $\mathcal{B}$) using a linear model. In the second phase, a list scheduling algorithm has been proposed to generate a feasible schedule. This algorithm achieves an approximation ratio of 6. The following algorithms use the same technique to solve the problem with communication costs by adding new constraints. Two algorithms are proposed: a non polynomial-time algorithm (LSWPA) with a performance ratio of 6 and a polynomial time algorithm (PLSWPA) with a relative performance guarantee.

4.4.1 List Scheduling algorithm With Pre-Allocation (LSWPA)

LSWPA algorithm uses two steps to keep the same ratio of 6 for the scheduling with communication costs. The first step consists in solving an assignment problem to find which type of processor ($\mathcal{A}$ or $\mathcal{B}$) will execute each task. Two models ($P1$) and ($P2$) are then used in the first phase for solving the assignment problem while the precedence constraints are satisfied. The solution obtained by the model ($P1$) or ($P2$) represents a lower bound for the final makespan. Then, after solving the relaxation ($P1^{'}$) (resp. ($P2^{'}$)) of the model ($P1$) (resp. ($P2$)), the fractional solution is rounded to obtain a feasible assignment of the tasks. In the second phase, a list scheduling algorithm is used to provide a feasible schedule. In the following, the two phases are presented in detail.

I) Phase 1: Assignment of tasks (non-polynomial mapping θ_1)

To solve the assignment of tasks problem, we first propose the following Non-Linear Mixed Integer Program (NLMIP) using the following data and decision variables for $i = \overline{1..n}$ and $j = \overline{1..n}$.

Data:

For this model, we use the two following data:

- $w_{i,\mathcal{A}}$ (resp. $w_{i,\mathcal{B}}$) is the execution time of t_i on a processor of type $\mathcal{A}$ (resp. $\mathcal{B}$).
- $cm_{i,j}$ represents the communication delay between t_i and t_j if they are executed on two different types of processing elements.

Variables:

The decision variables are:

- $x_i = \begin{cases} 1 & \text{if the task } t_i \text{ is assigned to a processor type } \mathcal{A} \\ 0 & \text{if the task } t_i \text{ is assigned to a processor type } \mathcal{B}. \end{cases}$
- C_i represents the completion time of the task t_i.
- $z_{i,j} = \begin{cases} 1 & \text{if } t_i \text{ and } t_j \text{ are assigned to a processor type } \mathcal{A} \\ 0 & \text{if } t_i \text{ or } t_j, \text{ or both are assigned to a processor type } \mathcal{B}. \end{cases}$
- $y_{i,j} = \begin{cases} 1 & \text{if } t_i \text{ and } t_j \text{ are assigned to a processor type } \mathcal{B} \\ 0 & \text{if } t_i \text{ or } t_j, \text{ or both are assigned to a processor type } \mathcal{A}. \end{cases}$

Model ($P1$):

Based on the model given in [77], the following model is proposed:

$$
(P1)\begin{cases}
C_i + x_j w_{j,\mathcal{A}} + (1-x_j) w_{j,\mathcal{B}} + (1 - |y_{i,j} - z_{i,j}|) cm_{i,j} \leqslant C_j, \ \forall (t_i, t_j) \in E & (1)\\
z_{i,j} \leqslant x_i, \forall (t_i, t_j) \in E & (2)\\
z_{i,j} \leqslant x_j, \forall (t_i, t_j) \in E & (3)\\
y_{i,j} \leqslant 1 - x_i, \forall (t_i, t_j) \in E & (4)\\
y_{i,j} \leqslant 1 - x_j, \forall (t_i, t_j) \in E & (5)\\
x_i w_{i,\mathcal{A}} + (1-x_i) w_{i,\mathcal{B}} \leqslant C_i, \forall i = \overline{1..n}, \Gamma^-(t_i) = \emptyset & (6)\\
0 \leqslant C_i \leqslant C_{max}, \forall i = \overline{1..n}, \Gamma^+(t_i) = \emptyset & (7)\\
\sum_{i=1}^{n} x_i w_{i,\mathcal{A}} \leqslant \ell C_{max} & (8)\\
\sum_{i=1}^{n} (1-x_i) w_{i,\mathcal{B}} \leqslant k C_{max} & (9)\\
x_i, y_{i,j}, z_{i,j} \in \{0,1\}, \ \forall i = \overline{1..n}, j = \overline{1..n} & \\
Z(min) = C_{max} &
\end{cases}
$$

Constraints:

- Constraints (1) to (7) describe the constraints of the critical path of the graph $G(V, E)$. If $(t_i, t_j) \in E$, then:
  - If t_i and t_j are assigned to two different types of processors, we obtain two cases: either $x_i = 1$ and $x_j = 0$ or $x_i = 0$ and $x_j = 1$. In the two cases, we obtain $y_{i,j} = 0$ and $z_{i,j} = 0$ because of the four constraints (2), (3), (4) and (5). This implies that $1 - |y_{i,j} - z_{i,j}| = 1$.
  - If t_i and t_j are assigned to the same type of processors, we obtain also two cases:

⋆ Case 1: $x_i = 0$ and $x_j = 0$. From constraints (2) and (3), $z_{i,j} = 0$. From constraints (4) and (5), $y_{i,j} = 1$ (minimization problem), then $1 - |y_{i,j} - z_{i,j}| = 0$.

⋆ Case 2: $x_i = 1$ and $x_j = 1$. From constraints (4) and (5), $y_{i,j} = 0$. From constraints (2) and (3), $z_{i,j} = 1$ (minimization problem), then $1 - |y_{i,j} - z_{i,j}| = 0$.

- Tasks without predecessors (resp. successors) are considered in the constraint (6) (resp. (7)).

- Constraint (8) (resp. (9)) simply expresses that the makespan cannot be smaller than the average execution time of the tasks assigned to all processors of type $\mathcal{A}$ (resp. $\mathcal{B}$).

Linear model ($P2$):

The first constraint of model ($P1$) contains an absolute value that can be processed in the *CPLEX* APIs [69], by using *Cplex.abs* option. To see the efficiency of *CPLEX* in managing the absolute value, we have proposed another model ($P2$) without constraint containing an absolute value.

By adding two binary variables $a_{i,j}$ and $b_{i,j}$ and two constraints (5.1) and (5.2), we can get rid of the absolute value and we obtain a second model ($P2$).

$$(P2)\begin{cases} C_i + x_j w_{j,\mathcal{A}} + (1-x_j) w_{j,\mathcal{B}} + \mathbf{(1-b_{i,j})} cm_{i,j} \leqslant C_j,\ \forall (t_i,t_j) \in E & (1) \\ z_{i,j} \leqslant x_i, \forall (t_i,t_j) \in E & (2) \\ z_{i,j} \leqslant x_j, \forall (t_i,t_j) \in E & (3) \\ y_{i,j} \leqslant 1 - x_i, \forall (t_i,t_j) \in E & (4) \\ y_{i,j} \leqslant 1 - x_j, \forall (t_i,t_j) \in E & (5) \\ \mathbf{(z_{i,j} - y_{i,j}) + 2(1-a_{i,j}) \geqslant b_{i,j}, \forall (t_i,t_j) \in E} & \mathbf{(5.1)} \\ \mathbf{(y_{i,j} - z_{i,j}) + 2a_{i,j} \geqslant b_{i,j}, \forall (t_i,t_j) \in E} & \mathbf{(5.2)} \\ x_i w_{i,\mathcal{A}} + (1-x_i) w_{i,\mathcal{B}} \leqslant C_i, \forall i = \overline{1..n}, \Gamma^-(t_i) = \emptyset & (6) \\ 0 \leqslant C_i \leqslant C_{max}, \forall i = \overline{1..n}, \Gamma^+(t_i) = \emptyset & (7) \\ \sum_{i=1}^{n} x_i w_{i,\mathcal{A}} \leqslant \ell C_{max} & (8) \\ \sum_{i=1}^{n} (1-x_i) w_{i,\mathcal{B}} \leqslant k C_{max} & (9) \\ x_i, y_{i,j}, z_{i,j}, a_{i,j}, b_{i,j} \in \{0,1\},\ \forall i = \overline{1..n}, j = \overline{1..n} & \\ Z(min) = C_{max} & \end{cases}$$

For each couple of tasks $(t_i, t_j) \in E$, the variable $b_{i,j}$ is added to manage the communication delay between them. $b_{i,j} = 1$ if t_i and t_j are assigned to the same type of processing elements, 0 otherwise. The variable $a_{i,j}$ is added to validate one of the two constraints (5.1) of (5.2) such that:

1. If $a_{i,j} = 1$, then the two constraints (5.1) and (5.2) become $\begin{cases} (z_{i,j} - y_{i,j}) \geqslant b_{i,j} & (5.1) \\ (y_{i,j} - z_{i,j}) + 2 \geqslant b_{i,j} & (5.2) \end{cases}$

   In this case, the constraint (5.2) is not important, since $(y_{i,j} - z_{i,j}) \geqslant -1$ and then $1 \geqslant b_{i,j}$. Thus, in this case the value of $b_{i,j}$ depends on the constraint (5.1).

2. If $a_{i,j} = 0$, then the two constraints (5.1) and (5.2) become $\begin{cases} (z_{i,j} - y_{i,j}) + 2 \geqslant b_{i,j} & (5.1) \\ (y_{i,j} - z_{i,j}) \geqslant b_{i,j} & (5.2) \end{cases}$

   In this case, the constraint (5.1) is not important, since $(z_{i,j} - y_{i,j}) \geqslant -1$ and then $1 \geqslant b_{i,j}$. Thus, in this case the value of $b_{i,j}$ depends on the constraint (5.2).

The difference between ($P1$) and ($P2$):

The two models ($P1$) and ($P2$) are equivalent. Indeed, the variable $b_{i,j}$ replaces $|y_{i,j} - z_{i,j}| \in \{0,1\}$ in ($P2$). We obtain two cases:

- If $|y_{i,j} - z_{i,j}| = 0$ then $2a_{i,j} \geqslant b_{i,j}$ and $2(1 - a_{i,j}) \geqslant b_{i,j}$ from the two constraints (5.1) and (5.2). Thus, $b_{i,j} = 0$ for $a_{i,j} \in \{0, 1\}$. Finally, $|y_{i,j} - z_{i,j}| = b_{i,j} = 0$.
- If $|y_{i,j} - z_{i,j}| = 1$ and if $(y_{i,j} - z_{i,j}) = 1$, then $1 + 2a_{i,j} \geqslant b_{i,j}$ and $-1 + 2(1 - a_{i,j}) \geqslant b_{i,j}$ from the two constraints (5.1) and (5.2). Thus, since $b_{i,j} \in \{0, 1\}$, $b_{i,j} = 1$ for $a_{i,j} = 0$. If $(z_{i,j} - y_{i,j}) = 1$, then $-1 + 2a_{i,j} \geqslant b_{i,j}$ and $1 + 2(1 - a_{i,j}) \geqslant b_{i,j}$ from the two constraints (5.1) and (5.2). Then, since $b_{i,j} \in \{0, 1\}$, $b_{i,j} = 1$ for $a_{i,j} = 1$. In the two cases, we can have $|y_{i,j} - z_{i,j}| = b_{i,j}$.

Thus, both models $(P1)$ and $(P2)$ are equivalent and give the same result. In the following, we focus on the model $(P1)$. The results obtained for $(P1)$ remain valid for $(P2)$.

What represents the solution of $(P1)$ or $(P2)$ without constraints 8 and 9?

By removing the constraints (8) and (9) from the model $(P1)$, we obtain the model (PI) as follows.

$$(PI) \begin{cases} C_i + x_j w_{j,\mathcal{A}} + (1 - x_j) w_{j,\mathcal{B}} + (1 - |y_{i,j} - z_{i,j}|) cm_{i,j} \leqslant C_j, \ \forall (t_i, t_j) \in E & (1) \\ z_{i,j} \leqslant x_i, \forall (t_i, t_j) \in E & (2) \\ z_{i,j} \leqslant x_j, \forall (t_i, t_j) \in E & (3) \\ y_{i,j} \leqslant 1 - x_i, \forall (t_i, t_j) \in E & (4) \\ y_{i,j} \leqslant 1 - x_j, \forall (t_i, t_j) \in E & (5) \\ x_i w_{i,\mathcal{A}} + (1 - x_i) w_{i,\mathcal{B}} \leqslant C_i, \forall i = \overline{1..n}, \Gamma^-(t_i) = \emptyset & (6) \\ 0 \leqslant C_i \leqslant C_{max}, \forall i = \overline{1..n}, \Gamma^+(t_i) = \emptyset & (7) \\ x_i, y_{i,j}, z_{i,j} \in \{0, 1\}, \ \forall i = \overline{1..n}, j = \overline{1..n} & \\ Z(min) = C_{max} & \end{cases}$$

The optimal solution obtained by this model represents the solution of the scheduling problem on platforms with unlimited resources presented in the previous chapter (Chapter 3). In fact, the number of resources is unlimited, and there are no communication delays between processors of the same type. Thus, all tasks assigned to the same type of processing elements can all be performed on a different processor. The problem then comes to find the best scheduling that minimizes the completion execution time of the longest path without taking into account the workload constraints ((8) and (9)) on each type of processor, which is exactly the aim of this model.

Complexity:

We have previously shown by Theorem 3 that finding an optimal scheduling that minimizes makespan on a platform with unlimited resources is strongly NP-complete even for graphs of depth 3. Thus, we can say that the complexity of this model is at least strongly NP-complete.

Relaxed problem:

The problem of finding the optimal mapping using $(P1)$ or $(P2)$ is at least strongly NP-complete even without constraints (8) and (9). We can use solvers like *CPLEX* to solve $(P1)$ or $(P2)$ only for small instances. In order to have an easier problem, we relax the integer variables x_i, $y_{i,j}$ and $z_{i,j}$ and we obtain the model $(P1')$. We denote by x'_i, $y'_{i,j}$, $z'_{i,j}$, all in $[0, 1]$, the fractional value of x_i, $y_{i,j}$, $z_{i,j}$ in the optimal solution of the model $(P1')$, with $i = \overline{1..n}$ and $j = \overline{1..n}$.

We also obtain the model $(P2')$ by relaxing the integer variables x_i, $y_{i,j}$, $z_{i,j}$ and $b_{i,j}$. We denote by x'_i, $y'_{i,j}$, $z'_{i,j}$, $b'_{i,j}$ all in $[0, 1]$, the fractional value of x_i, $y_{i,j}$, $z_{i,j}$, $b_{i,j}$ in the optimal solution of the model $(P2')$, with $i = \overline{1..n}$ and $j = \overline{1..n}$. However, $a_{i,j}$ must remain integer, so that $(P1')$ and $(P2')$ are equivalent, i.e., $|y'_{i,j} - z'_{i,j}| = b'_{i,j}$. For $(t_i, t_j) \in E$, if we relax $a_{i,j} \in \{0, 1\}$

to $a^{'}_{i,j} \in [0,1]$, the solutions of $(P1^{'})$ and $(P2^{'})$ may be different, i.e., $|y^{'}_{i,j} - z^{'}_{i,j}| \neq b^{'}_{i,j}$. For example, we suppose that $x^{'}_{i} = 1$ and $x^{'}_{j} = 0$. Then, from the constraints (2), (3), (4) and (5) of $(P1^{'})$ or $(P2^{'})$, $y^{'}_{i,j} = 0$ and $z^{'}_{i,j} = 0$. Thus, the communication delay between t_i and t_j in the model $(P1^{'})$ will be calculated as $(1 - |y^{'}_{i,j} - z^{'}_{i,j}|)cm_{i,j} = (1-0)cm_{i,j} = cm_{i,j}$. However, from the constraint (5.1) and (5.2) of $(P2^{'})$, $2a^{'}_{i,j} \geqslant b^{'}_{i,j}$ and $2(1 - a^{'}_{i,j}) \geqslant b^{'}_{i,j}$. For $a^{'}_{i,j} = 0.5$, we will always have $1 \geqslant b^{'}_{i,j}$, and thus $(1 - b^{'}_{i,j})cm_{i,j} = 0$ since it is a minimization problem ($b^{'}_{i,j} = 1$). In other words, $a_{i,j}$ must remain integer, so that $b^{'}_{i,j}$ will be equal to $|y^{'}_{i,j} - z^{'}_{i,j}|$, i.e., $b^{'}_{i,j} = y^{'}_{i,j} - z^{'}_{i,j}$ or $b^{'}_{i,j} = z^{'}_{i,j} - y^{'}_{i,j}$.

Rounding strategy:

If $x^{'}_{i}$ is integer for $i \in \overline{1..n}$, the solution obtained is feasible and optimal for $(P1)$ and $(P2)$, otherwise the fractional values are rounded. We denote by x^{r}_{i} the rounded value of the fractional value $x^{'}_{i}$ in the optimal solution of $(P1^{'})$ or $(P2^{'})$. We set $x^{r}_{i} = 0$ if $x^{'}_{i} < \frac{1}{2}$, and $x^{r}_{i} = 1$ otherwise. Let θ_1 be the mapping obtained by this rounding. Each task t_i is mapped in either a processor type $\mathcal{A}$ or type $\mathcal{B}$. Thus, $\theta_1(t_i) \longrightarrow \{\mathcal{A}, \mathcal{B}\}$.

II) Phase 2: Scheduling algorithm

The second phase consists in finding the final scheduling of the tasks. Using the mapping θ_1, the proposed algorithm determines for a task order given by a priority list L, the corresponding scheduling by executing the first ready task of the list as long as there are free processing elements.

The priority list L can be defined by different way, where $n!$ lists are possible for an instance of n tasks. To achieve good scheduling, the most important and influential tasks must be executed first. For this purpose, we propose in the following several priority lists that will be used for the experimentations.

List by using the model $(P1^{'})$ or $(P2^{'})$

Two interesting lists can be extracted from the resolution of model $(P1^{'})$ or $(P2^{'})$, LST (List by Start Time) and LFT (List by Finish Time). The LST (resp. LFT) list can be obtained by sorting the tasks in ascending order of their processing start time (resp. processing finish time) obtained by solving the model $(P1^{'})$ or $(P2^{'})$. Let $S^{'}_{i}$ (resp. $C^{'}_{i}$) be the processing start time (resp. processing finish time) of the task t_i obtained by solving the model $(P1^{'})$ or $(P2^{'})$, $i = \overline{1..n}$. Thus, $LST = (t_{i1}, t_{i2}, \cdots, t_{in})$, with $S^{'}_{i1} \leqslant S^{'}_{i2} \leqslant \cdots \leqslant S^{'}_{in}$. $LFT = (t_{i1}, t_{i2}, \cdots, t_{in})$, with $C^{'}_{i1} \leqslant C^{'}_{i2} \leqslant \cdots \leqslant C^{'}_{in}$.

List by longest path (LLP):

First, we start by defining the graph $G'(V, E)$, with $V = \{t_1, t_2, \cdots, t_n\}$ and E represents the set of graph edges. The vertices are labelled by the execution times of the tasks according to their assignments. The edges are labelled by the communication costs for each $(t_i, t_j) \in E$ and $\theta_1(t_i) \neq \theta_1(t_j)$, 0 otherwise. Then, we can calculate the longest path LP_i from each task t_i to its last successor. The list LLP is given by $LLP = (t_{i1}, t_{i2}, \cdots, t_{in})$, such that $LP_{i1} \geqslant LP_{i2} \geqslant \cdots \geqslant LP_{in}$.

The following list algorithm is then applied using the mapping θ_1 and one of the three lists LST, LFT or LLP.

List Scheduling algorithm With Pre-Allocation (LSWPA) algorithm:

The following algorithm 7 executes task by task from the ready task list according to the order given by list L. Thus, if two tasks are executable at a given time on a free processor, then we

chose the first tasks according to the order given by the list L. Furthermore, each task t_i is executed on one of the processors of type $\theta_1(t_i)$ using an insertion-based scheduling policy [26]. It tries to insert a task at the earliest idle time between two already scheduled tasks on the processing element, if the slot is large enough to execute the task.

Algorithm 7: LSWPA algorithm

Data: mapping θ_1, list L (LST, LFT, LLP).
Result: Feasible scheduling.
begin
  Create an empty list ready-list;
  ready-list= $\{t_j,\ \Gamma^-(t_j) = \emptyset,\ j = \overline{1..n}\}$;
  while *ready-list* $\neq \emptyset$ **do**
    $t_i \longleftarrow$ the first executable task from the ready-list according to the order given by list L;
    for *all* $p_r \in P(\theta_1(t_i))$ **do**
      Compute $EFT(t_i, p_r)$ value using an insertion-based scheduling policy;
    Assign task t_i to the processor p_r that minimize EFT of task t_i;
    Update ready-list with unscheduled tasks;

Complexity:

The first phase consists in solving either a NLMIP model ($P1^{'}$) or a MIP model ($P2^{'}$), which makes the complexity of the first phase exponential, its resolution therefore depends on the size of the instance to be solved.

The second phase has the same structure as HEFT which has $\mathcal{O}(n^2m)$ time complexity. However, for LSWPA algorithm, the EFT of each task is calculated for either ℓ or k processors according to the mapping θ_1. For each task (n in total), we calculate its EFT by taking into account all its predecessors ($n-1$ at most) on each available processing element of type $\theta_1(t_i)$ (at most max(ℓ,k)). The insertion policy is verified on a processing element by checking the non-overlapping with at most $n-1$ tasks. This makes a complexity of $\mathcal{O}(max(\ell,k)n^2)$ for the second phase. Thus, the complexity time of LSWPA algorithm is exponential even if the second phase is polynomial.

Algorithm analysis:

In Section 4.5, we perform experiments to see the efficiency of LSWPA algorithm in terms of running time and makespan compared to HEFT. To study the theoretical worst-case performance of our method, we prove in this section that LSWPA algorithm solves the scheduling problem with a performance guarantee of 6 in the worst case compared to the optimal solution.

Proposition 3. *The rounding previously defined satisfies the following two inequalities:*

$$x_i^r \leqslant 2x_i^{'}$$

$$(1 - x_i^r) \leqslant 2(1 - x_i^{'}).$$

Proof. If $0 \leqslant x_i^{'} < \frac{1}{2}$, then $x_i^r = 0 \leqslant 2x_i^{'}$. Furthermore, $2x_i^{'} \leqslant 1$, then $0 \leqslant 1 - 2x_i^{'}$, follows $-x_i^r = 0 \leqslant 1 - 2x_i^{'}$, then $1 - x_i^r \leqslant 2(1 - x_i^{'})$. If $\frac{1}{2} \leqslant x_i^{'}$ then $1 \leqslant 2x_i^{'}$, follows $x_i^r = 1 \leqslant 2x_i^{'}$. Furthermore, $x_i^{'} \leqslant 1$ then $-2x_i^{'} \geqslant -2$, follows $1 - 2x_i^{'} \geqslant -1$, then $-x_i^r = -1 \leqslant 1 - 2x_i^{'}$, then $1 - x_i^r \leqslant 2(1 - x_i^{'})$. □

Let $\zeta_{i,j}$ be the value defined as $\zeta_{i,j} = (1 - |y_{i,j}^{'} - z_{i,j}^{'}|)$ for each couple of tasks $(t_i, t_j) \in E$. The communication cost between $(t_i, t_j) \in E$ obtained by solving ($P1^{'}$) is then given by $\zeta_{i,j} cm_{i,j} =$

$(1 - |y'_{i,j} - z'_{i,j}|)cm_{i,j}$. Thus, to minimize the communication cost between each couple of tasks $(t_i, t_j) \in E$, the value of $\zeta_{i,j}$ must be minimized. In the following lemma 9, we show that by solving the model $(P1')$, $\zeta_{i,j} = (1 - |y'_{i,j} - z'_{i,j}|)$ can take the smallest possible value for each $(t_i, t_j) \in E$, such that $\zeta_{i,j} = (1 - |y'_{i,j} - z'_{i,j}|) = (1 - max(y'_{i,j}, z'_{i,j}))$.

Lemma 9. *Let C'_{max} be the optimal solution obtained by solving the model $(P1')$. We can get another solution $C''_{max} = C'_{max}$, such that for each two $(t_i, t_j) \in E$:*

1. *If $min\{1 - x'_i, 1 - x'_j\} \geqslant min\{x'_i, x'_j\}$, then $y'_{i,j} = min\{1 - x'_i, 1 - x'_j\}$ and $z'_{i,j} = 0$.*
2. *If $min\{1 - x'_i, 1 - x'_j\} < min\{x'_i, x'_j\}$, then $y'_{i,j} = 0$ and $z'_{i,j} = min\{x'_i, x'_j\}$.*

Proof. From constraints (2) and (3) of model $(P1')$, $z'_{i,j} \leqslant \min\{x'_i, x'_j\}$. From constraints (4) and (5) of model $(P1')$, $y'_{i,j} \leqslant \min\{1 - x'_i, 1 - x'_j\}$. Let β be the value given by $\beta = |y'_{i,j} - z'_{i,j}|$. To minimize the communication cost $(\zeta_{i,j}cm_{i,j} = (1 - |y'_{i,j} - z'_{i,j}|)cm_{i,j})$ between t_i and t_j, we have to maximize β. Thus, $\beta = |y'_{i,j} - z'_{i,j}| \leqslant max\{y'_{i,j}, z'_{i,j}\} \leqslant max\{\min\{1 - x'_i, 1 - x'_j\}, \min\{x'_i, x'_j\}\}$. If $\min\{1 - x'_i, 1 - x'_j\} \geqslant \min\{x'_i, x'_j\}$, then we can put $y'_{i,j} = \min\{1 - x'_i, 1 - x'_j\}$ and $z'_{i,j} = 0$ to maximize β. Otherwise, we can put $z'_{i,j} = \min\{x'_i, x'_j\}$ and $y'_{i,j} = 0$. □

In the following, we suppose that the solution obtained by solving the model $(P1')$ follows the properties given by the Lemma 9. i,e., $\zeta_{i,j}$ always take the smallest possible value for each $(t_i, t_j) \in E$. In the following, we look for the relation between the value of $\zeta_{i,j}$ and the assignment of the tasks t_i and t_j.

Remark 4. The value $\zeta_{i,j}$ is always positive for each couple of tasks $(t_i, t_j) \in E$, i.e., $\zeta_{i,j} \geqslant 0$. Indeed, if $y'_{i,j} = \min\{1 - x'_i, 1 - x'_j\} \leqslant 1$ and $z'_{i,j} = 0$, then $\zeta_{i,j} = 1 - y'_{i,j} \geqslant 0$. Furthermore, if $z'_{i,j} = \min\{x'_i, x'_j\} \leqslant 1$ and $y'_{i,j} = 0$, then $\zeta_{i,j} = 1 - z'_{i,j} \geqslant 0$.

Lemma 10. *If t_i and t_j are allocated to two different processing elements ($x^r_i = 0$ and $x^r_j = 1$, or $x^r_i = 1$ and $x^r_j = 0$), then $\zeta_{i,j} \geqslant \frac{1}{2}$.*

Proof. Let t_i and t_j be two tasks assigned to two different processing elements $(x^r_i \neq x^r_j)$. We obtain two cases:

a. If $x^r_i = 0$ and $x^r_j = 1$, with $x'_i < \frac{1}{2}$ and $x'_j \geqslant \frac{1}{2}$, then from the constraints (2) and (3), $z'_{i,j} < \frac{1}{2}$. Furthermore, $1 - x'_i > \frac{1}{2}$ and $1 - x'_j \leqslant \frac{1}{2}$, then, from the constraints (4) and (5), $y'_{i,j} \leqslant \frac{1}{2}$. Finally, $\zeta_{i,j} = 1 - |y'_{i,j} - z'_{i,j}| \geqslant 1 - \max\{y_{i,j}, z_{i,j}\} \geqslant \frac{1}{2}$.

b. If $x^r_i = 1$ and $x^r_j = 0$, with $x'_i \geqslant \frac{1}{2}$ and $x'_j < \frac{1}{2}$ Then from the constraints (2) and (3), $z'_{i,j} < \frac{1}{2}$. Furthermore, $1 - x'_i \leqslant \frac{1}{2}$ and $1 - x'_j > \frac{1}{2}$, then, from the constraints (4) and (5), $y'_{i,j} \leqslant \frac{1}{2}$. Finally, $\zeta_{i,j} = 1 - |y'_{i,j} - z'_{i,j}| \geqslant 1 - \max\{y_{i,j}, z_{i,j}\} \geqslant \frac{1}{2}$.

□

Let t_i and t_j be two tasks, such that $(t_i, t_j) \in E$. We denote by $Cost^r_{i,j}$ the value given by $Cost^r_{i,j} = 0$ if $x^r_i = x^r_j$, $Cost^r_{i,j} = cm_{i,j}$ otherwise.

Proposition 4. $Cost^r_{i,j} \leqslant 2\zeta_{i,j}cm_{i,j}$.

Proof. If t_j and t_j are executed by the same processing element, $Cost^r_{i,j} = 0 \leqslant 2\zeta_{i,j}cm_{i,j}$, because $\zeta_{i,j} \geqslant 0$. If t_j and t_j are executed by two different processing elements, then $Cost^r_{i,j} = cm_{i,j}$. Then, from Lemma 10, $\zeta_{i,j} \geqslant \frac{1}{2}$, then $2\zeta_{i,j} \geqslant 1$, follows $Cost^r_{i,j} = cm_{i,j} \leqslant 2\zeta_{i,j}cm_{i,j}$. □

With this result, we know that by applying the rounding θ_1 previously defined, the communication delay between each couple of tasks $(t_j, t_j) \in E$ in the final scheduling can be bounded. It will be at most twice the communication delay taken into account in the solution obtained by the model $(P1^{'})$ or $(P2^{'})$ between these two tasks. In the following, we use this result to prove the performance guarantee of our method.

Lower bound:

We denote by $C^{'}_{max}$ the optimal solution obtained by $(P1^{'})$ or $(P2^{'})$, this solution is a lower bound for the optimal solution $C^{\star}_{max}$ of our problem. $C^{'}_{max}$ is bounded by:

1. $L(P_f)$: the length of the fractional critical path P_f in the optimal solution of the model $(P1^{'})$ or $(P2^{'})$.
2. $\frac{W^f_{\mathcal{A}}}{\ell}$: the fractional execution times of the tasks allocated to the processing elements of type $\mathcal{A}$ in the optimal solution of the model $(P1^{'})$ or $(P2^{'})$ divided by ℓ, with $W^f_{\mathcal{A}} = \sum_{i=1}^{n} x^{'}_i w_{i,\mathcal{A}}$.
3. $\frac{W^f_{\mathcal{B}}}{k}$: the fractional execution times of the tasks allocated to the processing elements of type $\mathcal{B}$ in the solution of the optimal model $(P1^{'})$ or $(P2^{'})$ divided by k, with $W^f_{\mathcal{B}} = \sum_{i=1}^{n}(1 - x^{'}_i) w_{i,\mathcal{B}}$.

Furthermore, the solution $\widehat{C}_{max}$ obtained by LSWPA algorithm 7 is bounded by $L(P_r)$, $\frac{W^r_{\mathcal{A}}}{\ell}$, $\frac{W^r_{\mathcal{B}}}{k}$, the rounded values of $L(P_f)$, $\frac{W^f_{\mathcal{A}}}{\ell}$ and $\frac{W^f_{\mathcal{B}}}{k}$, where $\frac{W^r_{\mathcal{A}}}{\ell} = \frac{\sum_{i=1}^{n} x^r_i w_{i,\mathcal{A}}}{\ell}$ and $\frac{W^r_{\mathcal{B}}}{k} = \frac{\sum_{i=1}^{n}(1-x^r_i) w_{i,\mathcal{B}}}{k}$.

Worst case approximation ratio:

We denote by A (resp. I) the cumulative sum of periods of activity (resp. inactivity) in the solution obtained by LSPWA algorithm, where the processors are busy (resp. idle). Let $A_1 = \sum_{i=1}^{n} x^r_i w_{i,\mathcal{A}}$ (resp. $A_2 = \sum_{i=1}^{n}(1 - x^r_i) w_{i,\mathcal{B}}$) be the cumulative sum of periods of activity of all the processing elements of type $\mathcal{A}$ (resp, $\mathcal{B}$), $A = A_1 + A_2$. Let I_1 (resp. I_2) be the cumulative sum of periods of inactivity where all the processing elements of type $\mathcal{A}$ (resp. $\mathcal{B}$) are busy and at least one processor of type $\mathcal{B}$ (resp. $\mathcal{A}$) is idle. The maximum duration where all the processing elements of type $\mathcal{A}$ (resp. $\mathcal{B}$) are busy is $\frac{A_1}{\ell}$ (resp. $\frac{A_2}{k}$), then $I_1 \leqslant k\frac{A_1}{\ell}$ (resp. $I_2 \leqslant \ell\frac{A_2}{k}$).

Let I_3 be the cumulative sum of periods of inactivity where at least one processing element of type $\mathcal{A}$ and one processing element of type $\mathcal{B}$ are idle, $I = I_1 + I_2 + I_3$. Figure 4.1 represents the occupation of processing elements during the execution of an application.

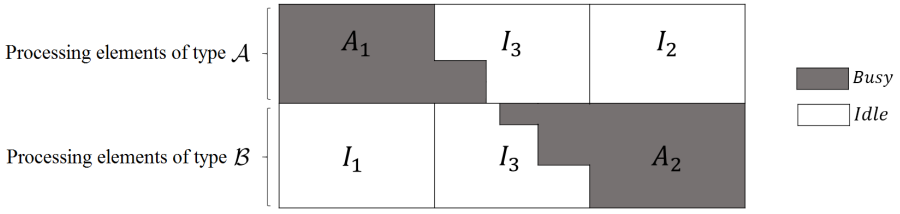


Figure 4.1: *Occupation of processing elements.*

We multiply $\widehat{C}_{max}$ by the number of processors, we find the cumulative sum of the periods of activity and inactivity, $(\ell + k)\widehat{C}_{max} = A + I$.

We look now for the ratio between $\widehat{C}_{max}$ and $C^{\star}_{max}$. For this purpose, we try to bound A and I with formulas in functions of $C^{\star}_{max}$.

Proposition 5. *$A_1 \leqslant 2\ell C^{\star}_{max}$ and $A_2 \leqslant 2kC^{\star}_{max}$.*

Proof. By definition, $A_1 = \sum_{i=1}^{n} x_i^r w_{i,\mathcal{A}}$. From Proposition 3, $x_i^r \leqslant 2x_i^{'}$. Then, $A_1 = \sum_{i=1}^{n} x_i^r w_{i,\mathcal{A}} \leqslant \sum_{i=1}^{n} 2x_i^{'} w_{i,\mathcal{A}} = 2W_{\mathcal{A}}^{f} \leqslant 2\ell C_{max}^{'} \leqslant 2\ell C^{\star}_{max}$. Furthermore, by definition, $A_2 = \sum_{i=1}^{n}(1-x_i^r)w_{i,\mathcal{B}}$. From Proposition 3, $(1-x_i^r) \leqslant 2(1-x_i^{'})$. Then, $A_2 = \sum_{i=1}^{n}(1-x_i^r)w_{i,\mathcal{B}} \leqslant \sum_{i=1}^{n} 2(1-x_i^{'})w_{i,\mathcal{B}} = 2W_{\mathcal{B}}^{f} \leqslant 2kC_{max}^{'} \leqslant 2kC^{\star}_{max}$. □

Corollary 4. *If for each task t_i, x_i^r is an integer, such that $x_i^r = x_i^{'}$ and $(1-x_i^r) = (1-x_i^{'})$ for $i = \overline{1..n}$, then the mapping θ_1 is optimal. Follow, $A_1 = \sum_{i=1}^{n} x_i^r w_{i,\mathcal{A}} = \sum_{i=1}^{n} x_i^{'} w_{i,\mathcal{A}} \leqslant \ell C^{\star}_{max}$ and $A_2 = \sum_{i=1}^{n}(1-x_i^r)w_{i,\mathcal{B}} = \sum_{i=1}^{n}(1-x_i^{'})w_{i,\mathcal{B}} \leqslant kC^{\star}_{max}$.*

Corollary 5. *$I_1 \leqslant 2kC^{\star}_{max}$ and $I_2 \leqslant 2\ell C^{\star}_{max}$.*

Proof. $I_1 \leqslant k\frac{A_1}{\ell} \leqslant k\frac{2\ell C^{\star}_{max}}{\ell} = 2kC^{\star}_{max}$. Furthermore, $I_2 \leqslant \frac{\ell A_2}{k} \leqslant \frac{2k\ell C^{\star}_{max}}{k} = 2\ell C^{\star}_{max}$. □

Proposition 6. *$I_3 \leqslant 2(\ell + k)C^{\star}_{max}$.*

Proof. There exists a critical path γ in the final scheduling such that the sum of the instants where at least one processing element of type $\mathcal{A}$ and one of type $\mathcal{B}$ are idle is less than $2L(P_f)$. Indeed, we assume that the tasks are stalled on the left.

Let t_0 be the last task, such as during the execution of t_0, there is an idle processing element of type $\mathcal{A}$ and an idle processing element of type $\mathcal{B}$. Let S_0 be the processing start time of the task t_0.

If there is an idle processing element of type $\mathcal{A}$ and idle processing element of type $\mathcal{B}$ before S_0, then t_0 has a predecessor t_1 that ends before S_0, the idle slots between S_1 and S_0 are covered either by the execution time of the task t_1 and eventually the communication cost between t_1 and his successor $t_0^{'}$ which can be t_0 or a task on the path from t_1 to t_0.

If there is an idle processing element of type $\mathcal{A}$ and idle processing element of type $\mathcal{B}$ before S_1, then t_1 has a predecessor t_2 that ends before S_1 which can be obtained in the same precedent way. Let t_0, $t_0^{'}$, t_1, $t_1^{'}$,$\cdots$, t_ℓ be the maximum sequence of tasks obtained. There are no more slots before S_ℓ where at least one processing element of type $\mathcal{A}$ and one processing element of type $\mathcal{B}$ are idle. Let γ be the path containing all these tasks which covers all periods when at least one processing element of type $\mathcal{A}$ and one processing element of type $\mathcal{B}$ are idle, let $L(\gamma)$ be its length. From Proposition 3, for any task t_i in P_r, the processing time of t_i in the final scheduling will be at most twice the fractional solution obtained by the model $(P1^{'})$ or $(P2^{'})$. For any two tasks t_i, t_j in P_r, from the Proposition 4, the communication cost $Cost_{i,j}^{r}$ between t_i and t_j in the final scheduling will increase by at most twice the fractional communication cost $\zeta_{i,j}cm_{i,j}$ obtained by the model $(P1^{'})$ or $(P2^{'})$.

$$\begin{aligned}
\text{Then, } L(\gamma) \leqslant L(P_r) &= \sum_{t_i \in P_r} (x_i^r w_{i,\mathcal{A}} + (1-x_i^r)w_{i,\mathcal{B}}) + \sum_{(t_i,t_j)\in P_r} Cost_{i,j}^{r} \\
&\leqslant \sum_{t_i \in P_r} (2x_i^{'} w_{i,\mathcal{A}} + 2(1-x_i^{'})w_{i,\mathcal{B}}) + \sum_{(t_i,t_j)\in P_r} 2\zeta_{i,j}cm_{i,j} \\
&\leqslant 2L(P_f)
\end{aligned}$$

$$\text{Finally, } I_3 \leqslant (\ell + k)L(P_r) \leqslant 2(\ell + k)L(P_f) \leqslant 2(\ell + k)C^{\star}_{max}$$

□

Theorem 7. *The ratio between the solution $\widehat{C}_{max}$ obtained by LSWPA algorithm and the optimal scheduling solution $C^{\star}_{max}$ is given by $\frac{\widehat{C}_{max}}{C^{\star}_{max}} \leqslant 6$.*

Proof. $\widehat{C}_{max} = \frac{A+I}{\ell+k} = \frac{A_1+A_2+I_1+I_2+I_3}{\ell+k}$. Then, from Proposition 5, 6 and Corollary 5, $\widehat{C}_{max} \leqslant \frac{(2\ell+2k+2k+2\ell+2(\ell+k))C^{\star}_{max}}{\ell+k} = 6C^{\star}_{max}$. Finally, $\frac{\widehat{C}_{max}}{C^{\star}_{max}} \leqslant 6$. □

Corollary 6. *If the mapping θ_1 is optimal, then $\frac{\widehat{C}_{max}}{C^{\star}_{max}} \leqslant 5$.*

Proof. From Corollary 4, we know that $A_1 \leqslant \ell C^{\star}_{max}$ and $A_2 \leqslant kC^{\star}_{max}$. Then, $\widehat{C}_{max} = \frac{A_1+A_2+I_1+I_2+I_3}{\ell+k} \leqslant \frac{(\ell+k+2k+2\ell+2(\ell+k))C^{\star}_{max}}{\ell+k}$. Finally, $\frac{\widehat{C}_{max}}{C^{\star}_{max}} \leqslant 5$. □

Remark 5. We can obtain the optimal solution of the assignment problem by solving the model $(P3)$.

$$(P3)\begin{cases} C_i + x_j w_{j,\mathcal{A}} + (1-x_j)w_{j,\mathcal{B}} + \zeta_{i,j} cm_{i,j} \leqslant C_j,\ \forall (t_i,t_j) \in E & (1a)\\ x_i - x_j \leqslant \zeta_{i,j}, \forall (t_i,t_j) \in E & (2a)\\ x_j - x_i \leqslant \zeta_{i,j}, \forall (t_i,t_j) \in E & (3a)\\ x_i w_{i,\mathcal{A}} + (1-x_i)w_{i,\mathcal{B}} \leqslant C_i, \forall i = \overline{1..n}, \Gamma^-(t_i) = \emptyset & (4a)\\ 0 \leqslant C_i \leqslant C_{max}, \forall i = \overline{1..n}, \Gamma^+(t_i) = \emptyset & (5a)\\ \sum_{i=1}^{n} x_i w_{i,\mathcal{A}} \leqslant \ell C_{max} & (6a)\\ \sum_{i=1}^{n} (1-x_i) w_{i,\mathcal{B}} \leqslant k C_{max} & (7a)\\ x_i \in \{0,1\},\ \zeta_{i,j} \in [0,1],\ \forall i = \overline{1..n}, j = \overline{1..n} & (8a)\\ Z(min) = C_{max} \end{cases}$$

This model is equivalent to models $(P1)$ and $(P2)$, but its relaxation is different from models $(P1')$ and $(P2')$. Indeed, the precedence constraints are represented in the models $(P1)$ and $(P3)$ for each couple of tasks $(t_i,t_j) \in E$ as follows:

$$(\widehat{P1})\begin{cases} C_i + x_j w_{j,\mathcal{A}} + (1-x_j)w_{j,\mathcal{B}} + (1-|y_{i,j}-z_{i,j}|)cm_{i,j} \leqslant C_j & (1)\\ z_{i,j} \leqslant x_i & (2)\\ z_{i,j} \leqslant x_j & (3)\\ y_{i,j} \leqslant 1 - x_i & (4)\\ y_{i,j} \leqslant 1 - x_j & (5) \end{cases}$$

$$(\widehat{P3})\begin{cases} C_i + x_j w_{j,\mathcal{A}} + (1-x_j)w_{j,\mathcal{B}} + \zeta_{i,j} cm_{i,j} \leqslant C_j & (1a)\\ x_i - x_j \leqslant \zeta_{i,j} & (2a)\\ x_j - x_i \leqslant \zeta_{i,j} & (3a) \end{cases}$$

By solving the models $(P1)$ and $(P3)$, we can obtain one of the four following possible cases for each couple of tasks $(t_i,t_j) \in E$:

1. $x_i = 1$ and $x_j = 1$: from constraints $(2a)$ and $(3a)$, we have $0 \leqslant \zeta_{i,j}$. From constraints (2), (3), (4) and (5), we have $z_{i,j} \leqslant 1$ and $y_{i,j} \leqslant 0$, and then $(1-|y_{i,j}-z_{i,j}|)$ can take 0 or 1.

2. $x_i = 0$ and $x_j = 0$: from constraints $(2a)$ and $(3a)$, we have $0 \leqslant \zeta_{i,j}$. From constraints (2), (3), (4) and (5), we have $z_{i,j} \leqslant 0$ and $y_{i,j} \leqslant 1$, and then $(1-|y_{i,j}-z_{i,j}|)$ can take 0 or 1.

3. $x_i = 1$ and $x_j = 0$: from constraints $(2a)$ and $(3a)$, we have $1 \leqslant \zeta_{i,j}$. From constraints (2), (3), (4) and (5), we have $z_{i,j} \leqslant 0$ and $y_{i,j} \leqslant 0$, and then $\zeta_{i,j} = (1-|y_{i,j}-z_{i,j}|) = 0$.

4. $x_i = 0$ and $x_j = 1$: from constraints $(2a)$ and $(3a)$, we have $1 \leqslant \zeta_{i,j}$. From constraints (2), (3), (4) and (5), we have $z_{i,j} \leqslant 0$ and $y_{i,j} \leqslant 0$, and then $\zeta_{i,j} = (1-|y_{i,j}-z_{i,j}|) = 0$.

In all cases, we can always have $\zeta_{i,j} = (1 - |y_{i,j} - z_{i,j}|)$. Then, since the other constraints are equivalent, models $(P1)$ and $(P3)$ are equivalent.

However, solving the relaxed version $(P3^{'})$ of this model $(x_i^{'} \in [0,1], \forall i = \overline{1..n})$ usually gives bad solutions after rounding the fractional values. Indeed, the communication delay for each couple of tasks $(t_i, t_j) \in E$ is given by $Cost_{i,j}^{'} = \zeta_{i,j} cm_{i,j} \geqslant |x_i^{'} - x_j^{'}| cm_{i,j}$. Assuming that $x_i^{'} = 0.4$ and $x_j^{'} = 0.6$, we obtain $Cost_{i,j}^{'} = \zeta_{i,j} cm_{i,j} \geqslant 0.2 cm_{i,j}$ and then $Cost_{i,j}^{'} = 0.2 cm_{i,j}$ (minimization problem). By rounding the values of $x_i^{'}$ and $x_j^{'}$, we obtain $x_i^r = 0$ and $x_j^r = 1$ with $Cost_{i,j}^r = cm_{i,j} = 5 Cost_{i,j}^{'}$. This means that we cannot deduce the performance guarantee of a method (worse case) which uses the solution obtained by $(P3^{'})$. Thus, rounding the solution obtained by $(P3^{'})$ does not guarantee the quality of LSWPA algorithm.

The complexity of the three models $(P1^{'})$, $(P2^{'})$ and $(P3)$, is considered to be exponential. An obvious question which could be asked is why we use the models $(P1^{'})$ and $(P2^{'})$ and not directly $(P3)$ which can give us the optimal assignment. To answer this question, we compare in the following the two linear models $(P3)$ and $(P2^{'})$.

Comparison between $(P3)$ and $(P2^{'})$:

$(P3)$ gives the optimal mapping contrary to $(P2^{'})$, but with different complexity. Both models contain binary variables and considered as MIP (Mixed Integer Programming) models. Thus, we compare the complexity between the two models as follows:

- Number of binary variables Bv_1 in $(P2^{'})$: the only binary variables are $a_{i,j}, \forall (t_i, t_j) \in E$. Then, $Bv_1 = |E|$.
- Number of binary variables Bv_2 in $(P3)$: in this model, $x_i \in \{0,1\}\ \forall i = \overline{1..n}$. Then, $Bv_2 = n$.
- Number of constraints which contain binary variables Cbv_1 in $(P2^{'})$: we have binary variables only on constraints (5.1) and (5.2). Then, $Cbv_1 = 2|E|$.
- Number of constraints which contain binary variables Cbv_2 in $(P3)$: constraints which contain binary variables are (1), (2), (3), (4), (6), (7). Then, $Cbv_2 \geqslant |E| + |E| + |E| + 1 + 1 + n + n = 3|E| + 2n + 2 > Cbv_1$.

In connected graphs, usually we have $|E| \geqslant n$, then $Bv_1 \geqslant Bv_2$. This means that we have more binary variables in model $(P2^{'})$ than $(P3)$, but $(P3)$ contains more constraints with binary variables than $(P2^{'})$. Both models are considered as non polynomial-time programs. In Section 4.5.4, we compare the running time of the two models and the solutions obtained by rounding the fractional values obtained by each model on different instances with different sizes (number of tasks and number of processors).

Using the pre-allocation technique, LSWPA algorithm provides a solution for hybrid platforms with a performance guarantee of 6. However, it is considered as a non polynomial-time two-phase approach. Numerical evaluations illustrated in Section 4.5 demonstrate that this method achieves a close-to-optimal performance. This method was published in [107]. However, the running time of this method can be important for large instances. We focus in the following on finding a polynomial-time approach which is able to maintain an interesting performance with reasonable complexity.

4.4.2 Polynomial List Scheduling With Pre-Allocation (PLSWPA)

In order to solve large instances, we introduce a new technique of pre-allocation, where the first phase is modified to obtain a polynomial algorithm, while keeping the second phase of LSWPA

algorithm. For this purpose, a new model is proposed to find a polynomial mapping θ_2. In the following, we use the solution obtained by solving the relaxation $(P3^{'})$ of the model $(P3)$ defined previously to propose another model $(P4^{'})$. Then, by rounding its fractional solution, we get a feasible assignment of the tasks.

Some properties of the model $(P3^{'})$:

As presented previously, the model $(P3)$ is defined as follows:

$$(P3)\begin{cases} C_i + x_j w_{j,\mathcal{A}} + (1-x_j) w_{j,\mathcal{B}} + \zeta_{i,j} cm_{i,j} \leqslant C_j, \ \forall (t_i,t_j) \in E & (1)\\ x_i - x_j \leqslant \zeta_{i,j}, \ \forall (t_i,t_j) \in E & (2)\\ x_j - x_i \leqslant \zeta_{i,j}, \ \forall (t_i,t_j) \in E & (3)\\ x_i w_{i,\mathcal{A}} + (1-x_i) w_{i,\mathcal{B}} \leqslant C_i, \forall i = \overline{1..n}, \Gamma^-(t_i) = \emptyset & (4)\\ 0 \leqslant C_i \leqslant C_{max3}, \forall i = \overline{1..n}, \Gamma^+(t_i) = \emptyset & (5)\\ \sum_{i=1}^{n} x_i w_{i,\mathcal{A}} \leqslant \ell C_{max3} & (6)\\ \sum_{i=1}^{n} (1-x_i) w_{i,\mathcal{B}} \leqslant k C_{max3} & (7)\\ x_i \in \{0,1\}, \ \zeta_{i,j} \in [0,1], \ \forall i,j = \overline{1..n} & \\ Z(min) = C_{max3} & \end{cases}$$

The optimal solution C_{max3} of this model does not take into account non-overlapping constraints. Thus, it computes a lower bound for the problem, $C^{\star}_{max3} \leqslant C^{\star}_{max}$.

Lemma 11. *For each two successive tasks $(t_i,t_j) \in E$, constraints (2) and (3) can be written as $max(x_i,x_j) - min(x_i,x_j) \leqslant \zeta_{i,j}$. Furthermore, $max(x_i,x_j) - min(x_i,x_j) = (1 - min(x_i,x_j)) + (max(x_i,x_j) - 1) = max(1-x_i, 1-x_j) - min(1-x_i, 1-x_j)$. Thus, constraints (2) and (3) can also be written as $max(1-x_i,1-x_j) - min(1-x_i,1-x_j) \leqslant \zeta_{i,j}$.*

Remark 6. In each feasible solution of $(P3)$, for each couple of tasks $(t_i,t_j) \in E$, we have always $max(x_i,x_j) = 1$ or $max(1-x_i,1-x_j) = 1$ (or both), $\forall (x_i,x_j) \in \{0,1\} \times \{0,1\}$.

Corollary 7. *In the optimal solution of $(P3)$, for each couple of tasks $(t_i,t_j) \in E$, from Lemma 11 we have at least $max(x_i,x_j) = 1$ or $max(1-x_i,1-x_j) = 1$, such that:*

- *if $max(x_i,x_j) = 1$, then constraints (2) and (3) can be represented by:*

$$\widetilde{Con}^{1}_{i,j} : 1 - min(x_i,x_j) \leqslant \zeta_{i,j}$$

- *if $max(1-x_i,1-x_j) = 1$, then constraints (2) and (3) can be represented by:*

$$\widetilde{Con}^{2}_{i,j} : 1 - min(1-x_i,1-x_j) \leqslant \zeta_{i,j}$$

The optimal solution obtained by the model $(P3)$ without constraints (6) and (7) represents the optimal solution of the scheduling problem on platforms with unlimited resources as presented in the previous chapter (Chapter 3). This problem has been proven to be NP-complete.

In order to have an easier problem, we relax the integer variables x_i for $i = \overline{1..n}$ and we obtain the relaxed model $(P3^{'})$. We denote by $\widetilde{x}^{'}_i \in [0,1]$, the fractional value of x_i in the optimal solution of the model $(P3^{'})$.

Model $(P4^{'})$:

Based on Lemma 11 and using the optimal solution of $(P3^{'})$, we define another model $(P4^{'})$. The decision variables are $x^{'}_i$, and an intermediary variable $y^{'}_{i,j} \in [0,1]$, with $i = \overline{1..n}$ and $j = \overline{1..n}$. For all $(t_i,t_j) \in E$, we define the constraint $Con_{i,j}$ as follows:

- if $\min\{\widetilde{x}_i^{'}, \widetilde{x}_j^{'}\} > \min\{1-\widetilde{x}_i^{'}, 1-\widetilde{x}_j^{'}\}$, then $Con_{i,j} = \begin{cases} y_{i,j}^{'} \leqslant x_i^{'} & (1) \\ y_{i,j}^{'} \leqslant x_j^{'} & (2) \\ \zeta_{i,j}^{'} = (1 - y_{i,j}^{'}) & (3) \end{cases}$

  From $Con_{i,j}$, we have $y_{i,j}^{'} \leqslant min\{x_i^{'}, x_j^{'}\}$. Then, $\zeta_{i,j}^{'} = 1 - y_{i,j}^{'} \geqslant (1 - min\{x_i^{'}, x_j^{'}\})$, which is equivalent to constraint $\widetilde{Con}_{i,j}^{1}$. Since it is a minimization problem, we can set $\zeta_{i,j}^{'} = (1 - min\{x_i^{'}, x_j^{'}\})$.

- if $\min\{\widetilde{x}_i^{'}, \widetilde{x}_j^{'}\} \leqslant \min\{1-\widetilde{x}_i^{'}, 1-\widetilde{x}_j^{'}\}$ then $Con_{i,j} = \begin{cases} y_{i,j}^{'} \leqslant 1 - x_i^{'} & (1) \\ y_{i,j}^{'} \leqslant 1 - x_j^{'} & (2) \\ \zeta_{i,j}^{'} = (1 - y_{i,j}^{'}) & (3) \end{cases}$

  From $Con_{i,j}$, we have $y_{i,j}^{'} \leqslant min\{1 - x_i^{'}, 1 - x_j^{'}\}$. Then, $\zeta_{i,j}^{'} = 1 - y_{i,j}^{'} \geqslant (1 - min\{1 - x_i^{'}, 1 - x_j^{'}\})$, which is equivalent to constraint $\widetilde{Con}_{i,j}^{2}$. Since it is a minimization problem, we can set $\zeta_{i,j}^{'} = (1 - min\{1 - x_i^{'}, 1 - x_j^{'}\})$.

The model $(P4^{'})$ is then given by:

$$(P4^{'}) \begin{cases} C_i^{'} + x_j^{'} w_{j,\mathcal{A}} + (1 - x_j^{'}) w_{j,\mathcal{B}} + \zeta_{i,j}^{'} cm_{i,j} \leqslant C_j^{'}, \forall (t_i, t_j) \in E & (1) \\ Con_{i,j}, \forall (t_i, t_j) \in E & (2) \\ x_i^{'} w_{i,\mathcal{A}} + (1 - x_i^{'}) w_{i,\mathcal{B}} \leqslant C_i^{'}, \forall i = \overline{1..n}, \Gamma^{-}(t_i) = \emptyset & (3) \\ 0 \leqslant C_i^{'} \leqslant C_{max4'}, \forall i = \overline{1..n}, \Gamma^{+}(t_i) = \emptyset & (4) \\ \sum_{i=1}^{n} x_i^{'} w_{i,\mathcal{A}} \leqslant \ell C_{max4'} & (5) \\ \sum_{i=1}^{n} (1 - x_i^{'}) w_{i,\mathcal{B}} \leqslant k C_{max4'} & (6) \\ x_i^{'}, y_{i,j}^{'}, \zeta_{i,j}^{'} \in [0,1], \ \forall i = \overline{1..n}, j = \overline{1..n} & \\ Z(min) = C_{max4'} & \end{cases}$$

We can notice that constraints (1, 3, 4, 5, 6) of model of $(P4^{'})$ are equivalent to constraints (1, 4, 5, 6, 7) of model $(P3^{'})$.

Polynomial List Scheduling With Pre-Allocation (PLSWPA) algorithm:

To obtain a feasible mapping, the same rounding strategy used to obtain θ_1 is used to round the solution obtained by the model $(P4^{'})$, where each task t_i is mapped in either a processor type $\mathcal{A}$ or type $\mathcal{B}$. We set $x_i^r = 0$ if $x_i^{'} < \frac{1}{2}$, and $x_i^r = 1$ otherwise. We denoted by θ_2 the mapping obtained by this mapping, $\theta_2(t_i) \longrightarrow \{\mathcal{A}, \mathcal{B}\}$.

PLSWPA algorithm has the same structure of LSWPA algorithm, using the mapping θ_2 instead of θ_1 for the first phase, and the same list algorithm in the last phase. The three steps of PLSWPA algorithm can be summarized as follows:

1. Solve the relaxed model $(P3^{'})$.
2. Use the solution of $(P3^{'})$ to define another model $(P4^{'})$, then solve $(P4^{'})$.
3. After rounding the solutions obtained by $(P4^{'})$, use algorithm 7 with the obtained mapping θ_2 and a priority list L (LST, LFT, LLP).

Complexity:

Mapping θ_2 is based on solving two linear models with continuous variables, which are two polynomial problems. The following table 4.1 shows the number of different types of variables and constraints used in θ_1 and θ_2, where $\Gamma = max\{max(\Gamma^{-}(t_i), \Gamma^{+}(t_i)), i = \overline{1..n}\}$.

Table 4.1: *A comparison between the models used in θ_1 and θ_2*

Mapping strategies	**mapping θ_1**		**mapping θ_2**	
Models	**model $P1'$**	**model $P2'$**	**model $P3'$**	**model $P4'$**
Continuous variables	$3\|E\|+n+1$	$n+4\|E\|+1$	$n+\|E\|$	$n+2\|E\|$
Discrete variables	0	$\|E\|$	0	0
Total	$3\|E\|+n+1$	$5\|E\|+n+1$	$n+\|E\|$	$n+2\|E\|$
Linear constraints with discrete variables	0	$2\|E\|$	0	0
Linear constraints without discrete variables	$4\|E\|+2\Gamma+2$	$5\|E\|+2\Gamma+2$	$3\|E\|+2\Gamma+2$	$3\|E\|+2\Gamma+2$
Non-linear constraints	$\|E\|$	0	0	0
Total	$5\|E\|+2\Gamma+2$	$7\|E\|+2\Gamma+2$	$3\|E\|+2\Gamma+2$	$3\|E\|+2\Gamma+2$

The mapping θ_1 is obtained using two models $(P1^{'})$ and $(P2^{'})$. $(P1^{'})$ contains $|E|$ discrete variables and $(P2^{'})$ contains $|E|$ non-linear constraints. On the other hand, θ_2 is obtained using two models containing no discrete variables and without non-linear constraints. This makes polynomial the resolution of the first phase of PLSWPA algorithm. Furthermore, the second phase of PLSWPA and LSWPA algorithms are identical, with a $\mathcal{O}(max(\ell,k)n^2)$ time complexity.

In the following, we study the performance of PLSWPA algorithm in the worst case compared to the optimal solution.

Algorithm analysis:

We look in the following for the ratio between the solution obtained by PLSWPA algorithm $(\widehat{C}_{max1})$ and the optimal solution of the problem $(C^{\star}_{max})$. We denote by $C^{\star}_{max3'}$ (resp. $C^{\star}_{max4'}$) the optimal solution of the model $(P3^{'})$ (resp. $(P4^{'})$). We denote by $\widetilde{x}^{\star}_i$, the value of $\widetilde{x}^{'}_i$ in the optimal solution of $(P3^{'})$, $i \in \overline{1..n}$.

Theorem 8. *In the optimal solution obtained by the model $(P3^{'})$, if $\widetilde{x}^{\star}_i$ is an integer for all $i \in \overline{1..n}$, then the solutions of the models $(P3^{'})$ and $(P4^{'})$ are equal, i.e., $C^{\star}_{max4'} = C^{\star}_{max3'}$.*

Proof. By setting the $x^{'}_i$ value to $x^{'}_i = \widetilde{x}^{\star}_i$ in $(P4^{'})$, for all $i \in \overline{1..n}$, then for each two successive tasks $(t_i, t_j) \in E$, we have two cases:

1. if $\min\{\widetilde{x}^{\star}_i, \widetilde{x}^{\star}_j\} > \min\{1-\widetilde{x}^{\star}_i, 1-\widetilde{x}^{\star}_j\}$, then $Con_{i,j} = \begin{cases} y^{'}_{i,j} \leqslant x^{'}_i & (1) \\ y^{'}_{i,j} \leqslant x^{'}_j & (2) \\ \zeta^{'}_{i,j} = (1 - y^{'}_{i,j}) & (3) \end{cases}$

   Furthermore, $\min\{\widetilde{x}^{\star}_i, \widetilde{x}^{\star}_j\} = 1$ (since $\min\{\widetilde{x}^{\star}_i, \widetilde{x}^{\star}_j\}$ is strictly greater than $\min\{1-\widetilde{x}^{\star}_i, 1-\widetilde{x}^{\star}_j\}$), then $\widetilde{x}^{\star}_i = 1$ and $\widetilde{x}^{\star}_j = 1$, follows $\widetilde{x}^{\star}_i - \widetilde{x}^{\star}_j = 0 \leqslant \widetilde{\zeta}^{\star}_{i,j}$ ($\widetilde{\zeta}^{\star}_{i,j} = 0$ since it is a minimization problem). Furthermore, $\zeta^{'}_{i,j} = (1 - min\{x^{'}_i, x^{'}_j\}) = 1 - 1 = 0$, then $\zeta^{'}_{i,j} = \widetilde{\zeta}^{\star}_{i,j}$.

2. if $\min\{\widetilde{x}^{\star}_i, \widetilde{x}^{\star}_j\} \leqslant \min\{1-\widetilde{x}^{\star}_i, 1-\widetilde{x}^{\star}_j\}$, then $Con_{i,j} = \begin{cases} y^{'}_{i,j} \leqslant 1 - x^{'}_i & (1) \\ y^{'}_{i,j} \leqslant 1 - x^{'}_j & (2) \\ \zeta^{'}_{i,j} = (1 - y^{'}_{i,j}) & (3) \end{cases}$

- if $\min\{1-\widetilde{x}_i^\star, 1-\widetilde{x}_j^\star\} = 1$, then $1-\widetilde{x}_i^\star = 1$ and $1-\widetilde{x}_j^\star = 1$, follows $\widetilde{x}_i^\star - \widetilde{x}_j^\star = 0 \leqslant \widetilde{\zeta}_{i,j}^\star$ with $\widetilde{x}_i^\star = 0$ and $\widetilde{x}_j^\star = 0$. Furthermore, $\zeta_{i,j}^{'} = (1 - min\{1-x_i^{'}, 1-x_j^{'}\}) = 1-1 = 0$.
- if $\min\{1-\widetilde{x}_i^\star, 1-\widetilde{x}_j^\star\} = 0$, then $\widetilde{x}_i^\star = 1$ or $\widetilde{x}_j^\star = 1$. If $\widetilde{x}_i^\star = 1$ and $\widetilde{x}_j^\star = 0$, then $\widetilde{x}_i^\star - \widetilde{x}_j^\star = 1 \leqslant \widetilde{\zeta}_{i,j}^\star$. If $\widetilde{x}_i^\star = 0$ and $\widetilde{x}_j^\star = 1$, then $\widetilde{x}_j^\star - \widetilde{x}_i^\star = 1 \leqslant \widetilde{\zeta}_{i,j}^\star$. Furthermore, $\zeta_{i,j}^{'} = (1 - min\{1-x_i^{'}, 1-x_i^{'}\}) = 1-0 = 1$. In both cases, we have $\zeta_{i,j}^{'} = \widetilde{\zeta}_{i,j}^\star$.

Thus, the finish execution times of each task in the two models $(P3^{'})$ and $(P4^{'})$ are equal. Furthermore, since $\sum_{i=1}^{n} x_i^{'} w_{i,\mathcal{A}} = \sum_{i=1}^{n} x_i^\star w_{i,\mathcal{A}}$ and $\sum_{i=1}^{n}(1-x_i^{'})w_{i,\mathcal{B}} = \sum_{i=1}^{n}(1-x_i^\star)w_{i,\mathcal{B}}$, then the bounds imposed by constraints (5) and (6) in the two models $(P3^{'})$ and $(P4^{'})$ are also equal. Finally, $C_{max4'}^\star = C_{max3'}^\star$. □

However, finding the ratio between $C_{max4'}^\star$ and $C_{max3'}^\star$ for the general case is difficult. In the following, we suppose that $C_{max4'}^\star \leqslant \alpha C_{max3'}^\star \leqslant \alpha C_{max3}^\star$, with $\alpha \in \mathbb{R}^+$. Therefore, the ratio that we will now look for is relative to the solution of $C_{max3}^\star$.

To get an idea of the α value in the general case, we performed some numerical tests. Table 4.2 shows the standard deviation between $C_{max4'}^\star$ and $C_{max3'}^\star$ for 20 randomly generated instances of different sizes (DAG graphs). For each instance I_i, we compute $\alpha_i = \frac{C_{max4'}^\star(I_i)}{C_{max3'}^\star(I_i)}$. Then, we compute **Average GAP**= $\frac{\sum_{i=1}^{20}\alpha_i}{20}$ and **Standard deviation**= $\sqrt{\frac{\sum_{i=1}^{20}\alpha_i^2}{20}}$.

Table 4.2: *GAP and standard deviation between* $C_{max4'}^\star$ *and* $C_{max3'}^\star$.

Instances	Number of tasks	Average GAP	Standard deviation
test_1	10	1.12457	1.12871
test_2	30	1.11632	1.12065
test_3	60	1.00046	1.00046
test_4	100	1.00007	1.00007
test_5	200	1.00124	1.00126
test_6	400	1	1
test_7	500	1	1
test_8	600	1	1
test_9	800	1	1
test_10	1000	1	1
Average	/	1.024266	1.025115

From Table 4.2, we can notice that α tends towards 1 when we increase the size of the instances. What can be said, is that the solution of $C_{max4'}^\star$ is very close to the solution of $C_{max3'}^\star$ in the general case.

Lemma 12. *The ratio between the optimal solution* $C_{max4'}^\star$ *of the model* $P4'$ *and the optimal scheduling solution* $C_{max}^\star$ *of our main problem is given by* $C_{max4'}^\star \leqslant \alpha C_{max}^\star$.

Proof. The optimal solution C_{max3} of the model $(P3)$ does not take into account non-overlapping constraints, it represents a lower bound for the main problem, i.e., $C_{max3}^\star \leqslant C_{max}^\star$. Then, $C_{max4'}^\star \leqslant \alpha C_{max3'}^\star \leqslant \alpha C_{max3}^\star \leqslant \alpha C_{max}^\star$. □

Let w_i^r be the execution time of the task t_i by considering the rounding θ_2, where $w_i^r = w_{i,\mathcal{A}}$ if $x_i^r = 1$ and $w_i^r = w_{i,\mathcal{B}}$ if $x_i^r = 0$. Let $w_i^{'}$ be the execution time of the task t_i by considering the solution of $(P4^{'})$, where $w_i^{'} = x_i^{'} w_{i,\mathcal{A}} + (1-x_i^{'})w_{i,\mathcal{B}}$, $i = \overline{1..n}$.

Lemma 13. *The relation between w_i^r and $w_i^{'}$ of each task t_i is given by $w_i^r \leqslant 2w_i^{'}$, $i = \overline{1..n}$.*

Proof. Mappings θ_1 and θ_2 are defined in the same way, i.e., $x_i^r = 0$ if $x_i^{'} < \frac{1}{2}$, and $x_i^r = 1$ otherwise. Thus, from Proposition 3, we have $2w_i^{'} = 2x_i^{'}w_{i,\mathcal{A}} + 2(1-x_i^{'})w_{i,\mathcal{B}} \geqslant x_i^r w_{i,\mathcal{A}} + (1-x_i^r)w_{i,\mathcal{B}} = w_i^r$, and then $w_i^r \leqslant 2w_i^{'}$. □

Lemma 14. *For two successive tasks $(t_i, t_j) \in E$, if t_i and t_j are executed by two different processing elements, then $\zeta_{i,j}^{'} > \frac{1}{2}$.*

Proof. We have two cases:

1. If $\min\{\widetilde{x}_i^{'}, \widetilde{x}_j^{'}\} > \min\{1-\widetilde{x}_i^{'}, 1-\widetilde{x}_j^{'}\}$, then from $Con_{i,j}$ we have $\zeta_{i,j}^{'} = (1 - min\{x_i^{'}, x_j^{'}\})$:
   a. If $x_i^{'} < \frac{1}{2}$ and $x_j^{'} \geqslant \frac{1}{2}$, then $\zeta_{i,j}^{'} = (1-x_i^{'}) > \frac{1}{2}$.
   b. If $x_i^{'} \geqslant \frac{1}{2}$ and $x_j^{'} < \frac{1}{2}$, then $\zeta_{i,j}^{'} = (1-x_j^{'}) > \frac{1}{2}$.
2. If $\min\{\widetilde{x}_i^{'}, \widetilde{x}_j^{'}\} \leqslant \min\{1-\widetilde{x}_i^{'}, 1-\widetilde{x}_j^{'}\}$, then from $Con_{i,j}$ we have $\zeta_{i,j}^{'} = (1 - min\{1-x_i^{'}, 1-x_j^{'}\})$:
   a. If $1-x_i^{'} < \frac{1}{2}$ and $1-x_j^{'} \geqslant \frac{1}{2}$, then $\zeta_{i,j}^{'} = x_i^{'} > \frac{1}{2}$.
   b. If $1-x_i^{'} \geqslant \frac{1}{2}$ and $1-x_j^{'} < \frac{1}{2}$, then $\zeta_{i,j}^{'} = x_j^{'} > \frac{1}{2}$.

□

For each couple of tasks $(t_i, t_j) \in E$, we denote by $Cost_{i,j}^r$ the value given by $Cost_{i,j}^r = 0$ if $x_i^r = x_j^r$, $Cost_{i,j}^r = cm_{i,j}$ otherwise. Let $Cost_{i,j}^{'}$ be the value given by $Cost_{i,j}^{'} = \zeta_{i,j}^{'} cm_{i,j}$.

Lemma 15. *For each couple of tasks $(t_i, t_j) \in E$, the relation between $Cost_{i,j}^r$ and $Cost_{i,j}^{'}$ is given by $Cost_{i,j}^r < 2Cost_{i,j}^{'}$.*

Proof. If t_j and t_j are executed by the same processing element, $Cost_{i,j}^r = 0 \leqslant 2\zeta_{i,j}^{'} cm_{i,j}$, because $\zeta_{i,j}^{'} \geqslant 0$. If t_j and t_j are executed by two different processing elements, then $Cost_{i,j}^r = cm_{i,j}$. Then, from Lemma 14, $\zeta_{i,j}^{'} > \frac{1}{2}$, then $2\zeta_{i,j}^{'} > 1$, follows $Cost_{i,j}^r = cm_{i,j} \leqslant 2\zeta_{i,j}^{'} cm_{i,j} = 2Cost_{i,j}^{'}$. □

Proposition 7. *For each two successive tasks $(t_i, t_j) \in E$, let be $l_{i,j}^r = w_i^r + Cost_{i,j}^r + w_j^r$ (resp. $l_{i,j}^{'} = w_i^{'} + Cost_{i,j}^{'} + w_j^{'}$) the length of (t_i, t_j) in PLSWPA solution (resp. $(P4^{'})$ solution), then, we have $l_{i,j}^r < 2l_{i,j}^{'}$.*

Proof. From Lemma 13 and Lemma 15, $l_{i,j}^r = w_i^r + Cost_{i,j}^r + w_j^r < 2w_i^{'} + 2Cost_{i,j}^{'} + 2w_j^{'} = 2l_{i,j}^{'}$. Thus, $l_{i,j}^r < 2l_{i,j}^{'}$. □

Lemma 16. *Let $\widehat{C}_{max1}$ be the solution obtained by using PLSWPA algorithm, then, $\widehat{C}_{max1} < 6C_{max4'}^{\star}$.*

Proof. From Proposition 7, the length of each path L from $G(V,E)$ is given by $length(L)^r = \sum_{(t_i,t_{i+1})\in L} l_{i,i+1}^r \leqslant 2\sum_{(t_i,t_{i+1})\in L} l_{i,i+1}^{'} = 2length(L)^{'}$, where $length(L)^r$ (resp. $length(L)^{'}$) is the length of L in PLSWPA solution (resp. $(P4^{'})$ solution). Furthermore, the workload of the tasks assigned to the processing elements of type $\mathcal{A}$ (resp. $\mathcal{B}$) is given by $\sum_{i=1}^n x_i^r w_{i,\mathcal{A}} = 2\sum_{i=1}^n x_i^{'} w_{i,\mathcal{A}}$ (resp. $\sum_{i=1}^n (1-x_i^r) w_{i,\mathcal{A}} = 2\sum_{i=1}^n (1-x_i^{'}) w_{i,\mathcal{A}}$). Finally, form lemmas 13 and 15, we proved that models $(P1^{'})$ and $(P4')$ satisfy the same properties. Thus, since the second phases of LSWPA and PLSWPA algorithms are identical, then the result obtained by theorem 7 can be applied to the model $(P4')$, i.e., $\widehat{C}_{max1} < 6C_{max4'}^{\star}$.

□

Theorem 9. *The ratio between the solution $\widehat{C}_{max1}$ obtained by PLSWPA algorithm and the optimal scheduling solution $C^{\star}_{max}$ of our main problem is given by $\frac{\widehat{C}_{max1}}{C^{\star}_{max}} < 6\alpha$.*

Proof. From Lemma 12, we have $C^{\star}_{max4'} \leqslant \alpha C^{\star}_{max}$. From Lemma 16, we have $\widehat{C}_{max1} < 6C^{\star}_{max4'}$. Then, $\frac{\widehat{C}_{max1}}{C^{\star}_{max}} < \frac{6C^{\star}_{max4'}}{C^{\star}_{max}} \leqslant \frac{6\alpha C^{\star}_{max}}{C^{\star}_{max}} \leqslant 6\alpha$. □

Iterative mapping:

In order to find a more efficient rounding than θ_1 and θ_2 described previously, we try to assign the tasks progressively. Let θ_3 be the rounding obtained by Algorithm 8.

Algorithm 8: Rounding algorithm.

```
Data: model (P) ((P1'), (P2') or (P4')), υ ⩾ 2.
Result: mapping θ3.
begin
    for it = 1 to ⌊υ/2⌋ do
        Solve (P);
        for j = 1 to n do
            if x'_j < it/υ then
                Set in (P) : x'_j = 0
            if x'_j ⩾ 1 − it/υ then
                Set in (P) : x'_j = 1
    for l = 1 to n do
        if x'_l < 1/2 then
            x^r_l = 0
        else
            x^r_l = 1
```

For a given integer $\upsilon \geqslant 2$, we solve the model (P) ($(P1')$ or $(P2')$ for LSWPA algortihm, $(P4')$ for PLSWPA algortihm) $\lfloor \frac{\upsilon}{2} \rfloor$ times by adding new assignment constraints at each resolution. Contrary to θ_1 and θ_2, we try to assign the tasks progressively, starting by setting x'_j to 0 if $x'_j < (\frac{1}{\upsilon})$ and x'_j to 1 if $x'_j \geqslant 1 - (\frac{1}{\upsilon})$ for the first resolution of (P), and finishing by setting x'_j to 0 if $x'_j < (\frac{\lfloor \frac{\upsilon}{2} \rfloor}{\upsilon})$ and x'_j to 1 if $x'_j \geqslant 1 - (\frac{\lfloor \frac{\upsilon}{2} \rfloor}{\upsilon})$, where $\frac{\lfloor \frac{\upsilon}{2} \rfloor}{\upsilon} \leqslant \frac{1}{2}$ according to Lemma 17.

Thus, for each iteration it, we solve a model (P), then we add assignment constraints to it to obtain another model to solve in iteration $it+1$. For each iteration $it \leqslant \lfloor \frac{\upsilon}{2} \rfloor$, we add assignment constraints for each task t_j, $j = \overline{1..n}$, such that:

- If $x'_j < \frac{it}{\upsilon}$, we set in (P) $x'_j = 0$ as a new affectation constraint.
- If $x'_j \geqslant 1 - \frac{it}{\upsilon}$, we set in (P) $x'_j = 1$ as a new affectation constraint.

For example, if $\upsilon = 10$, then we obtain for the first iteration $it = 1$ for $j = \overline{1..n}$:

- If $x'_j < \frac{1}{10}$, we set in (P) $x'_j = 0$ as a new affectation constraint.
- If $x'_j \geqslant 1 - \frac{1}{10}$, we set in (P) $x'_j = 1$ as a new affectation constraint.

For $\upsilon = 10$, algorithm 8 will have $\lfloor \frac{10}{2} \rfloor = 5$ iterations (the model (P) will be solved 5 times by adding new constraints at each iteration). But if $\upsilon = 20$, algorithm 8 will have $\lfloor \frac{20}{2} \rfloor = 10$ iterations, and we obtain for the first iteration $it = 1$ for $j = \overline{1..n}$:

- If $x_j^{'} < \frac{1}{20}$, we set in (P) $x_j^{'} = 0$ as a new affectation constraint.
- If $x_j^{'} \geqslant 1 - \frac{1}{20}$, we set in (P) $x_j^{'} = 1$ as a new affectation constraint.

After the last iteration, it may be that some tasks are not assigned, they are then assigned using the same principle as the rounding θ_1 or θ_2, i.e., for each task t_i with $i \in \overline{1..n}$, we set $x_i^r = 0$ if $x_i^{'} < \frac{1}{2}$, and $x_i^r = 1$ otherwise.

Thus, the solution obtained by algorithm 8 depends on the value of υ. In Section 4.5.3, we will study the behaviour of Algorithm 8 by varying the value of υ. We compare the solution obtained by using the rounding θ_3 generated for each υ value.

Remark 7. For $\upsilon = 2$, we obtain the rounding θ_1.

Lemma 17. *For an integer* $\upsilon \geqslant 2$, $\frac{\lfloor \frac{\upsilon}{2} \rfloor}{\upsilon} \leqslant \frac{1}{2}$.

Proof. According to the parity of υ, we distinguish two possible cases:

1. υ is even, $\upsilon = 2\lambda : \frac{\lfloor \frac{\upsilon}{2} \rfloor}{\upsilon} = \frac{\lambda}{2\lambda} = \frac{1}{2}$
2. υ is odd, $\upsilon = 2\lambda + 1 : \frac{\lfloor \frac{\upsilon}{2} \rfloor}{\upsilon} = \frac{\lambda}{2\lambda+1} < \frac{1}{2}$.

□

Remark 8. The specialized accelerator problem (where a set of tasks $T^{'} \in T$ can be executed by only one type of processing elements, i.e., by a CPU or a GPU only). can be solved with the same methods (LSWPA and PLSWPA). It is sufficient to set the corresponding variables $x_i^{'}$ to 0 or 1 in the models $(P1^{'})$ or $(P2^{'})$ for LSWPA algorithm, $(P3^{'})$ and $(P4^{'})$ for PLSWPA algorithm.

4.5 Numerical results

In this section, we compare the performance of LSWPA, PLSWPA and HEFT [26] algorithms. The benchmarks are generated by Turbine [108]. It allows us to generate random DAG graphs, with the possibility to specify each graph characteristic (number of tasks, cost intervals, parallelism degree...). In what follows, we first describe the generation of benchmarks and all used parameters. Then, we discuss the efficiency of the first phase of LSWPA algorithm using models $(P1^{'})$ and $(P2^{'})$, and then the efficiency of its second phase using the mapping θ_1 and then θ_3, with the different lists (LST, LFP, LLP) given in Section 4.4.1.

Then, the most effective list (LST, LFP or LLP) for LSWPA algorithm will be used to compare the performance of PLSWPA algorithm (using θ_2 and θ_3) to LSWPA and HEFT algorithms.

4.5.1 Benchmark

The benchmark used for all tests is composed of ten parallel DAG applications. We denote by $test\_i$ the instance number i. We generate 10 different applications for each $test\_i$ with $i = \overline{1..10}$. The execution times of the tasks are generated randomly over an interval $[w_{min}, w_{max}]$, w_{min} has been fixed at 5 and w_{max} at 30. The number of successors of each task is generated randomly over an interval $[d_{min}, d_{max}]$, d_{min} has been fixed at 1 and d_{max} at 10.

Furthermore, communication delay for each arc was generated on an interval $[cm_{min}, cm_{max}]$, we set cm_{min} to 35 and cm_{max} to 50. Table 4.3 presents the size of each instance generated as well as the number of processing elements of type $\mathcal{A}$ (given by ℓ) and type $\mathcal{B}$ (given by k) used to execute each instance. For Platform 1, we increase the number of processing elements while

increasing the size of the applications. For Platform 2, we only use one processor of type $\mathcal{A}$ and one processor of type $\mathcal{B}$.

Table 4.3: *Description of the applications and the platforms.*

Instances	Number of tasks	Platform 1		Platform 2	
		ℓ	k	ℓ	k
test_1	10	3	3	1	1
test_2	30	4	4	1	1
test_3	60	4	4	1	1
test_4	100	6	6	1	1
test_5	200	6	6	1	1
test_6	400	6	6	1	1
test_7	500	8	8	1	1
test_8	600	8	8	1	1
test_9	800	8	8	1	1
test_10	1000	12	12	1	1

4.5.2 LSWPA algorithm compared to HEFT algorithm

To study the performance of our method, we compared the ratio between each makespan value obtained by LSWPA[1] algorithm with the solution of HEFT algorithm, the optimal solution obtained by *CPLEX* (if it is possible) and the lower bound C'_{max} obtained by $(P1')$ or $(P2')$. The results obtained for the two platforms 1 and 2 are presented in the following.

Platform 1

Opt presents the number of instances where HEFT provides the optimal solution. We take the number of solutions equal to the optimal solution provided by *CPLEX* or to the value of the lower bound C'_{max} obtained by $(P1')$ or $(P2')$. Column **Best HEFT** presents the number of instances where HEFT algorithm provides better or the same solution obtained by LSWPA algorithm using the rounding θ_1 or θ_3 for the three lists (LST, LFT, LLP). Finally, column **Time HEFT** presents the average time that was needed for HEFT to provide a solution. A line Average is added at the end of each table which represents the average of the values of each column.

Table 4.5 (resp. 4.6) shows the results of tests of LSWPA algorithm on 10 instances for each application size given in column **Inst** using three lists (LST[2], LFP[3], LLP[4]) with the rounding θ_1 (resp. θ_3) on the same platform 1. For the rounding θ_3, we set $v = 10$, which makes about 5 iterations for Algorithm 8. The next three columns concern the results of LSWPA algorithm using LST, where the column **GAP** gives the average ratio between makespan obtained by LSWPA algorithm and C'_{max}, $GAP = \frac{\text{LSWPA makespan} - C'_{max}}{C'_{max}} \times 100$. Column **Best** presents the number of instances where LSWPA algorithm provides better or the same solution obtained by using the list LST instead of LFT or LLP. Column **Opt** presents the number of instances where LSWPA provides optimal solution using LST (for small instances). We take the number

[1] List Scheduling With Pre-Allocation (LSWPA) algorithm
[2] List by Start Time (LST)
[3] List by Finish Time (LFT)
[4] List by longest path (LLP)

of solutions equal to the optimal solution provided by *CPLEX*. The next six columns concern the results of LSWPA algorithm using the lists LFT and LLP. Column **Best LS with θ_1** (resp. **Best LS with θ_3** in table 4.6) presents the number of instances where LSWPA algorithm provides better or the same solution obtained by using the rounding θ_1 (resp. θ_3) and the best list from the three lists (LST,LFT,LLP) compared to the solution obtained by HEFT and LSWPA algorithm using θ_3 (resp. θ_1) with the three lists (LST,LFT,LLP). Finally, column **Time ($P1'$)** (resp. **Time ($P1'$)**) gives the average time that was needed for LSWPA algorithm to provide a solution using the model ($P1'$) (resp. ($P2'$)) for the first phase.

Obtaining the optimal solution using *CPLEX* is very expensive in term of running time. As an example, for the instance test_3 with 60 tasks and 4 processors of type $\mathcal{A}$ and 4 processors of type $\mathcal{B}$, *CPLEX* cannot provide the optimal solution after 14 hours. Thus, we compare our method to the optimal solution for only the first two instances. The aim of these tables is to compare LSWPA algorithm to HEFT algorithm, and determine which is the most efficient list (LST, LFP or LLP) for the list algorithm used in the second phase by LSWPA. We will evaluate after the most efficient υ value for the mapping θ_3.

Table 4.4: *HEFT algorithm results and CPLEX running time for platform 1.*

Instances	Number tasks	Number of ℓ	Number of k	Time *CPLEX*	HEFT Gap	HEFT Opt	Best HEFT	Time HEFT
test_1	10	3	3	5m	11.08%	5	5	0.01s
test_2	30	4	4	8h	13.48%	4	4	0.01s
test_3	60	4	4	/	22.16%	/	2	0.02s
test_4	100	6	6	/	16.33%	/	3	0.02s
test_5	200	6	6	/	19.31%	/	4	0.04s
test_6	400	6	6	/	16.72%	/	0	0.09s
test_7	500	8	8	/	15.05%	/	1	0.15s
test_8	600	8	8	/	15.05%	/	0	0.20s
test_9	800	8	8	/	11.79%	/	0	0.33s
test_10	1000	12	12	/	15.26%	/	0	0.64s
Average	/	/	/	>4h	15.62%	/	19%	0.15s

Table 4.5: *LSWPA algorithm results using the lists (LST, LFP, LLP) with rounding θ_1 in platform* 1.

Inst	*LST* Gap	*LST* Best	*LST* Opt	*LFT* Gap	*LFT* Best	*LFT* Opt	*LLP* Gap	*LLP* Best	*LLP* Opt	Best LS with θ_1	Time $P1'$	Time $P2'$
test_1	0%	10	10	0%	10	10	0%	10	10	10	0.04s	0.06s
test_2	4.51%	3	3	3.31%	4	4	2.35%	8	5	6	0.01s	0.11s
test_3	17.61%	0	/	17.98%	0	/	11.25%	8	/	5	0.10s	0.16s
test_4	13.39%	1	/	13.43%	1	/	7.38%	7	/	4	0.29s	0.35s
test_5	27.69%	2	/	26.06%	2	/	15.93%	7	/	6	0.76s	0.67s
test_6	25.93%	3	/	25.36%	3	/	12.82%	4	/	1	4.52s	2.64s
test_7	28.23%	1	/	26.6%	2	/	14.32%	6	/	2	6.11s	3.75s
test_8	22.79%	0	/	22.39%	1	/	11.51%	9	/	0	8.28s	4.85s
test_9	14.87%	0	/	14.07%	0	/	2.204%	10	/	3	10.82s	5.47s
test_10	20.55%	0	/	18.90%	1	/	5.32%	9	/	0	13.55s	6.65s
Average	17.55%	20%	/	16.81%	22%	/	8.30%	78%	/	37%	4.44s	2.47s

Table 4.6: *LSWPA algorithm results using the lists (LST, LFP, LLP) with rounding θ_3 in platform* 1.

Inst	*LST*			*LFT*			*LLP*			Best LS	Time	
	Gap	Best	Opt	Gap	Best	Opt	Gap	Best	Opt	with θ_3	$P1'$	$P2'$
test_1	0%	10	10	0%	10	10	0%	10	10	10	0.07s	0.09s
test_2	2.92%	5	5	1.24%	5	5	0.09%	9	6	9	0.018s	0.16s
test_3	17.77%	0	/	14.16%	0	/	10.31%	8	/	5	0.17s	0.17s
test_4	15.64%	2	/	11.32%	2	/	7.76%	5	/	6	0.31s	0.34s
test_5	36.33%	2	/	25.29%	4	/	21.08%	5	/	4	0.85s	0.83s
test_6	38.07%	0	/	20.59%	0	/	9.28%	10	/	9	5.01s	2.95s
test_7	33.61%	0	/	21.66%	1	/	11.10%	8	/	8	6.71s	4.59s
test_8	36.51%	0	/	18.19%	0	/	6.96%	10	/	10	8.63s	5.28s
test_9	40.07%	0	/	13.71%	0	/	2.01%	10	/	7	11.55s	7.06s
test_10	39.33%	0	/	18.63%	0	/	4.77%	10	/	10	15.10s	8.13s
Average	26.02%	19%	/	14.47%	22%	/	7.33%	85%	/	78%	4.84s	2.96s

From Table 4.5 and 4.6, we can notice that the list LLP is better than LFT and LST. Using the list LLP and θ_1, LSWPA algorithm provides 78% of the best solutions compared to 20% (resp. 22%) obtained using LST (resp. LFT). Using the list LLP and θ_3, LSWPA algorithm does even better, and provides 85% of the best solutions compared to 19% (resp. 22%) obtained using LST (resp. LFT).

Comparing to table 4.4 results, LSWPA algorithm provides a better solution than HEFT using rounding θ_1 or θ_3 with the list LFT or LLP. Furthermore, LSWPA algorithm using rounding θ_3 and the list LLP is the most efficient method, with 78% of the best solutions against 17% using θ_1, and a ratio of 7.33% comparing to the lower bound C'_{max}.

For the running time, HEFT algorithm needs less time than LSWPA algorithm to provide a solution, where the first iteration of the rounding algorithm 8 (θ_3) takes the same time than the time needed to provide θ_1 (Remark 7). Finally, we notice that the model ($P2'$) is more efficient than ($P1'$) in running time using rounding θ_1 or θ_3, where LSWPA algorithm provides a solution in less than 3 seconds for instances of 1000 tasks using the model ($P2'$), while it needs more than 4 seconds using the model ($P1'$). Thus, it may provide better running time for much bigger instances.

Platform 2

For this platform, we suppose that we have only one processor of type $\mathcal{A}$ and one of type $\mathcal{B}$. *CPLEX* is more efficient than on the platform 1, but the running time is still large. For instance test_3 with 60 tasks, *CPLEX* cannot provide the optimal solution after 6 hours. Thus, we compare our method to the optimal solution for only the first two instances. Tables 4.7, 4.8 and 4.9 present the results obtained for platform 2.

Table 4.7: *HEFT algorithm results and CPLEX running time for platform 2.*

Instances	Number tasks	Number of ℓ	Number of k	Time *CPLEX*	HEFT		Best HEFT	Time HEFT
					Gap	Opt		
test_1	10	1	1	0.39s	20.58%	4	6	0.01s
test_2	30	1	1	1h40	44.31%	3	4	0.01s
test_3	60	1	1	/	29.99%	/	3	0.01s
test_4	100	1	1	/	17.26%	/	1	0.02s
test_5	200	1	1	/	12.17%	/	0	0.03s
test_6	400	1	1	/	11.26%	/	0	0.05s
test_7	500	1	1	/	12.15%	/	0	0.09s
test_8	600	1	1	/	11.86%	/	2	0.11s
test_9	800	1	1	/	11.21%	/	0	0.14s
test_10	1000	1	1	/	11.90%	/	0	0.22s
Average	/	/	/	>50m	18.26%	/	16%	0.06s

Table 4.8: *LSWPA algorithm results using the lists (LST, LFP, LLP) with rounding θ_1 in platform 2.*

Inst	*LST*			*LFT*			*LLP*			Best LS	Time	
	Gap	Best	Opt	Gap	Best	Opt	Gap	Best	Opt	with θ_1	$P1'$	$P2'$
test_1	18.57%	7	5	18.57%	7	5	17.61%	8	5	8	0.04s	0.08s
test_2	52.72%	1	1	49.55%	1	1	41.05%	6	3	5	0.30s	0.31s
test_3	44.14%	0	/	41.18%	0	/	30.69%	6	/	5	0.35s	0.35s
test_4	25.00%	0	/	24.02%	0	/	9.80%	7	/	6	0.05s	0.06s
test_5	5.94%	2	/	5.93%	3	/	0.33%	10	/	10	0.07s	0.08s
test_6	0.31%	8	/	0.31%	8	/	0.31%	10	/	4	0.40s	0.59s
test_7	1.59%	8	/	1.54%	8	/	0.34%	10	/	4	0.27s	0.37s
test_8	0.19%	10	/	0.19%	10	/	0.19%	10	/	8	0.38s	0.47s
test_9	0.61%	9	/	0.39%	9	/	0.05%	10	/	5	0.64s	0.72s
test_10	0.16%	9	/	0.16%	10	/	0.16%	10	/	5	0.88s	0.90s
Average	14.92%	54%	/	14.18%	56%	/	10.05%	87%	/	60%	0.33s	0.39s

Table 4.9: *LSWPA algorithm results using the lists (LST, LFP, LLP) with rounding θ_3 in platform 2.*

Inst	*LST*			*LFT*			*LLP*			Best LS	Time	
	Gap	Best	Opt	Gap	Best	Opt	Gap	Best	Opt	with θ_3	$P1'$	$P2'$
test_1	17.26%	8	6	17.26%	8	6	16.30%	9	6	9	0.07s	0.09s
test_2	55.68%	1	1	49.07%	1	1	43.03%	6	3	3	0.33s	0.35s
test_3	60.16%	0	/	39.38%	0	/	29.21%	6	/	5	0.40s	0.40s
test_4	41.96%	0	/	23.31%	0	/	9.11%	9	/	9	0.11s	0.14s
test_5	20.52%	0	/	6.22%	2	/	0.43%	10	/	7	0.17s	0.23s
test_6	11.41%	0	/	0.15%	8	/	0.15%	10	/	10	0.62s	0.82s
test_7	8.73%	0	/	1.68%	8	/	0.19%	10	/	8	0.52s	0.82s
test_8	8.08%	2	/	0.11%	10	/	0.11%	10	/	9	0.77s	1.13s
test_9	10.90%	0	/	0.44%	9	/	0.03%	10	/	8	1.25s	1.89s
test_10	8.67%	0	/	0.08%	10	/	0.08%	10	/	8	1.32s	1.91s
Average	24.33%	11%	/	9.96%	56%	/	9.86%	90%	/	76%	0.55s	0.77s

From Table 4.8 and Table 4.9, we can notice that the list LLP is still better than LFT and LST. Comparing to Table 4.7 results, LSWPA algorithm also provides a better solution than HEFT using rounding θ_1 or θ_3 with the list LFT or LLP. Furthermore, LSWPA algorithm using rounding θ_3 and list LLP is the most efficient method, with 76% of the best solutions and a ratio of 9.86% comparing to the lower bound. For the running time, HEFT algorithm also needs less time than LSWPA algorithm to provide a solution. Finally, unlike the platform 1, we notice that the model ($P1'$) is more efficient than ($P2'$) in running time using rounding θ_1 or θ_3, where LSWPA algorithm provides a solution in less than 1 second for an instance of 1000 tasks using the two models.

These first results showed that the list LLP is the most effective for the second phase of LSWPA algorithm. However, it is difficult to say which model $P1'$ or $P2'$ is more effective (running time) for the first phase, where a solver (*CPLEX* in this case) can interpret each model differently for each instance (number of discrete variables, number of non-linear constraints, ...). Considering the results obtained for the two platforms, it can be said that $P2'$ is slightly faster than $P1'$, where the ratio between average times of the two models for the two roundings on the two platforms is given by:

$$\frac{\text{Average time of } P1'}{\text{Average time of } P2'} = \frac{4.44 + 4.84 + 0.33 + 0.55}{2.47 + 2.96 + 0.39 + 0.77} = \frac{10.16}{6.59} = 1.54 \text{ seconds}$$

In the following, we study the behaviour of θ_3 by varying the value of υ as input of algorithm 8, which will determine the average number of iterations necessary to obtain a good mapping (which in general allows to provide a good makespan after the second phase).

4.5.3 Best value of υ for the rounding θ_3

The tests previously performed have shown that the list LLP is more effective for LSWPA algorithm using θ_1 with $P2'$ and θ_3 with $\upsilon = 10$. On tables 4.10 and 4.11, we tested other υ values ($\upsilon = 20$ and $\upsilon = 30$) to see if we can get better results by using LSWPA with θ_3 if we increase the number of iterations of Algorithm 8 on the two platforms 1 and 2.

Table 4.10: *Best υ for the rounding θ_3 in platform 1.*

Inst	Number of tasks	LSWPA											
		using θ_1 with $P2'$			using θ_3 with $\upsilon = 10$			using $\theta_3$with $\upsilon = 20$			using θ_3 with $\upsilon = 30$		
		GAP	Time	Best	GAP	Time	Best	GAP	Time	Best	GAP	Time	Best
test_1	10	0.050%	0.13s	10	0.05%	0.34s	10	0.05%	0.32s	10	0.05%	0.37s	10
test_2	30	0.615%	1.19s	9	0.184%	1.66s	9	0.18%	1.38s	9	0.184%	2.11s	9
test_3	60	5.80%	1.30s	2	2.98%	1.42s	4	2.98%	1.70s	4	2.98%	2.10s	4
test_4	100	5.94%	2.88s	1	4.09%	4.80s	3	4.09%	5.06s	3	4.09%	7.40s	3
test_5	200	8.08%	8.58s	3	10.41%	12.01s	3	10.41%	12.62s	3	10.41%	14.03s	3
test_6	400	8.19%	6.76s	1	6.02%	9.86s	6	6.02%	12.04s	6	6.02%	14.08s	6
test_7	500	10.22%	9.88s	0	5.80%	11.88s	4	5.80%	12.53s	4	5.80%	18.93s	4
test_8	600	4.41%	10.68s	4	3.65%	11.52s	5	3.65%	12.83s	5	3.65%	14.62s	5
test_9	800	0.65%	13.31s	7	0.65%	16.60s	7	0.65%	17.96s	7	0.65%	20.67s	7
test_10	1000	3.64%	17.24s	4	2.44%	18.96s	6	2.44%	19.95s	6	2.44%	21.47s	6
Average	/	4.75%	7.19s	41%	3.62%	8.9s	57%	3.62%	9.63s	57%	3.62%	11.57s	57%

Table 4.11: *Best υ for the rounding θ_3 in platform 2*

Inst	Number of tasks	LSWPA											
		using θ_1 with $P2'$			using θ_3 with $\upsilon = 10$			using $\theta_3$with $\upsilon = 20$			using θ_3 with $\upsilon = 30$		
		GAP	Time	Best	GAP	Time	Best	GAP	Time	Best	GAP	Time	Best
test_1	10	1.07%	0.02s	8	0.09%	0.06s	9	0.09%	0.11s	9	0.092%	0.17s	9
test_2	30	2.12%	0.080s	7	1.26%	0.22s	8	1.26%	0.40s	8	1.26%	0.61s	8
test_3	60	8.41%	0.11s	0	4.95%	0.31s	4	4.95%	0.55s	4	4.95%	0.89s	4
test_4	100	8.77%	0.22s	1	6.42%	0.55s	4	6.42%	0.96s	4	6.42%	1.51s	4
test_5	200	11.46%	0.75s	1	10.87%	1.40s	3	10.87%	2.37s	3	10.87%	3.62s	3
test_6	400	8.37%	1.86s	2	5.90%	3.23s	5	5.90%	5.13s	5	5.90%	7.58s	5
test_7	500	9.11%	2.22s	0	5.56%	2.60s	7	5.56%	3.41s	7	5.56%	4.76s	7
test_8	600	5.03%	4.28s	7	3.78%	4.81s	8	3.78%	5.35s	8	3.78%	7.36s	8
test_9	800	0.75%	9.91s	8	0.62%	10.51s	9	0.62%	11.77s	9	0.62%	14.21s	9
test_10	1000	3.92%	15.28s	4	2.52%	15.28s	5	2.52%	16.24s	5	2.52%	17.65s	5
Average	/	5.90%	3.47s	38%	4.19%	3.89s	62%	4.19%	4.62s	62%	4.19%	5.83s	62%

We can notice that for all the generated instances, the average solutions obtained by using $\upsilon = 20$ or $\upsilon = 30$ are equivalent to the solution obtained by $\upsilon = 10$. Thus, we can say that for instances of size less than 1000 tasks, $\upsilon = 10$ is a good choice for the rounding θ_3. For large instances, it may be interesting to increase the value of υ to obtain better solutions.

4.5.4 LSWPA algorithm evaluation using different mappings

To see the efficiency of the LSWPA algorithm using the optimal mapping obtained using the model $P3$ introduced by remark 5, we have performed several tests on the two platforms 1 and 2.

Table 4.12 and Table 4.13 compare for the two platforms 1 and 2, the average running time and the average solution obtained by LSWPA algorithm using: θ_1 with the model $P2^{'}$, then θ_3 with $\upsilon = 10$, then θ_1 based on the solutions obtained by the relaxed model $P3^{'}$ (instead of $P1^{'}$ or $P1^{'}$), and finally the optimal mapping obtained by $P3$.

Table 4.12: *LSWPA algorithm evaluation for platform* 1.

Inst	Number of tasks	LSWPA using θ_1 with $P2^{'}$			LSWPA using θ_3 with $\upsilon = 10$			LSWPA using θ_1 with $P3^{'}$			LSWPA using θ_1 with $P3$		
		GAP	Time	Best	GAP	Time	Best	GAP	Time	Best	GAP	Time	Best
test_1	10	0%	0.31s	10	0%	0.36s	10	0.91%	0.005s	9	0%	0.019s	10
test_2	30	2.41%	1.48s	7	0.94%	0.83s	9	12.67%	0.013s	4	0.35%	0.04s	9
test_3	60	5.17%	1.45s	5	5.14%	1.53s	2	12.99%	0.02s	1	4.40%	1.77s	7
test_4	100	3.28%	3.40s	4	2.07%	6.14s	6	9.79%	0.11s	1	1.34%	1.33s	5
test_5	200	11.04%	3.67s	2	9.69%	4.55s	2	53.09%	0.075s	0	9.72%	1.30s	6
test_6	400	11.22%	4.16s	2	9.67%	5.84s	4	43.07%	0.54s	0	10.15%	10.55s	4
test_7	500	9.65%	5.93s	3	9.66%	11.30s	3	46.50%	2.50s	0	6.89%	17.66s	6
test_8	600	10.41%	7.22s	0	7.71%	16.86s	3	19.44%	1.43s	0	6.12%	44.89s	7
test_9	800	5.89%	7.11s	1	4.33%	10.23s	1	7.55%	2.51s	1	2.24%	82.67s	8
test_10	1000	6.13%	8.66s	1	2.79%	8.20s	1	13.50%	8.81s	1	2.36%	438.82s	8
Average	/	6.52%	4.33s	35%	5.2%	6.58s	41%	21.95 %	1.60s	17%	4.35%	59.90s	70%

Table 4.13: *LSWPA algorithm evaluation for platform* 2.

Inst	Number of tasks	LSWPA using θ_1 with $P2^{'}$			LSWPA using θ_3 with $\upsilon = 10$			LSWPA using θ_1 with $P3^{'}$			LSWPA using θ_1 with $P3$		
		GAP	Time	Best	GAP	Time	Best	GAP	Time	Best	GAP	Time	Best
test_1	10	14.17%	3.82s	8	16.90%	0.80s	6	28.96%	0.002s	4	15.14%	0.03s	7
test_2	30	40.89%	3.13s	4	42.58%	2.52s	4	57.56%	0.015s	4	56.26%	3.74s	1
test_3	60	23.69%	4.13s	5	24.50%	2.89s	4	26.29%	0.024s	4	26.42%	18.82s	2
test_4	100	4.29%	0.19s	4	4.29%	0.37s	4	5.71%	0.04s	1	4.84%	28.13s	7
test_5	200	0.36%	0.68s	1	0.33%	0.96s	1	0.86%	0.10s	1	0.079%	28.68s	9
test_6	400	0.25%	2.19s	7	0.25%	3.13s	7	0.59%	0.78s	2	0.81%	30.72s	6
test_7	500	0.16%	3.02s	1	0.13%	3.13s	2	0.55%	1.15s	0	0.006%	18.51s	10
test_8	600	0.14%	4.39s	3	0.12%	4.94s	1	0.41%	1.96s	0	0.033%	24.18s	9
test_9	800	0.14%	7.55s	2	0.13%	9.90s	2	0.36%	4.32s	2	0.11%	14.77s	8
test_10	1000	0.099%	10.52s	1	0.11%	15.23s	0	0.27%	7.99s	0	0.005%	14.89s	9
Average	/	8.42%	3.96s	36%	8.93%	4.39s	31%	12.16%	1.64s	18%	10.37%	18.25s	68%

For the platform 1, LSWPA algorithm is more effective using θ_1 with $(P3)$. However, its average running time is 59.9 seconds, whereas average running time of LSWPA algorithm using θ_1 with $P2^{'}$ (resp. θ_3 with $\upsilon = 10$) is only 4.33 seconds (resp. 6.58 seconds).

Contrary to platform 1, for the platform 2, LSWPA algorithm is more effective using θ_1 with $(P2^{'})$ and θ_3 with $\upsilon = 10$. Furthermore, its average execution time is only 3.96 seconds, whereas average running time of LSWPA algorithm using $(P3)$ is 18.25 seconds.

LSWPA algorithm is not effective when using the model $(P3^{'})$ for both platforms 1 and 2.

Finally, we get 70% (resp. 68%) of the best solutions by using θ_3 for the platform 1 (resp. 2). What can be said for this part is that the optimal mapping is generally effective, but requires

more running time to find a solution. Thus, in the next part, we will not use the optimal mapping, and we use $P2'$ to compare the two algorithms PLSWPA and LSWPA.

4.5.5 PLSWPA compared to LSWPA and HEFT algorithms

Now we compare the performance of PLSWPA[5] algorithm to LSWPA and HEFT algorithms. We compare the ratio between each makespan value obtained by PLSWPA algorithm with HEFT and LSWPA algorithms, the optimal solution obtained by *CPLEX* and the lower bound $C_{max4'}$ obtained by $(P4')$.

Table 4.14 (resp. 4.16) shows the average results obtained on 10 instances given in column **Inst** of each application size given in the second column using *CPLEX* and HEFT for the platform 1 (resp. 2). Table 4.15 (resp. 4.17) shows the average results of tests of PLSWPA and LSWPA algorithms using the list LLP with the rounding θ_1 and θ_2 with $v = 10$ for the platform 1 (resp. 2).

We show the average time that was needed to *CPLEX* to provide the optimal solution using the model (Opt). We only have the result for the first two instances due to the large running time for instances with more than 60 tasks (> 4 hours).

Then, we show the results obtained by HEFT algorithm, then PLSWPA and LSWPA algorithm using θ_1 and θ_2. Column **GAP** gives the average ratio between the makespan obtained by each method compared to the lower bound $C_{max4'}$ using the following formula:

$$GAP = \frac{\text{method makespan} - C_{max4'}}{C_{max4'}} \times 100.$$

Column **Time** shows the average time that was needed for each method to provide a solution. Column **Best** presents the number of instances where each algorithm provides better or the same solution than other methods. A line Average is added at the end of each table which represents the average values of each column.

Platform 1

Table 4.14: *Optimal solution using CPLEX and HEFT results for platform* 1.

Inst	Number of tasks	Number of ℓ	Number of k	*CPLEX*		HEFT		
				Optimal	Time	Gap	Time	Best
test_1	10	3	3	✓	2.46s	9.65%	0.008s	6
test_2	30	4	4	✓	2m50s	11.64%	0.03s	3
test_3	60	4	4	X	X	8.67%	0.076s	3
test_4	100	6	6	X	X	11.02%	0.20s	0
test_5	200	6	6	X	X	18.51%	0.43s	1
test_6	400	6	6	X	X	14.74%	1.03s	0
test_7	500	8	8	X	X	10.55%	1.90s	0
test_8	600	8	8	X	X	11.55%	2.30s	0
test_9	800	8	8	X	X	7.30%	3.27s	0
test_10	1000	12	12	X	X	9.27%	7.37s	0
Average	/	/	/	/	/	11.29%	1.66s	13%

[5]Polynomial List Scheduling With Pre-Allocation (PLSWPA) algorithm

Table 4.15: *PLSWPA and LSWPA algorithms results.*

Inst	Number of tasks	LSWPA using θ_1 with $P2'$			LSWPA using θ_3 with $\upsilon = 10$			PLSWPA using θ_2			PLSWPA using θ_3 with $\upsilon = 10$		
		GAP	Time	Best	GAP	Time	Best	GAP	Time	Best	GAP	Time	Best
test_1	10	2.15%	0.14s	6	0.08%	0.37s	10	4.36%	0.013s	4	0.08%	0.035s	10
test_2	30	1.93%	1.67s	8	0.54%	1.27s	10	1.37%	0.045s	8	0.94%	0.11s	9
test_3	60	6.48%	1.17s	3	4.08%	1.10s	4	4.34%	0.046s	2	3.37%	0.11s	3
test_4	100	3.51%	2.67s	4	2.46%	2.82s	5	6.69%	0.108s	2	3.99%	0.21s	6
test_5	200	9.64%	5.33s	3	11.10%	10.53s	2	12.06%	0.37s	0	9.95%	0.56s	4
test_6	400	9.20%	19.43s	2	6.74%	24.89s	7	8.80%	1.81s	2	6.06%	2.40s	6
test_7	500	5.45%	21.24s	3	4.21%	31.71s	3	6.67%	7.49s	1	4.91%	8.34s	5
test_8	600	5.58%	18.51s	3	4.93%	23.05s	6	5.71%	5.22s	1	4.71%	6.40s	6
test_9	800	2.02%	13.88s	6	1.07%	16.68s	6	1.93%	10.24s	5	0.99%	11.06s	7
test_10	1000	3.12%	14.82s	3	2.94%	15.69s	4	3.21%	11.69s	2	2.26%	13.51s	4
Average	/	4.90%	9.88s	41%	3.81%	12.45s	57%	5.51%	3.69s	27%	3.72%	4.27s	60%

Platform 2

Table 4.16: *Optimal solution using CPLEX and HEFT results for platform 2.*

Inst	Number of tasks	Number of ℓ	Number of k	*CPLEX* Optimal	*CPLEX* Time	HEFT Gap	HEFT Time	HEFT Best
test_1	10	1	1	✓	0.35s	33.23%	0.0016s	2
test_2	30	1	1	✓	59.58s	38.11%	0.0049s	4
test_3	60	1	1	X	X	25.89%	0.009s	4
test_4	100	1	1	X	X	15.80%	0.017s	0
test_5	200	1	1	X	X	14.56%	0.044s	0
test_6	400	1	1	X	X	11.80%	0.19s	0
test_7	500	1	1	X	X	11.50%	0.26s	0
test_8	600	1	1	X	X	11.53%	0.61s	0
test_9	800	1	1	X	X	11.78%	1.40s	0
test_10	1000	1	1	X	X	12.11%	1.90s	0
Average	/	/	/	/	/	18.63%	0.44s	6%

Table 4.17: *PLSWPA and LSWPA algorithms results.*

Inst	Number of tasks	LSWPA using θ_1 with $P2'$			LSWPA using θ_3 with $\upsilon = 10$			PLSWPA using θ_2			PLSWPA using θ_3 with $\upsilon = 10$		
		GAP	Time	Best	GAP	Time	Best	GAP	Time	Best	GAP	Time	Best
test_1	10	20.84%	0.25s	7	24.46%	0.42s	5	21.24%	0.008s	7	22.38%	0.017s	6
test_2	30	52.34%	0.600s	0	49.08%	0.88s	1	43.19%	0.028s	5	42.5%	0.0780s	3
test_3	60	24.86%	0.29s	6	27.60%	0.37s	4	24.81%	0.081s	7	27.60%	0.22s	4
test_4	100	6.85%	0.198s	8	6.58%	0.53s	9	6.46%	0.184s	9	6.46%	0.543s	9
test_5	200	1.34%	0.51s	7	0.45%	1.87s	8	1.08%	0.68s	6	0.93%	1.06s	8
test_6	400	0.25%	1.72s	7	0.19%	5.76s	9	0.31%	1.22s	6	0.31%	4.01s	6
test_7	500	0.16%	1.84s	6	0.15%	4.72s	7	0.11%	1.13s	9	0.11%	4.10s	9
test_8	600	0.32%	2.06s	6	0.036%	3.48s	7	0.237%	1.09s	7	0.125%	4.79s	7
test_9	800	0.14%	3.15s	7	0.13%	6.59s	8	0.13%	1.09s	6	0.13%	6.62s	7
test_10	1000	0.052%	4.32s	7	0.063%	7.27s	6	0.06%	4.26s	7	0.069%	7.17s	8
Average	/	10.71%	1.49s	61%	10.87%	3.18s	64%	9.76%	1.17s	69%	10.06%	2.96s	67%

For both platforms 1 and 2, HEFT algorithm requires less time than PLSWPA and LSWPA algorithms to provide a solution, where the rounding θ_2 takes more time than θ_1 for both PLSWPA and LSWPA algorithms.

For platform 1, PLSWPA algorithm using θ_3 with $\upsilon = 10$ is the most efficient method with a GAP of 3.72% and 60% of better solutions compared to other methods. Its average running time is 4.26 seconds, which is slightly higher than the running time of PLSWPA algorithm using θ_1 (3.69 seconds) and better than LSWPA algorithm using θ_1 (9.88 seconds).

For platform 2, PLSWPA algorithm using θ_1 is the most efficient method with a GAP of 9.76% and 69% of better solutions compared to other methods. Its average running time is 1.72 seconds, which is slightly better than the running time of LSWPA algorithm using θ_1 (1.49 seconds).

4.5.6 PLSWPA compared to LSWPA and HEFT algorithms using only one processor of type $\mathcal{A}$

In tables 4.18 and 4.19, we compare PLSWPA to LSWPA and HEFT on a platform with only one processor of type $\mathcal{A}$ ($\ell = 1$) and $k > 2$ processors of type $\mathcal{B}$.

Table 4.18: *Optimal solution using CPLEX and HEFT results if $\ell = 1$ and $k > 2$.*

Inst	Number of tasks	Number of ℓ	Number of k	*CPLEX* Optimal	*CPLEX* Time	HEFT Gap	HEFT Time	HEFT Best
test_1	10	1	3	✓	0.895s	19.94%	0.005s	2
test_2	30	1	4	✓	31.93s	19.55%	0.021s	3
test_3	60	1	4	X	X	19.32%	0.044s	1
test_4	100	1	6	X	X	16.22%	0.11s	1
test_5	200	1	6	X	X	13.42%	0.24s	1
test_6	400	1	6	X	X	7.68%	0.60s	0
test_7	500	1	8	X	X	7.42%	1.03s	0
test_8	600	1	8	X	X	7.21%	1.37s	0
test_9	800	1	8	X	X	6.04%	2.29s	0
test_10	1000	1	12	X	X	5.30%	4.29s	0
Average	/	/	/	/	/	12.21%	1s	8%

Table 4.19: *PLSWPA and LSWPA algorithms results.*

Inst	Number of tasks	LSWPA using θ_1 with $P2'$ GAP	LSWPA using θ_1 with $P2'$ Time	LSWPA using θ_1 with $P2'$ Best	LSWPA using θ_3 with $\upsilon = 10$ GAP	LSWPA using θ_3 with $\upsilon = 10$ Time	LSWPA using θ_3 with $\upsilon = 10$ Best	PLSWPA using θ_2 GAP	PLSWPA using θ_2 Time	PLSWPA using θ_2 Best	PLSWPA using θ_3 with $\upsilon = 10$ GAP	PLSWPA using θ_3 with $\upsilon = 10$ Time	PLSWPA using θ_3 with $\upsilon = 10$ Best
test_1	10	3.34%	0.76s	10	4.78%	0.58s	8	3.52%	0.013s	9	3.70%	0.033s	9
test_2	30	10.86%	0.922s	6	9.10%	1.60s	7	15.23%	0.038s	4	14.28%	0.09s	5
test_3	60	14.68%	3.93s	2	14.79%	3.14s	4	30.52%	0.056s	1	25.75%	0.12s	3
test_4	100	20.04%	2.96s	1	20.92%	3.21s	4	18.61%	0.098s	3	14.61%	0.193s	4
test_5	200	10.27%	1.46s	8	9.43%	1.96s	6	10.98%	0.15s	8	9.92%	0.32s	7
test_6	400	0.39%	1.93s	9	0.38%	2.31s	10	0.39%	0.88s	9	0.38%	1.27s	10
test_7	500	0.44%	2.74s	8	0.42%	3.28s	8	0.46%	1.37s	9	0.38%	1.85s	10
test_8	600	0.38%	4.94s	10	0.95%	5.68s	9	1.36%	2.89s	9	0.94%	3.44s	9
test_9	800	0.23%	5.03s	10	0.23%	5.37s	10	0.27%	4.01s	9	0.27%	4.4s	9
test_10	1000	0.31%	13.11s	9	0.24%	14.30s	10	0.58%	11.52s	9	0.36%	12.02s	9
Average	/	6.09%	3.77s	73%	6.12%	4.14s	76%	8.19%	2.09s	70%	7.05%	2.36s	66%

We notice that LSWPA algorithm using θ_1 with $P2'$ is the most efficient method with a GAP of 6.09%. LSWPA algorithm using θ_3 with $\upsilon = 10$ provides 76% of better solutions compared to other methods. However, PLSWPA using θ_2 has the most efficient average running time with 2.09 seconds.

4.5.7 PLSWPA compared to LSWPA and HEFT algorithms with consistent model

In tables 4.20 and 4.21, we compare PLSWPA to LSWPA and HEFT on platforms where the processors of type $\mathcal{A}$ are faster than the processors of type $\mathcal{B}$.

Table 4.20: *Optimal solution using CPLEX and HEFT results if the processors of type $\mathcal{A}$ are faster than the processors of type $\mathcal{B}$.*

Inst	Number of tasks	Number of ℓ	Number of k	*CPLEX*		HEFT		
				Optimal	Time	Gap	Time	Best
test_1	10	3	3	✓	0.32s	0%	0.008s	10
test_2	30	4	4	✓	20.62s	0.12%	0.034s	9
test_3	60	4	4	X	X	6.21%	0.07s	4
test_4	100	6	6	X	X	3.48%	0.20s	6
test_5	200	6	6	X	X	9.73%	0.42s	7
test_6	400	6	6	X	X	6.03%	0.95s	3
test_7	500	8	8	X	X	6.69%	1.66s	0
test_8	600	8	8	X	X	6.88%	2.16s	0
test_9	800	8	8	X	X	6.48%	3.29s	0
test_10	1000	12	12	X	X	6.88%	6.40s	0
Average	/	/	/	/	/	5.25%	1.51s	39%

Table 4.21: *PLSWPA and LSWPA algorithms results.*

Inst	Number of tasks	LSWPA						PLSWPA					
		using θ_1 with $P2'$			using θ_3 with $\upsilon = 10$			using θ_2			using θ_3 with $\upsilon = 10$		
		GAP	Time	Best	GAP	Time	Best	GAP	Time	Best	GAP	Time	Best
test_1	10	0%	0.028s	10	0%	0.071s	10	2.54%	0.010s	9	0%	0.03s	10
test_2	30	1.94%	0.071s	7	0%	0.16s	10	2.35%	0.04s	5	0.44%	0.10s	7
test_3	60	13.55%	0.34s	2	10.1%	0.54s	3	16.32%	0.046s	1	9.51%	0.10s	5
test_4	100	5.68%	0.47s	3	4.57%	0.49s	3	12.48%	0.088s	0	6.29%	0.18s	3
test_5	200	20.33%	7.56s	0	16.15%	6.54s	1	19.01%	0.19s	1	15.31%	0.37s	2
test_6	400	4.04%	2.34s	6	3.65%	2.87s	6	3.99%	0.85s	5	3.58%	1.21s	6
test_7	500	5.34%	3.60s	8	5.24%	4.33s	7	3.52%	2.19s	9	3.51%	2.63s	9
test_8	600	2.55%	5.55s	10	2.69%	6.18s	8	3.27%	3.36s	6	3.23%	3.90s	6
test_9	800	1.15%	5.95s	9	1.15%	6.7s	9	1.10%	4.62s	10	1.15%	4.95s	9
test_10	1000	2.35%	15.9s	8	2.32%	17.62s	9	2.34%	14.36s	8	2.34%	14.82s	8
Average	/	5.69%	4.15s	63%	4.58%	4.42s	66%	6.69%	2.57s	54%	4.53%	2.82s	65%

We notice that PLSWPA algorithm using θ_3 with $\upsilon = 10$ is the most efficient method with a GAP of 4.53%. LSWPA algorithm using θ_3 with $\upsilon = 10$ provides 66% of better solutions compared to other methods. However, PLSWPA using θ_2 has the most efficient average running time with 2.57 seconds.

4.5.8 Comparison between PLSWPA, LSWPA and HEFT algorithms

In table 4.22, we give the average of the tables 4.14, 4.15, 4.16, 4.17, 4.18, 4.19, 4.20 and 4.21.

Table 4.22: *PLSWPA algorithm compared to LSWPA and HEFT algorithms.*

Inst	HEFT			LSWPA						PLSWPA					
				using θ_1 with $P2'$			using θ_3 with $\upsilon = 10$			using θ_2			using θ_3 with $\upsilon = 10$		
	GAP	Time	Best	GAP	Time	Best	GAP	Time	Best	GAP	Time	Best	GAP	Time	Best
Tables 4.14 and 4.15	11.29%	1.66s	13%	4.90%	9.88s	41%	3.81%	12.45s	57%	5.51%	3.69s	27%	3.72%	4.27s	60%
Tables 4.16 and 4.17	18.63%	0.44s	6%	10.71%	1.49s	61%	10.87%	3.18s	64%	9.76%	1.17s	69%	10.06%	2.96s	67%
Tables 4.18 and 4.19	12.21%	1s	8%	6.09%	3.77s	73%	6.12%	4.14s	76%	8.19%	2.09s	70%	7.05%	2.36s	66%
Tables 4.20 and 4.21	5.25%	1.51s	39%	5.69%	4.15s	63%	4.58%	4.42s	66%	6.69%	2.57s	54%	4.53%	2.82s	65%
Average	11.84 %	1.15 s	4.12 %	6.84 %	4.82 s	59.5%	6.34 %	6.04 s	65.75 %	7.53 %	2.38s	55 %	6.34 %	3.10 s	64.5 %

From Table 4.22, it can be said that both methods LSWPA and PLSWPA are effective methods with a general GAP smaller than 8 compared to the lower bound, and thus to the optimal solution. However, PLSWPA algorithm has the advantage of being a polynomial time method, and can handle large instances with an interesting running time.

4.6 Conclusion

This chapter presents two efficient algorithms to solve the problem of scheduling parallel applications on hybrid platforms with communication delays. The objective is to minimize the total execution time (makespan).

First, we proposed a non polynomial 6-approximation algorithm LSWPA with two phases: mapping then tasks assignment. Two models and two rounding strategies have been proposed for the mapping. In the second phase, a list scheduling algorithm has been proposed to generate a feasible schedule using several lists. LSWPA algorithm guarantees a ratio of 6 compared to the optimal solution using the rounding strategy θ_1. This method was published in [107].

Then we proposed a polynomial three-phase algorithm PLSWPA: the first two phases consist in solving linear models to find the type of processor assigned to execute each task. In the third phase, we compute the start execution time of each task to generate a feasible schedule.

Tests on large instances close to reality demonstrated the efficiency of our methods and shows the limits of solving the problem with a solver such as *CPLEX*.

A proof of the performance guarantee for PLSWPA algorithm was initiated. The ratio between the solution $\widehat{C}_{max1}$ obtained by PLSWPA algorithm and the optimal scheduling solution $C^{\star}_{max}$ is given by $\widehat{C}_{max1} < 6\alpha C^{\star}_{max}$. In future works, we will focus on finding the value of α to have a fixed bound on the ratio between $\widehat{C}_{max1}$ and $C^{\star}_{max}$.

As part of the future, it will be interesting to study the tightness of LSWPA and PLSWPA algorithms using rounding θ_3 which provide interesting solutions.

An extension to more general heterogeneous platforms with more than two types of processors is also interesting. The challenge is to find a good adaptation of the pre-allocation policy if we have more than two types of processing elements.

Due to the significant energy consumption of this kind of platforms, we will study in the following chapter the possibility of extending LSWPA and PLSWPA algorithms to solve the problem with an energy constraint.

Chapter

5 Hybrid platform with a limited number of processors with energy constraint

Chapter content

5.1 Introduction

The most common objective function of task scheduling problems is minimizing the total execution time (makespan). However, energy consumption is also an important issue to be considered. Indeed, the resulting energy consumption of these platforms is very high and its increase must be kept reasonable. Thus, it is time to invest in green computing, and computing servers must be built with energy-aware resource management. This focus on energy efficiency must also have as much as possible little impact on performance as possible. Thus, heterogeneous systems offer the opportunity to achieve high performance while saving energy which offers several execution possibilities with different cost consumption (execution time, energy...).

In the previous chapter, we have studied the problem of scheduling parallel applications presented with graphs of type DAG on hybrid platforms composed of a limited number of two types of processors denoted by $\mathcal{A}$ and $\mathcal{B}$. The number of processing elements of type $\mathcal{A}$ (resp. $\mathcal{B}$) is given by ℓ (resp. k). The objective is to minimize the total execution time (makespan) respecting precedence constraints with communication delays. The objective of this chapter is to propose a generic approach in order to minimize both makespan and energy consumption. For this purpose, we introduce a constraint on the total energy consumed by the platform. Executing

the task t_i on a processing element of type $\mathcal{A}$ (resp. $\mathcal{B}$) generates an energy consumption equal to $e_{i,\mathcal{A}}$ (resp. $e_{i,\mathcal{B}}$). We denote by $\mathbb{E}$ the allowed quantity of energy consumed during the execution. $\mathbb{E}$ represents in our case an energy bound that should not be exceeded during the execution. The minimum amount of energy to execute each application is calculated as $\mathbb{E}_{min} = \sum_{i=1}^{n} min\{e_{i,\mathcal{A}}, e_{i,\mathcal{B}}\}$. The maximum amount of energy to execute each application is calculated as $\mathbb{E}_{max} = \sum_{i=1}^{n} max\{e_{i,\mathcal{A}}, e_{i,\mathcal{B}}\}$. Choosing $\mathbb{E}$ between $\mathbb{E}_{min}$ and $\mathbb{E}_{max}$ is a sufficient condition for the existence of a solution. However, the aim is to use less energy and get a good solution. Thus, $\mathbb{E}$ can be varied to find a good compromise between makespan and energy consumption. The aim is then to propose efficient scheduling methods to minimize execution time while respecting an energy constraint $((P\ell, Pk)|prec, com, \mathbb{E}|C_{max})$. In addition, these methods must guarantee the quality of the solution in the worst case compared to the optimal solution.

Contrary to the problem without energy constraint, sometimes it is not possible to execute a task on its favorite processing element because of the limited available energy. Thus, it is important to properly manage the energy consumption and the number of each type of processing elements to exploit the parallelism effectively. For this purpose, we modify the two algorithms LSWPA and PLSWPA proposed in the previous chapter. We use the same technique to solve the problem with energy constraint by adding new constraints. Two algorithms are proposed: a non polynomial-time algorithm with a performance ratio of 6 and a polynomial time algorithm with a relative performance guarantee for makespan, but with at most twice authorized energy consumption. Both methods are based on a new scheduling technique, based on the pre-allocation of tasks before scheduling phase. This technique allows us to have an extended view of the entire graph that represents the application. We thus make the best decisions that arrange the maximum number of tasks.

The rest of this chapter is organized as follows. After analysing the complexity of our problem in Section 5.2, we present in Section 5.3 the mathematical model used to obtain the optimal solution for small instances. Then, we detail in Sections 5.4 and 5.5 the two proposed scheduling algorithms with a performance guaranty analysis. Finally, after testing the proposed algorithm on several instances in Section 5.6, we provide concluding remarks and future directions in Section 5.7.

5.2 Complexity

We prove by the following theorem that our problem is NP-Complete even for the particular case linear chain of tasks without communication delays. For this purpose, we perform a reduction from the well-known NP-Complete problem Knapsack [57, 109] in polynomial time. In the knapsack problem, we are given a set I of n items $I = \{i_1, ..., i_n\}$, where each item i has a value $v_i \in \mathbb{N}^*$ and a size $s_i \in \mathbb{N}^*$. The knapsack has capacity $B \in \mathbb{N}^*$. The goal is to find a subset of items $I^{'} \subseteq I$ that maximizes the value $\sum_{i \in I'} v_i$ of items in the knapsack such that $\sum_{i \in I'} s_i \leqslant B$. For $C_2 \in \mathbb{Q}^+$, the decision problem $KPS(I, B, C_2)$ associated with the knapsack problem is as follows. Is there a subset of items $I^{'} \subseteq I$ such that $\sum_{i \in I'} s_i \leqslant B$ and $\sum_{i \in I'} v_i \geqslant C_2$?.

Theorem 10. *Let $G(V, E)$ be a linear chain graph and HP a Hybrid Platform. For $C_1 \in \mathbb{Q}^+$, the decision problem $SCH(G, HP, C_1)$ associated with the scheduling problem is as follows. Is there a schedule S for $G(V, E)$ on HP respecting an energy constraint $\mathbb{E}$ with a makespan $C_{max} \leqslant C_1$?. $SCH(G, HP, C_1)$ in NP-Complete.*

Proof.

1. First, for any given solution S of $SCH(G, HP, C_1)$, it can be verified in polynomial time that S is feasible and $C_{max} \leqslant C_1$. Indeed, energy, precedence and no-overlapping constraints can be checked in polynomial time. Hence, $SCH(G, HP, C_1) \in NP$.

2. Next, to prove that $SCH(G, HP, C_1)$ is NP-Complete, we reduce the well-known NP-Complete problem Knapsack in polynomial time to $SCH(G, HP, C_1)$. For an arbitrary instance of knapsack $Inst\{v_1, \cdots, v_n, s_1, \cdots, s_n, B\}$, an instance of $SCH(G, HP, C_1)$ is constructed in the following way. A linear chain of tasks graph $G(V, E)$ is constructed as shown in Figure 5.1. It consists of n successive tasks, with a precedence constraint between each two tasks t_i and t_{i+1} without communication delays $cm_{i,i+1} = 0$, $i = \overline{1..n-1}$. Execution time of the task t_i on a processing element of type $\mathcal{A}$ (resp. $\mathcal{B}$) is given by $w_{i,\mathcal{A}}$ (resp. $w_{i,\mathcal{B}}$) and its energy consumption on a processing element of type $\mathcal{A}$ (resp. $\mathcal{B}$) is given by $e_{i,\mathcal{A}}$ (resp. $e_{i,\mathcal{B}}$). We denote by $\mathbb{E}$ the allowed quantity of energy consumed during the execution.

   For each task t_i, we set $w_{i,\mathcal{B}} = 1$ and $w_{i,\mathcal{A}} = v_i + w_{i,\mathcal{B}}$, where $w_{i,\mathcal{A}} > w_{i,\mathcal{B}}$. Hence, $v_i = w_{i,\mathcal{A}} - w_{i,\mathcal{B}}$, $i = \overline{1..n}$. We also set $e_{i,\mathcal{A}} = 0$ and $e_{i,\mathcal{B}} = s_i$, $i = \overline{1..n}$; $\mathbb{E} = B$ and $C_1 = \sum_{i=1}^{n} w_{i,\mathcal{A}} - C_2$. Clearly, the construction of the instance of $SCH(G, HP, C_1)$ is polynomial in the size of the instance of Knapsack.

   Let $C'_{max} = \sum_{i=1}^{n} w_{i,\mathcal{A}}$ be an initial scheduling length (worst case) with a total energy consumption equal to $\sum_{i=1}^{n} e_{i,\mathcal{A}} = 0$. Thus, $C_1 = C'_{max} - C_2$. We want to find a subset $T' \subseteq T$ such that $\sum_{i \in T'} e_{i,\mathcal{B}} \leqslant \mathbb{E}$ and $C_{max} \leqslant C_1$, then $\sum_{i \in I'} s_i \leqslant B$ and $\sum_{i \in I'} v_i \geqslant C_2$ for $I' = T'$. Follows, $C_{max} = \sum_{i \in T'}^{n} w_{i,\mathcal{B}} + \sum_{i \in T \setminus T'}^{n} w_{i,\mathcal{A}} = \sum_{i \in T'}^{n} w_{i,\mathcal{B}} + (\sum_{i=1}^{n} w_{i,\mathcal{A}} - \sum_{i \in T'}^{n} w_{i,\mathcal{A}}) = \sum_{i=1}^{n} w_{i,\mathcal{A}} - \sum_{i \in T'} (w_{i,\mathcal{A}} - w_{i,\mathcal{B}}) = C'_{max} - \sum_{i \in T'} (w_{i,\mathcal{A}} - w_{i,\mathcal{B}}) \leqslant C_1$, then $\sum_{i \in T'} (w_{i,\mathcal{A}} - w_{i,\mathcal{B}}) \geqslant C'_{max} - C_1$. Hence, since $v_i = w_{i,\mathcal{A}} - w_{i,\mathcal{B}}$ and $C_2 = C'_{max} - C_1$, $\sum_{i \in T'} v_i \geqslant C_2$. Furthermore, since $e_{i,\mathcal{B}} = s_i$ for $i = \overline{1..n}$, $\sum_{i \in T'} e_{i,\mathcal{B}} \leqslant \mathbb{E}$ and $\mathbb{E} = B$, we obtain $\sum_{i \in T'} s_i \leqslant B$. Thus, for $I' = T'$, a solution obtained for the scheduling problem corresponds to a solution of the knapsack problem.

   Conversely, for $I' \subseteq I$, $\sum_{i \in I'} s_i \leqslant B$ and $\sum_{i \in I'} v_i \geqslant C_2$. Hence, since $\sum_{i \in I'} v_i = \sum_{i \in I'} (w_{i,\mathcal{A}} - w_{i,\mathcal{B}})$ and $C_2 = \sum_{i=1}^{n} w_{i,\mathcal{A}} - C_1$, we obtain $\sum_{i \in I'} (w_{i,\mathcal{A}} - w_{i,\mathcal{B}}) \geqslant \sum_{i=1}^{n} w_{i,\mathcal{A}} - C_1$. Follows, $C_1 \geqslant \sum_{i=1}^{n} w_{i,\mathcal{A}} - \sum_{i \in I'} (w_{i,\mathcal{A}} - w_{i,\mathcal{B}})$. Finally, for $T' = I'$, we have $C_{max} = \sum_{i=1}^{n} w_{i,\mathcal{A}} - \sum_{i \in T'} (w_{i,\mathcal{A}} - w_{i,\mathcal{B}}) \leqslant C_1$. Furthermore, $\sum_{i \in T'} e_{i,\mathcal{B}} \leqslant \mathbb{E}$, since $\sum_{i \in I'} s_i \leqslant B$, $\mathbb{E} = B$ and $e_{i,\mathcal{B}} = s_i$ for $i = \overline{1..n}$. Thus, a solution obtained for the knapsack problem corresponds to a solution of the scheduling problem.

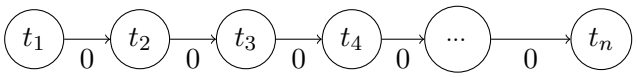


Figure 5.1: *Linear chain of tasks.*

□

Proposition 8. *The execution of a task t_i generates an energy consumption equal to at least the minimum between $e_{i,\mathcal{A}}$ and $e_{i,\mathcal{B}}$. Thus, we can transform the energy data of an instance into $\mathbb{E}' = \mathbb{E} - \sum_{i=1}^{n} minimum\{e_{i,\mathcal{A}}, e_{i,\mathcal{B}}\}$, $e'_{i,\mathcal{A}} = e_{i,\mathcal{A}} - minimum\{e_{i,\mathcal{A}}, e_{i,\mathcal{B}}\}$ and $e'_{i,\mathcal{B}} = e_{i,\mathcal{B}} - minimum\{e_{i,\mathcal{A}}, e_{i,\mathcal{B}}\}$ for $i = \overline{1..n}$. Therefore, for each task t_i, $e'_{i,\mathcal{A}} = 0$ or $e'_{i,\mathcal{B}} = 0$, $i = \overline{1..n}$.*

In the following, we will use this simplification to handle the energy constraint. In the next section, we present the mathematical model used to obtain the optimal solution for small instances (small number of tasks and processors).

5.3 Mathematical model

This section aims at providing a modelling of the scheduling problem using a Mixed Integer Program (MIP) for which the numbers of variables and constraints are polynomial. By adding a

new constraint to the model (Opt) defined in the previous chapter in section 4.2, we obtain the model ($Opte$) which takes into account the energy constraint.

Data:

A hybrid platform is composed of m resources $P = \{p_1, p_2, \cdots, p_m\}$ of two types: $\mathcal{A}$ and $\mathcal{B}$ (e.g. GPU+CPU). ℓ represents the number of processing elements of type $\mathcal{A}$ and k the number of processing elements of type $\mathcal{B}$, $m = \ell + k$. Let $P(\mathcal{A})$ (resp. $P(\mathcal{B})$) be the set of processors type $\mathcal{A}$ (resp. B). Clearly, $P(\mathcal{A}) = \{p_1, p_2, \cdots, p_\ell\}$ and $P(\mathcal{B}) = \{p_{\ell+1}, p_{\ell+2}, \cdots, p_m\}$. We denote by $\tau(r)$ the type of the processing element p_r, such that $\tau(r) = \mathcal{A}$ if $r \leqslant \ell$ and $\tau(r) = \mathcal{B}$ if $r > \ell$. $\mathbb{E}^{'}$ represents the allowed energy consumption. We also use the two following data:

- $w_{i,\mathcal{A}}$ (resp. $w_{i,\mathcal{B}}$) is the execution time of t_i on a processor of type $\mathcal{A}$ (resp. $\mathcal{B}$).
- $e^{'}_{i,\mathcal{A}}$ (resp. $e^{'}_{i,\mathcal{B}}$) is the energy consumption of t_i on a processor of type $\mathcal{A}$ (resp. $\mathcal{B}$).
- $cm_{i,j}$ represents the communication delays between t_i and t_j if they are executed on two different types of processing elements.

Variables:

Let us consider the following decision variables:

- $x_{i,r}$ equal to 1 if the task t_i is assigned to the processor p_r, 0 otherwise, $i = \overline{1..n}$ and $r = \overline{1..m}$.
- S_i is the starting time of the task t_i, $i = \overline{1..n}$.
- $o_{i,j}$ is an intermediary variable to manage overlapping tasks on the same processing element. For each two tasks $t_i \neq t_j$, $i = \overline{1..n}$ and $j = \overline{1..n}$, if t_i and t_j are executed by the same processing element, then:

$$o_{i,j} = \begin{cases} 1 & \text{if } t_j \text{ is executed after the completion time of } t_i \\ 0 & \text{if } t_i \text{ is executed after the completion time of } t_j \end{cases}$$

Model:

We obtain the following model ($Opte$) composed of 5 constraints:

$$(Opte) \begin{cases} \sum_{r=1}^{m} x_{i,r} = 1, \forall i = \overline{1..n} & (1) \\ S_i + x_{i,r} w_{i,\tau(r)} + (x_{i,r} + x_{j,u} - 1) cm_{i,j} \leqslant S_j & (2) \\ \forall\ (t_i, t_j) \in E,\ \ \forall r = \overline{1..m},\ \ \forall u = \overline{1..m},\ \tau(r) \neq \tau(u) & \\ S_i + x_{i,r} w_{i,\tau(r)} \leqslant S_j + B \times (3 - x_{ir} - x_{jr} - o_{i,j}) & (3) \\ S_j + x_{j,r} w_{j,\tau(r)} \leqslant S_i + B \times (2 - x_{ir} - x_{jr} + o_{i,j}) & \\ \forall i, j = \overline{1..n} \quad i \neq j,\ (t_i, t_j) \notin E,\ \forall r = \overline{1..m}, B = Cte & \\ S_i + \sum_{r=1}^{m} x_{i,r} w_{i,\tau(r)} \leqslant C_{max}, \forall i, \Gamma^{+}(i) = \emptyset & (4) \\ \sum_{i=1}^{n} \sum_{r=1}^{m} x_{i,r} e^{'}_{i,\tau(r)} \leqslant \mathbb{E}^{'} & (5) \\ x_{i,r} \text{ and } o_{i,j} \in \{0, 1\},\ \forall i, j = \overline{1..n},\ r = \overline{1..m} & \\ Z(min) = C_{max} & \end{cases}$$

In addition to the four constraints ((1), (2), (3), (4)) explained in Section 4.2, we add only the energy constraint (5), such that the energy consumption of all tasks does not exceed $\mathbb{E}^{'}$. Solvers like *CPLEX* can be used to find the optimal solution for small instances. However, in a practical way, generic approaches are used to solve large instances with a reasonable running time. In the following, we propose two algorithms: a non polynomial-time algorithm (LSWPAe) with a performance ratio of 6 and a polynomial time algorithm (PLSWPAe) with a relative performance guarantee.

5.4 List Scheduling algorithm With Pre-Allocation (LSWPAe)

In the previous chapter, we proposed the LSWPA algorithm for the scheduling problem, with the objective of minimizing the makespan without considering an energy constraint.

In this section, we modify this algorithm to solve the problem with an energy constraint. For this purpose, we modify only the first phase, which is the task allocation phase, adding some constraints. The goal is therefore to find a task assignment so that the overall energy consumption respects the constraint $\mathbb{E}'$. We will then show that we keep the result of the performance guarantee of 6 for the problem of scheduling with an energy constraint.

New constraints will be added to the two models $(P1)$ and $(P2)$ introduced in the section 4.4.1 of the previous chapter. We will then obtain the LSWPAe algorithm which also uses two steps as LSWPA algorithm to keep the same ratio of 6 for the scheduling with communication costs and an energy constraint. The first step consists of solving an assignment problem to find which type of processor ($\mathcal{A}$ or $\mathcal{B}$) will execute each task. Two models $(P1e)$ and $(P2e)$ are then illustrated in the first phase for solving the assignment problem while the precedence constraints are satisfied and the energy constraint is respected. The solution obtained by the model $(P1e)$ or $(P2e)$ represents a lower bound for the final makespan. Then, after solving the relaxation $(P1e^{'})$ (resp. $(P2e^{'})$) of the model $(P1e)$ (resp. $(P2e)$), the fractional solution is rounded to obtain a feasible assignment of the tasks. In the second phase, the assignment of the tasks and a list scheduling algorithm are used to get a feasible schedule. In the following, the two phases are presented in detail.

I)Phase 1: assignment of tasks

By adding new constraints and new variables to the models $(P1)$ and $(P2)$, we propose two new models $(P1e)$ and $(P2e)$ to solve the assignment problem while the precedence constraints and the energy constraint are satisfied. Then we solve the relaxation $(P1e^{'})$ (resp. $(P2e^{'})$) of the model $(P1e)$ (resp. $(P2e)$). We get the model $(P1e)$ by adding the constraints $(1e)$, $(2e)$ and $(3e)$ to the model $(P1)$ using two new intermediary variables μ_i and υ_i to handle the energy bound, $i = \overline{1..n}$. The new model $(P1e)$ is then defined as follows.

$$
(P1e)\begin{cases}
C_i + x_j w_{j,\mathcal{A}} + (1-x_j)w_{j,\mathcal{B}} + (1-|y_{i,j}-z_{i,j}|)cm_{i,j} \leqslant C_j,\ \forall (t_i,t_j) \in E & (1)\\
z_{i,j} \leqslant x_i, \forall (t_i,t_j) \in E & (2)\\
z_{i,j} \leqslant x_j, \forall (t_i,t_j) \in E & (3)\\
y_{i,j} \leqslant 1-x_i, \forall (t_i,t_j) \in E & (4)\\
y_{i,j} \leqslant 1-x_j, \forall (t_i,t_j) \in E & (5)\\
x_i w_{i,\mathcal{A}} + (1-x_i)w_{i,\mathcal{B}} \leqslant C_i, \forall i = \overline{1..n}, \Gamma^-(i) = \emptyset & (6)\\
0 \leqslant C_i \leqslant C_{max}, \forall i = \overline{1..n}, \Gamma^+(i) = \emptyset & (7)\\
\sum_{i=1}^{n} x_i w_{i,\mathcal{A}} \leqslant \ell C_{max} & (8)\\
\sum_{i=1}^{n} (1-x_i) w_{i,\mathcal{B}} \leqslant k C_{max} & (9)\\
\boldsymbol{x_i \geqslant \mu_i, \forall\ i = \overline{1..n}} & \boldsymbol{(1e)}\\
\boldsymbol{(1-x_i) \geqslant \upsilon_i, \forall\ i = \overline{1..n}} & \boldsymbol{(2e)}\\
\boldsymbol{\sum_{i=1}^{n} ((x_i + (1-|\mu_i - \upsilon_i|))e'_{i,\mathcal{A}} + ((1-x_i) + (1-|\mu_i-\upsilon_i|)))e'_{i,\mathcal{B}}) \leqslant \mathbb{E}'} & \boldsymbol{(3e)}\\
x_i, \mu_i, \upsilon_i, y_{i,j}, z_{i,j} \in \{0,1\},\ \forall i = \overline{1..n}, j = \overline{1..n} &\\
Z(min) = C_{max} &
\end{cases}
$$

Let $\mathbb{E}'_i = (x_i + (1-|\mu_i - \upsilon_i|))e'_{i,\mathcal{A}} + ((1-x_i) + (1-|\mu_i-\upsilon_i|))e'_{i,\mathcal{B}}$ be the energy consumption of each task t_i in the model $(P1e)$, $i = \overline{1..n}$. We prove by the following Lemma 18 that the solution obtained by solving the model $(P1e)$ respects the energy constraint.

Lemma 18. *The solution obtained by solving the model* $(P1e)$ *respects the energy constraint, i.e.,* $\sum_{i=1}^{n} \mathbb{E}_i^{'} \leqslant \mathbb{E}^{'}$.

Proof. For each task t_i, we obtain $\mathbb{E}_i^{'} > 0$ in two cases:

1. If $e_{i,\mathcal{A}}^{'} = 0$ then $e_{i,\mathcal{B}}^{'} \geqslant 0$, two cases are possible: if $x_i = 0$, then $0 \geqslant \mu_i = 0$. Follow, $\mathbb{E}_i^{'} = (1 + (1 - \upsilon_i))e_{i,\mathcal{B}}^{'}$. Finally, since it is a minimization problem and $1 \geqslant \upsilon_i$ (constraint $(2e)$), $\mathbb{E}_i^{'} = e_{i,\mathcal{B}}^{'}$ for $\upsilon_i = 1$. If $x_i = 1$, then $0 \geqslant \upsilon_i = 0$. Follow, $\mathbb{E}_i^{'} = (1 - \mu_i)e_{i,\mathcal{B}}^{'}$. Finally, since $1 \geqslant \mu_i$ (constraint $(1e)$), $\mathbb{E}_i^{'} = 0$ for $\mu_i = 1$. In the two cases, $\mathbb{E}_i^{'} \leqslant e_{i,\mathcal{B}}^{'}$.

2. If $e_{i,\mathcal{B}}^{'} = 0$ then $e_{i,\mathcal{A}}^{'} \geqslant 0$, two cases are possible: if $x_i = 0$, then $0 \geqslant \mu_i = 0$. Follow, $\mathbb{E}_i^{'} = (1 - \upsilon_i)e_{i,\mathcal{A}}^{'}$. Finally, since it is a minimization problem and $1 \geqslant \upsilon_i$, $\mathbb{E}_i^{'} = 0$ for $\upsilon_i = 1$. If $x_i = 1$, then $0 \geqslant \upsilon_i = 0$. Follow, $\mathbb{E}_i^{'} = (1 + (1 - \mu_i))e_{i,\mathcal{A}}^{'}$. Finally, since $1 \geqslant \mu_i$, $\mathbb{E}_i^{'} = e_{i,\mathcal{A}}^{'}$ for $\mu_i = 1$. In the two cases, $\mathbb{E}_i^{'} \leqslant e_{i,\mathcal{A}}^{'}$.

Thus, for each task t_i, $\mathbb{E}_i^{'} \leqslant e_{i,\mathcal{A}}^{'} + e_{i,\mathcal{B}}^{'}$ (since $e_{i,\mathcal{A}}^{'} = 0$ or $e_{i,\mathcal{B}}^{'} = 0$). Finally, from the constraint $(3e)$ of $(P1e)$, the total energy consumption is given by $\sum_{i=1}^{n} \mathbb{E}_i^{'} \leqslant \sum_{i=1}^{n} e_{i,\mathcal{A}}^{'} + e_{i,\mathcal{B}}^{'} \leqslant \mathbb{E}^{'}$. □

Constraint $(3e)$ of the model $(P1e)$ contains an absolute value which we can get rid by adding two binary variables, σ_i and τ_i. Then, we get a linear model $(P2e)$ by adding the following constraints $(1e)$, $(2e)$, $(3e)$, $(4e)$ and $(5e)$ to the model $(P2)$. The new model $(P2e)$ is then defined as follows.

$$
(P2e)\begin{cases}
C_i + x_j w_{j,\mathcal{A}} + (1 - x_j)w_{j,\mathcal{B}} + \boldsymbol{(1 - b_{i,j})} cm_{i,j} \leqslant C_j,\ \forall (t_i, t_j) \in E & (1)\\
z_{i,j} \leqslant x_i, \forall (t_i, t_j) \in E & (2)\\
z_{i,j} \leqslant x_j, \forall (t_i, t_j) \in E & (3)\\
y_{i,j} \leqslant 1 - x_i, \forall (t_i, t_j) \in E & (4)\\
y_{i,j} \leqslant 1 - x_j, \forall (t_i, t_j) \in E & (5)\\
\boldsymbol{(z_{i,j} - y_{i,j}) + 2(1 - a_{i,j}) \geqslant b_{i,j}, \forall (t_i, t_j) \in E} & \boldsymbol{(5.1)}\\
\boldsymbol{(y_{i,j} - z_{i,j}) + 2a_{i,j} \geqslant b_{i,j}, \forall (t_i, t_j) \in E} & \boldsymbol{(5.2)}\\
x_i w_{i,\mathcal{A}} + (1 - x_i)w_{i,\mathcal{B}} \leqslant C_i, \forall i = \overline{1..n}, \Gamma^{-}(i) = \emptyset & (6)\\
0 \leqslant C_i \leqslant C_{max}, \forall i = \overline{1..n}, \Gamma^{+}(i) = \emptyset & (7)\\
\sum_{i=1}^{n} x_i w_{i,\mathcal{A}} \leqslant \ell C_{max} & (8)\\
\sum_{i=1}^{n} (1 - x_i) w_{i,\mathcal{B}} \leqslant k C_{max} & (9)\\
\boldsymbol{x_i \geqslant \mu_i, \forall\ i = \overline{1..n}} & \boldsymbol{(1e)}\\
\boldsymbol{(1 - x_i) \geqslant \upsilon_i, \forall\ i = \overline{1..n}} & \boldsymbol{(2e)}\\
\boldsymbol{(\upsilon_i - \mu_i) + 2(1 - \sigma_i) \geqslant \tau_i, \forall\ i = \overline{1..n}} & \boldsymbol{(3e)}\\
\boldsymbol{(\mu_i - \upsilon_i) + 2\sigma_i \geqslant \tau_i, \forall\ i = \overline{1..n}} & \boldsymbol{(4e)}\\
\boldsymbol{\sum_{i=1}^{n} ((x_i + (1 - \tau_i))e_{i,\mathcal{A}}^{'} + ((1 - x_i) + (1 - \tau_i))e_{i,\mathcal{B}}^{'}) \leqslant \mathbb{E}^{'}} & \boldsymbol{(5e)}\\
x_i, \mu_i, \upsilon_i, \sigma_i, \tau_i, y_{i,j}, z_{i,j}, a_{i,j}, b_{i,j} \in \{0, 1\},\ \forall i = \overline{1..n}, j = \overline{1..n} & \\
Z(min) = C_{max} &
\end{cases}
$$

We proved in Section 4.4.1 that the two models $(P1)$ and $(P2)$ are equivalent. By adding the constraints $(c1)$(resp. $(c2)$) to $(P1)$ (resp. $(P2)$), we obtain the model $(P1e)$ (resp. $(P2e)$) which handles the energy constraint.

$$
(c1)\begin{cases}
x_i \geqslant \mu_i, \forall\ i = \overline{1..n} & (1e)\\
(1 - x_i) \geqslant \upsilon_i, \forall\ i = \overline{1..n} & (2e)\\
\sum_{i=1}^{n} ((x_i + (1 - |\mu_i - \upsilon_i|))e_{i,\mathcal{A}}^{'} + ((1 - x_i) + (1 - |\mu_i - \upsilon_i|))e_{i,\mathcal{B}}^{'}) \leqslant \mathbb{E}^{'} & (3e)\\
x_i, \mu_i, \upsilon_i \in \{0, 1\}\ \forall\ i = \overline{1..n} &
\end{cases}
$$

$$(c2)\begin{cases} x_i \geqslant \mu_i, \forall\ i = \overline{1..n} & (1e) \\ (1-x_i) \geqslant \upsilon_i, \forall\ i = \overline{1..n} & (2e) \\ (\upsilon_i - \mu_i) + 2(1-\sigma_i) \geqslant \tau_i, \forall\ i = \overline{1..n} & (3e) \\ (\mu_i - \upsilon_i) + 2\sigma_i \geqslant \tau_i, \forall\ i = \overline{1..n} & (4e) \\ \sum_{i=1}^{n}((x_i + (1-\tau_i))e'_{i,\mathcal{A}} + ((1-x_i) + (1-\tau_i))e'_{i,\mathcal{B}}) \leqslant \mathbb{E}' & (5e) \\ x_i, \mu_i, \upsilon_i, \sigma_i, \tau_i \in \{0,1\}\ \forall\ i = \overline{1..n} \end{cases}$$

We prove by the following Lemma 19 that the constraints $(c1)$ and $(c2)$ are equivalent, and thus $(P1e)$ and $(P2e)$ are also equivalent.

Lemma 19. *The constraints* $(c1)$ *and* $(c2)$ *are equivalent.*

Proof. For each task t_i, the value of τ_i replaces the value of $|\mu_i - \upsilon_i|$ in $(P2e)$, we obtain two cases:

1. If $|\mu_i - \upsilon_i| = 0$, then, since $2\sigma_i \geqslant \tau_i$ and $2(1-\sigma_i) \geqslant \tau_i$ (constraints $(3e)$ and $(4e)$ from $(c2)$), then $\tau_i = 0$ for $\sigma_i \in \{0,1\}$.

2. If $|\mu_i - \upsilon_i| = 1$ then we have two cases:

   (a) If $(\mu_i - \upsilon_i) = 1$, then $1 + 2\sigma_i \geqslant \tau_i$ and $-1 + 2(1-\sigma_i) \geqslant \tau_i$ (constraints $(3e)$ and $(4e)$ from $(c2)$). Since $\tau_i \in \{0.1\}$, $\tau_i = 1$ for $\sigma_i = 0$.

   (b) If $(\mu_i - \upsilon_i) = 1$, then $-1 + 2\sigma_i \geqslant \tau_i$ and $1 + 2(1-\sigma_i) \geqslant \tau_i$ (constraints $(3e)$ and $(4e)$ from $(c2)$). Since $\tau_i \in \{0.1\}$, $\tau_i = 1$ for $\sigma_i = 1$.

In all cases, $|\mu_i - \upsilon_i| = \tau_i$. Thus, the constraints $(c1)$ and $(c2)$ are equivalent for each task t_i, $i = \overline{1..n}$. □

Therefore, the models $(P1e)$ and $(P2e)$ are equivalent and valid for energy constrained scheduling. The behavior of the two models will be compared in section 5.6 on different instances and platforms. In the following, we focus on the model $(P1e)$. The results obtained for $(P1e)$ remain valid for $(P2e)$.

Relaxed problem

The problem of finding the optimal mapping using $(P1e)$ or $(P2e)$ is at least strongly NP-complete even without constraints (8) and (9) and energy constraint. We can use solvers like *CPLEX* to solve $(P1e)$ or $(P2e)$ only for small instances. In order to have an easier problem, we relax integrity variables. We obtain the model $(P1e')$ by relaxing the integer variables x_i, μ_i, υ_i, $y_{i,j}$ and $z_{i,j}$. In the same way, we obtain the model $(P2e')$ by relaxing the integer variables x_i, μ_i, υ_i, τ_i, $y_{i,j}$, $z_{i,j}$ and $b_{i,j}$. However, as in the model $(P2')$ for the problem without an energy constraint, $a_{i,j}$ and σ_i must remain integer (see Section 4.4.1). These two variables must remain integer so that $(P1e')$ and $(P2e')$ are equivalent, i.e., $|y'_{i,j} - z'_{i,j}| = b'_{i,j}$ for each two tasks $(t_i, t_j) \in E$ and $|\mu'_i - \upsilon'_i| = \tau'_i$ for $i = \overline{1..n}$.

To obtain a feasible mapping, we use the same rounding technique θ_1 defined in section 4.4.1, i.e., for each task t_i for $i \in \overline{1..n}$, we set $x^r_i = 0$ if $x'_i < \frac{1}{2}$, and $x^r_i = 1$ otherwise. We denote by θe_1 the mapping obtained by this rounding. Each task t_i is mapped in either a processor type $\mathcal{A}$ or type $\mathcal{B}$. Thus, $\theta e_1(t_i) \longrightarrow \{\mathcal{A}, \mathcal{B}\}$. We also use the iterative mapping θ_3 defined in 4.4.2, which we denote by θe_3 for the scheduling problem with energy constraint, $\theta e_3(t_i) \longrightarrow \{\mathcal{A}, \mathcal{B}\}$.

Proposition 9. *The execution of the tasks according to the mapping* θe_1 *or* θe_3 *keeps the energy constraint valid.*

Proof. Let $\mathbb{E}_i^r = x_i^r e_{i,\mathcal{A}}^{'} + (1 - x_i^r) e_{i,\mathcal{B}}^{'}$ be the energy consumption generated by the execution of a task t_i according to the mapping θe_1 or θe_3, $i = \overline{1..n}$. Let $\mathbb{E}_i^{'}$ be the fractional energy consumption of the task t_i generated by solving the model $(P1e^{'})$ or $(P2e^{'})$. We are going to prove that $\mathbb{E}_i^{'} \geqslant \mathbb{E}_i^r$ for each task t_i, $i = \overline{1..n}$. For each task t_i, we obtain $\mathbb{E}_i^r > 0$ in two cases:

1. If $x_i^{'} \geqslant \frac{1}{2}$ and $\mathbb{E}_i^r = e_{i,\mathcal{A}}^{'} > 0$, then $e_{i,\mathcal{B}}^{'} = 0$, and thus $\mathbb{E}_i^{'} = (x_i^{'} + (1 - |\mu_i - \upsilon_i|)) e_{i,\mathcal{A}}^{'}$. Then, since $\mu_i \leqslant x_i^{'}$ and $\upsilon_i \leqslant 1 - x_i^{'} \leqslant \frac{1}{2}$ (constraint $(c1)$), and since $|\mu_i - \upsilon_i| \leqslant \text{maximum}\{\mu_i, \upsilon_i\}$, we obtain two cases:

   (a) If $\text{maximum}\{\mu_i, \upsilon_i\} = \mu_i$ then $\mathbb{E}_i^{'} = (x_i^{'} + (1 - |\mu_i - \upsilon_i|)) e_{i,\mathcal{A}}^{'} \geqslant (x_i^{'} + (1 - \mu_i)) e_{i,\mathcal{A}}^{'}$. Finally, since $x_i^{'} - \mu_i \geqslant 0$ (from constraint $(e1)$ of (c1)), then $\mathbb{E}_i^{'} \geqslant (x_i^{'} - \mu_i + 1) e_{i,\mathcal{A}}^{'} \geqslant e_{i,\mathcal{A}}^{'}$.

   (b) If $\text{maximum}\{\mu_i, \upsilon_i\} = \upsilon_i$ then $\mathbb{E}_i^{'} = (x_i^{'} + (1 - |\mu_i - \upsilon_i|)) e_{i,\mathcal{A}}^{'} \geqslant (x_i^{'} + (1 - \upsilon_i)) e_{i,\mathcal{A}}^{'}$. Finally, $\upsilon_i \leqslant \frac{1}{2} \leqslant x_i^{'}$, then $\mathbb{E}_i^{'} \geqslant (x_i^{'} - \upsilon_i + 1) e_{i,\mathcal{A}}^{'} \geqslant e_{i,\mathcal{A}}^{'}$.

   Then, since $\mathbb{E}_i^r = e_{i,\mathcal{A}}^{'}$, we have $\mathbb{E}_i^{'} \geqslant \mathbb{E}_i^r$.

2. If $x_i^{'} < \frac{1}{2}$ and $\mathbb{E}_i^r = e_{i,\mathcal{B}}^{'} > 0$ then $e_{i,\mathcal{A}}^{'} = 0$, and thus $\mathbb{E}_i^{'} = ((1 - x_i^{'}) + (1 - |\mu_i - \upsilon_i|)) e_{i,\mathcal{B}}^{'}$. Then, $\mu_i \leqslant x_i^{'} < \frac{1}{2}$ and $\upsilon_i \leqslant 1 - x_i$ (constraint $(c1)$), and since $|\mu_i - \upsilon_i| \leqslant \text{maximum}\{\mu_i, \upsilon_i\}$, we obtain two cases:

   (a) If $\text{maximum}\{\mu_i, \upsilon_i\} = \mu_i$ then $\mathbb{E}_i^{'} = ((1 - x_i^{'}) + (1 - |\mu_i - \upsilon_i|)) e_{i,\mathcal{B}}^{'} \geqslant ((1 - x_i^{'}) + (1 - \mu_i)) e_{i,\mathcal{B}}^{'}$. Finally, $\frac{1}{2} > x_i^{'} \geqslant \mu_i$, then $\mathbb{E}_i^{'} \geqslant (2 - x_i^{'} - \mu_i) e_{i,\mathcal{B}}^{'} > e_{i,\mathcal{B}}^{'}$.

   (b) If $\text{maximum}\{\mu_i, \upsilon_i\} = \upsilon_i$ then $\mathbb{E}_i^{'} = ((1 - x_i^{'}) + (1 - |\mu_i - \upsilon_i|)) e_{i,\mathcal{B}}^{'} \geqslant ((1 - x_i^{'}) + (1 - \upsilon_i)) e_{i,\mathcal{B}}^{'}$. Finally, $\upsilon_i \leqslant 1 - x_i^{'}$, then $\mathbb{E}_i^{'} \geqslant (1 - x_i^{'} - \upsilon_i + 1) e_{i,\mathcal{B}}^{'} \geqslant e_{i,\mathcal{B}}^{'}$.

   Then, since $\mathbb{E}_i^r = e_{i,\mathcal{B}}^{'}$, we have $\mathbb{E}_i^{'} \geqslant \mathbb{E}_i^r$.

In the two cases, $\mathbb{E}_i^{'} \geqslant \mathbb{E}_i^r$ for $i = \overline{1..n}$. Finally, $\sum_{i=1}^{n} \mathbb{E}_i^r \leqslant \sum_{i=1}^{n} \mathbb{E}_i^{'} \leqslant \mathbb{E}^{'}$. Thus, the execution of the tasks according to the mapping θe_1 or θe_3 keeps the energy constraint valid. □

Remark 9. An easier way to represent the constraint $(c1)$ (resp. $(c2)$) in $(P1e)$ (resp. $(P2e)$) is $\sum_{i=1}^{n} (x_i e_{i,\mathcal{A}}^{'} + (1 - x_i) e_{i,\mathcal{B}}^{'}) \leqslant \mathbb{E}^{'}$. However, solving $(P1e^{'})$ (resp. $(P2e^{'})$) with this constraint, and then apply the rounding θe_1 or θe_3, the resulting solution may not satisfy the energy constraint. Indeed, let us consider the single task t_1, such that $w_{i,\mathcal{A}} = 2$ and $w_{i,\mathcal{B}} = 10$, $e_{1,\mathcal{A}}^{'} = 20$ and $e_{1,\mathcal{B}}^{'} = 0$, $\mathbb{E}^{'} = 10$. Hence, the optimal solution of $(P1e^{'})$ and $(P2e^{'})$ is $C_{max}^{'} = \frac{1}{2} w_{i,\mathcal{A}}^{'} + \frac{1}{2} w_{i,\mathcal{B}} = 6$ with $x_1^{'} = \frac{1}{2}$, where $x_1^{'} e_{1,\mathcal{A}}^{'} + (1 - x_1^{'}) e_{i,\mathcal{B}}^{'} = 10 = \mathbb{E}^{'}$. By using the rounding θe_1 or θe_3, we obtain $x_1^r = 1$, with an energy consumption equal to $x_1^r e_{1,\mathcal{A}}^{'} + (1 - x_1^r) e_{1,\mathcal{B}}^{'} = 20 > \mathbb{E}^{'}$, which is not feasible. Thus, this constraint is not valid for this problem.

II) Phase 2: scheduling algorithm

Proposition 9 shows that the energy constraint will be satisfied by applying the mapping θe_1 obtained in the first phase. For the second phase, we use Algorithm 7 proposed in 4.4.1 for the scheduling problem without energy constraint.

The following algorithm 9 executes task by task from the ready task list according to the order given by list L (LST[1], LFP[2], LLP[3]). Thus, if two tasks are executable at a given time on a free processor, then we chose the first tasks according to the order given by the list L. Furthermore,

[1]List by Start Time (LST)
[2]List by Finish Time (LFT)
[3]List by longest path (LLP)

Each task t_i is executed on one of the processors of type $\theta_1(t_i)$ using an insertion-based scheduling policy.

Algorithm 9: LSWPAe Algorithm

```
Data: mapping θe1, list L (LST, LFT, LLP).
Result: Feasible scheduling.
begin
    Create an empty list ready-list;
    ready-list= {t_j, Γ⁻(t_j) = ∅, j = 1..n};
    while ready-list ≠ ∅ do
        t_i ⟵ the first executable task from the ready-list according to the order given
         by list L;
        for all p_r ∈ P(θe1(t_i)) do
            Compute EFT(t_i, p_r) value using an insertion-based scheduling policy;
        Assign task t_i to the processor p_r that minimize EFT of task t_i;
        Update ready-list with unscheduled tasks;
```

Complexity:

The first phase consists in solving either a NLMIP model ($P1e^{'}$) or a MIP model ($P2e^{'}$), which makes the complexity of the first phase exponential, its resolution therefore depends on the size of the instance to be solved.

The second phase has the same structure as HEFT which has $\mathcal{O}(n^2m)$ time complexity. However, for LSWPAe Algorithm, the Earliest Finish Time $EFT(t_i, p_r)$ of each task t_i is calculated for either ℓ or k processors according to the mapping θe_1. For each task t_i (n in total), we calculate its $EFT(t_i, p_r)$ by taking into account all its predecessors ($n-1$ at most) on each available processing element of type $\theta e_1(t_i)$ (at most max(ℓ,k)). The insertion policy is verified on a processing element by checking the non-overlapping with at most $(n-1)$ tasks. This makes a complexity of $\mathcal{O}(max(\ell, k)n^2)$ for the second phase. Thus, the complexity time of LSWPAe algorithm is exponential even if the second phase is polynomial.

Proposition 10. *From Proposition 9, the mapping θe_1 obtained in the phase 1 respects the energy constraint. Furthermore, since the solution obtained by solving the model ($P1e'$) or ($P2e'$) is a lower bound for our problem, and the second phase does not consider the energy constraint. Then, using Algorithm 9 keeps the same ratio of 6 for the scheduling problem with an energy constraint.*

Remark 10. The specialized accelerator problem (where a set of tasks $T^{'} \in T$ can be executed by either a CPU or a GPU only) can be solved with the same methods. It is sufficient to set the corresponding variable $x_i^{'}$ to 0 or 1 in ($P1e^{'}$) or ($P2e^{'}$). However, the amount of energy generated by the execution of the set of tasks $T^{'}$ must be sufficient, i.e., $\sum_{t_i \in T'} x_i^{'} e_{i,\mathcal{A}} + (1 - x_i^{'}) e_{i,\mathcal{B}} \leqslant \mathbb{E}^{'}$.

5.5 Polynomial List Scheduling algorithm With Pre-Allocation (PLSWPAe)

We have previously shown that we can modify LSWPA algorithm to manage the energy constraint. We thus proposed LSWPAe algorithm that takes into account the energy constraint by keeping the same performance guarantee ratio of 6.

In this section, we modify PLSWPA algorithm to obtain a polynomial method to solve the scheduling problem with an energy constraint. For this purpose, we modify only the first phase, which is the task allocation phase, adding a new constraint. To handle the energy constraint, a new constraint will be added to the two models ($P3$) and ($P4^{'}$) introduced in Section 4.4.2 of

the previous chapter. We will then obtain the PLSWPAe algorithm which also uses three steps as PLSWPA algorithm. However, to keep the method polynomial, the energy constraint could not be respected in this case. Indeed, we must use only continuous variables, which will allow us to keep the same ratio 6α. We will then demonstrate in the lemma 20 that the energy consumed using this method is at most $2\mathbb{E}^{'}$. The first step consists of solving an assignment problem to find which type of processor ($\mathcal{A}$ or $\mathcal{B}$) will execute each task. We obtain the model ($P3e$) by adding an energy constraint to the model ($P3$). We then use the solution obtained by solving the relaxation ($P3e^{'}$) of the model ($P3e$) to get another model ($P4e^{'}$). Then, by rounding its fractional solution, we get a feasible assignment of the tasks. In the last phase, the assignment of the tasks and a list scheduling algorithm are used to get a feasible schedule. In the following, the three phases are presented in detail.

Phase 1: assignment of tasks (model ($P3e$))

By adding the constraint ($1e$) to the model ($P3$) previously defined in Section 4.4.2 for the problem without energy constraint, we obtain the following model ($P3e$).

$$(P3e)\begin{cases} C_i + x_j w_{j,\mathcal{A}} + (1-x_j)w_{j,\mathcal{B}} + \zeta_{i,j} cm_{i,j} \leqslant C_j,\ \forall (t_i,t_j) \in E & (1)\\ x_i - x_j \leqslant \zeta_{i,j},\ \forall (t_i,t_j) \in E & (2)\\ x_j - x_i \leqslant \zeta_{i,j},\ \forall (t_i,t_j) \in E & (3)\\ x_i w_{i,\mathcal{A}} + (1-x_i)w_{i,\mathcal{B}} \leqslant C_i, \forall i = \overline{1..n}, \Gamma^-(i) = \emptyset & (4)\\ 0 \leqslant C_i \leqslant C_{max3}, \forall i = \overline{1..n}, \Gamma^+(i) = \emptyset & (5)\\ \sum_{i=1}^{n} x_i w_{i,\mathcal{A}} \leqslant \ell C_{max3} & (6)\\ \sum_{i=1}^{n} (1-x_i) w_{i,\mathcal{B}} \leqslant k C_{max3} & (7)\\ \sum_{i=1}^{n} (x_i e^{'}_{i,\mathcal{A}} + (1-x_i) e^{'}_{i,\mathcal{B}}) \leqslant \mathbb{E}^{'} & (1e)\\ x_i \in \{0,1\},\ \zeta_{i,j} \in [0,1],\ \forall i,j = \overline{1..n} & \\ Z(min) = C_{max3} & \end{cases}$$

We remind that we relax the integer variables $x^{'}_i$ for $i = \overline{1..n}$ to obtain an easier problem, and we get the relaxed model ($P3e^{'}$). We denote by $\widetilde{x}^{'}_i \in [0,1]$, the fractional value of $x^{'}_i$ in the optimal solution of the model ($P3e^{'}$).

Phase 2: assignment of tasks (model ($P4e^{'}$))

Then, by using the solution of ($P3e^{'}$), we define another model ($P4e^{'}$). The decision variables are $x^{'}_i$, and an intermediary variable $y^{'}_{i,j} \in [0,1]$, with $i = \overline{1..n}$ and $j = \overline{1..n}$. For all $(t_i,t_j) \in E$, we define the constraint $Con_{i,j}$ as follows:

- If $\min\{\widetilde{x}^{'}_i, \widetilde{x}^{'}_j\} > \min\{1-\widetilde{x}^{'}_i, 1-\widetilde{x}^{'}_j\}$, then $Con_{i,j} = \begin{cases} y^{'}_{i,j} \leqslant x^{'}_i & (1)\\ y^{'}_{i,j} \leqslant x^{'}_j & (2)\\ \zeta^{'}_{i,j} = (1-y^{'}_{i,j}) & (3) \end{cases}$

  From $Con_{i,j}$, we have $y^{'}_{i,j} \leqslant min\{x^{'}_i, x^{'}_j\}$. Then, $\zeta^{'}_{i,j} = 1 - y^{'}_{i,j} \geqslant (1 - min\{x^{'}_i, x^{'}_j\})$, which is equivalent to the constraint $\widetilde{Con}^{1}_{i,j}$. Since it is a minimization problem, we can set $\zeta^{'}_{i,j} = (1 - min\{x^{'}_i, x^{'}_j\})$.

- If $\min\{\widetilde{x}^{'}_i, \widetilde{x}^{'}_j\} \leqslant \min\{1-\widetilde{x}^{'}_i, 1-\widetilde{x}^{'}_j\}$, then $Con_{i,j} = \begin{cases} y^{'}_{i,j} \leqslant 1 - x^{'}_i & (1)\\ y^{'}_{i,j} \leqslant 1 - x^{'}_j & (2)\\ \zeta^{'}_{i,j} = (1-y^{'}_{i,j}) & (3) \end{cases}$

  From $Con_{i,j}$, we have $y^{'}_{i,j} \leqslant min\{1-x^{'}_i, 1-x^{'}_j\}$. Then, $\zeta^{'}_{i,j} = 1 - y^{'}_{i,j} \geqslant (1 - min\{1-x^{'}_i, 1-x^{'}_j\})$, which is equivalent to the constraint $\widetilde{Con}^{2}_{i,j}$. Since it is a minimization problem, we can set $\zeta^{'}_{i,j} = (1 - min\{1-x^{'}_i, 1-x^{'}_j\})$.

We then obtain the model $(P4e^{'})$ as follows:

$$(P4e^{'})\begin{cases} C_i^{'} + x_j^{'}w_{j,\mathcal{A}} + (1-x_j^{'})w_{j,\mathcal{B}} + \zeta_{i,j}^{'}cm_{i,j} \leqslant C_j^{'}, \forall (t_i,t_j) \in E & (1) \\ Con_{i,j}, \forall (t_i,t_j) \in E & (2) \\ x_i^{'}w_{i,\mathcal{A}} + (1-x_i^{'})w_{i,\mathcal{B}} \leqslant C_i^{'}, \forall i = \overline{1..n}, \Gamma^{-}(i) = \emptyset & (3) \\ 0 \leqslant C_i^{'} \leqslant C_{max4'}, \forall i = \overline{1..n}, \Gamma^{+}(i) = \emptyset & (4) \\ \sum_{i=1}^{n} x_i^{'}w_{i,\mathcal{A}} \leqslant \ell C_{max4'} & (5) \\ \sum_{i=1}^{n} (1-x_i^{'})w_{i,\mathcal{B}} \leqslant kC_{max4'} & (6) \\ \sum_{i=1}^{n} (x_i^{'}e_{i,\mathcal{A}}^{'} + (1-x_i^{'})e_{i,\mathcal{B}}^{'}) \leqslant \mathbb{E}^{'} & (1e) \\ x_i^{'}, y_{i,j}^{'}, \zeta_{i,j}^{'} \in [0,1], \ \forall i = \overline{1..n}, j = \overline{1..n} & \\ Z(min) = C_{max4'} & \end{cases}$$

Rounding strategy θe_2:

To obtain a feasible mapping, we use the same rounding technique θ_2 defined in section 4.4.1, i.e., for each task t_i with $i \in \overline{1..n}$, we set $x_i^r = 0$ if $x_i^{'} < \frac{1}{2}$, and $x_i^r = 1$ otherwise. We denote by θe_2 the mapping obtained by this rounding. We prove by lemma 20 that the solution obtained by this rounding does not respect the energy constraint, but does not exceed twice the authorized quantity $\mathbb{E}^{'}$.

Lemma 20. *The energy consumption generated by the assignment of the tasks obtained by θe_2 with rounding the fractional solutions $x_i^{'}$ (obtained by solving the model $P4e^{'}$) is smaller than $2\mathbb{E}^{'}$.*

Proof. Let $\mathbb{E}_i^r = x_i^r e_{i,\mathcal{A}}^{'} + (1-x_i^r)e_{i,\mathcal{B}}^{'}$ be the energy consumption generated by the execution of the task t_i according to the mapping θe_2, $i = \overline{1..n}$. Let $\mathbb{E}_i^{'} = x_i^{'}e_{i,\mathcal{A}}^{'} + (1-x_i^{'})e_{i,\mathcal{B}}^{'}$ be the fractional energy consumption of the task t_i generated by solving the model $P4e^{'}$. We are going to prove that $\sum_{i=1}^{n} \mathbb{E}_i^r \leqslant 2\mathbb{E}^{'}$. For each task t_i, we obtain $\mathbb{E}_i^r > 0$ in two cases:

1. If $x_i^{'} \geqslant \frac{1}{2}$ and $\mathbb{E}_i^r = e_{i,\mathcal{A}}^{'} > 0$ $(e_{i,\mathcal{B}}^{'} = 0)$, then $x_i^r = 1$ and $\mathbb{E}_i^{'} = x_i^{'}e_{i,\mathcal{A}}^{'}$. Then, $\mathbb{E}_i^r = x_i^r e_{i,\mathcal{A}}^{'} + (1-x_i^r)e_{i,\mathcal{B}}^{'} = e_{i,\mathcal{A}}^{'} \leqslant 2x_i^{'}e_{i,\mathcal{A}}^{'} = 2\mathbb{E}_i^{'}$. Thus, $\mathbb{E}_i^r \leqslant 2\mathbb{E}_i^{'}$.

2. If $x_i^{'} < \frac{1}{2}$ and $\mathbb{E}_i^r = e_{i,\mathcal{B}}^{'} > 0$ $(e_{i,\mathcal{A}}^{'} = 0)$, then $x_i^r = 0$ and $\mathbb{E}_i^{'} = (1-x_i^{'})e_{i,\mathcal{B}}^{'}$. Furthermore, since $x_i^{'} < \frac{1}{2}$, we have $1 - x_i^{'} > \frac{1}{2}$. Then, $\mathbb{E}_i^r = x_i^r e_{i,\mathcal{A}}^{'} + (1-x_i^r)e_{i,\mathcal{B}}^{'} = e_{i,\mathcal{B}}^{'} < 2(1-x_i^{'})e_{i,\mathcal{B}}^{'} = 2\mathbb{E}_i^{'}$. Thus, $\mathbb{E}_i^r < 2\mathbb{E}_i^{'}$.

In the two cases, $\mathbb{E}_i^r \leqslant 2\mathbb{E}_i^{'}$ for each task t_i, $i = \overline{1..n}$. Finally, $\sum_{i=1}^{n} \mathbb{E}_i^r \leqslant 2\sum_{i=1}^{n} \mathbb{E}_i^{'} \leqslant 2\mathbb{E}^{'}$. □

The behavior of PLSWPAe's energy consumption will be compared in section 5.6 on different instances and platforms. In the following, we summarize the three steps of PLSWPAe algorithm.

PLSWPAe algorithm:

To obtain a feasible mapping, the same rounding strategy used to obtain θe_1 is applied to round the solution obtained by the model $(P4e^{'})$, where each task t_i is mapped in either a processor type $\mathcal{A}$ or type $\mathcal{B}$. We set $x_i^r = 0$ if $x_i^{'} < \frac{1}{2}$, and $x_i^r = 1$ otherwise. We denoted by θe_2 the mapping obtained by this mapping, $\theta e_2(t_i) \longrightarrow \{\mathcal{A}, \mathcal{B}\}$.

PLSWPAe algorithm has the same structure of LSWPA algorithm, using the mapping θe_2 instead of θe_1 for the first phase, and the same list algorithm in the last phase. The three steps of PLSWPAe algorithm can be summarized as follows:

1. Solve the relaxed model $(P3e^{'})$.

2. Use the solution of $(P3e^{'})$ to define another model $(P4e^{'})$, then solve $(P4e^{'})$.

3. After rounding the solutions obtained by $(P4e^{'})$, use Algorithm 9 with the obtained mapping θe_2 and a priority list L (LST, LFT, LLP).

Complexity:

The rounding θe_2 is obtained using two models containing only continuous variables, which makes polynomial the resolution of the first phase of PLSWPAe algorithm. Furthermore, the second phase of PLSWPAe and LSWPAe algorithms are identical, with a $\mathcal{O}(max(\ell, k)n^2)$ time complexity.

Remark 11. Let $C^{\star}_{max4'}$ (resp. $C^{\star}_{max3'}$) be the optimal solution of the model $(P4e^{'})$ (resp. $(P3e^{'})$). Let $\widehat{C}_{max2e}$ be the solution obtained by PLSWPAe algorithm and $C^{\star}_{maxe}$ be the optimal scheduling solution of our main problem. For the makespan objective, PLSWPAe algorithm uses the same technique as PLSWPA algorithm. Thus, by supposing that $C^{\star}_{max4'} \leqslant \alpha C^{\star}_{max3'}$, with $\alpha \in \mathbb{R}^{+}$, we obtain the same performance guarantee as PLSWPA, i.e., $\dfrac{\widehat{C}_{max2e}}{C^{\star}_{maxe}} < 6\alpha$.

Iterative mapping θe_4:

The objective of this strategy is to do several iterations to assign the tasks progressively. For a given integer $\upsilon \geqslant 2$, we solve the model $(P4e^{'})$ $\lfloor \frac{\upsilon}{2} \rfloor$ times by adding new assignment constraints at each resolution. Contrary to θe_2, we try to assign the tasks progressively, starting by setting $x_j^{'}$ to 0 if $x_j^{'} < (\frac{1}{\upsilon})$ and $x_j^{'}$ to 1 if $x_j^{'} \geqslant 1 - (\frac{1}{\upsilon})$ for the first resolution of $(P4e^{'})$, and finishing by setting $x_j^{'}$ to 0 if $x_j^{'} < (\frac{\lfloor \frac{\upsilon}{2} \rfloor}{\upsilon})$ and $x_j^{'}$ to 1 if $x_j^{'} \geqslant 1 - (\frac{\lfloor \frac{\upsilon}{2} \rfloor}{\upsilon})$.

Algorithm 10: Rounding algorithm.

Data: model $(P4e^{'})$, a parameter $\upsilon \geqslant 2$.
Result: mapping θe_4.
begin
  $\mathbb{E}c = 0$;
  $T^{'} \longleftarrow \{\}$;
  for $it = 1$ *to* $\lfloor \frac{\upsilon}{2} \rfloor$ **do**
    Solve (P);
    for $j = 1$ *to* n **do**
      if $x_j^{'} < \frac{it}{\upsilon}$ *and* $\mathbb{E}c + e_{j,\mathcal{B}} \leqslant \mathbb{E}^{'}$ *and* $t_j \notin T^{'}$ **then**
        Set in $(P) : x_j^{'} = 0$;
        $\mathbb{E}c = \mathbb{E}c + e_{j,\mathcal{B}}$;
        $T^{'} \longleftarrow T^{'} \cup \{t_j\}$;
      if $x_j^{'} \geqslant 1 - \frac{it}{\upsilon}$ *and* $\mathbb{E}c + e_{j,\mathcal{A}} \leqslant \mathbb{E}^{'}$ *and* $t_j \notin T^{'}$ **then**
        Set in $(P) : x_j^{'} = 1$;
        $\mathbb{E}c = \mathbb{E}c + e_{j,\mathcal{A}}$;
        $T^{'} \longleftarrow T^{'} \cup \{t_j\}$;
  for $l = 1$ *to* n **do**
    if $x_l^{'} < \frac{1}{2}$ **then**
      $x_l^{r} = 0$
    else
      $x_l^{r} = 1$

However, the model $(P4e^{'})$ may not be solvable if the amount of energy related to the assignment of some tasks $T^{'}$ exceeds the amount of energy allowed, i.e., $\sum_{t_i \in T'} x_i^{'} e_{i,\mathcal{A}} + (1 - x_i^{'}) e_{i,\mathcal{B}} > \mathbb{E}^{'}$. Thus, to avoid this, we add in algorithm 10 an assignment condition, such that, before the assignment of each task t_i, we verify if it does not cause an excess of allowed energy consumption $\mathbb{E}$. For this purpose, We calculate the energy consumed $\mathbb{E}c =$ after each authorised assignment, and we make sure it does not exceed the quantity $\mathbb{E}$ for the next assignments.

After the last iteration, it may be that some tasks are not assigned, they are then assigned using the same principle as the rounding θe_2, i.e., for each task t_i with $i \in \overline{1..n}$, we set $x_i^r = 0$ if $x_i^{'} < \frac{1}{2}$, and $x_i^r = 1$ otherwise. Thus, as rounding θe_2, this rounding may not respect the energy constraint $\mathbb{E}^{'}$, but generates an energy consumption quantity smaller than $2\mathbb{E}^{'}$ (Lemma 20).

Remark 12. The specialized accelerator problem (where a set of tasks $T^{'} \in T$ can be executed by only one type of processing elements, i.e., by a CPU or a GPU only) can be solved with PLSWPAe algorithm. It is sufficient to set the corresponding variables $x_i^{'}$ to 0 or 1 in the models $(P3e^{'})$ and $(P4e^{'})$. However, the amount of energy generated by the execution of the set of tasks $T^{'}$ must be sufficient, i.e., $\sum_{t_i \in T'} x_i^{'} e_{i,\mathcal{A}} + (1 - x_i^{'}) e_{i,\mathcal{B}} \leqslant \mathbb{E}^{'}$.

5.6 Numerical results

In this section, we compare the performance of LSWPAe[4] and PLSWPAe[5] algorithms using benchmarks generated by Turbine [108] as the previous chapter.

In what follows, we first describe the generation of benchmarks and all used parameters. Then, we discuss the efficiency of LSWPAe algorithm using θe_1 and θe_3 compared to PLSWPAe algorithm using θe_2 and θe_4.

5.6.1 Benchmark

Table 5.1: *Description of the applications and the platforms.*

Instances	Number of tasks	Platform 1		Platform 2	
		ℓ	k	ℓ	k
test_1	10	3	3	1	1
test_2	30	4	4	1	1
test_3	60	4	4	1	1
test_4	100	6	6	1	1
test_5	200	6	6	1	1
test_6	400	6	6	1	1
test_7	500	8	8	1	1
test_8	600	8	8	1	1
test_9	800	8	8	1	1
test_10	1000	12	12	1	1

The benchmark used for all tests is composed of ten parallel DAG applications. We denote by $test\_i$ the instance number i. We generate 10 different applications for each $test\_i$ with $i = \overline{1..10}$. The execution times of the tasks are generated randomly over an interval $[w_{min}, w_{max}]$, w_{min} has been fixed at 5 and w_{max} at 30. The degrees of the tasks (successors) are generated randomly over an interval $[d_{min}, d_{max}]$, d_{min} has been fixed at 1 and d_{max} at 10.

Furthermore, communication delay for each arc was generated on an interval $[ct_{min}, ct_{max}]$, we set ct_{min} to 35 and ct_{max} to 50. Table 5.1 presents the size of each instance generated as well

[4]List Scheduling With Pre-Allocation algorithm
[5]Polynomial List Scheduling With Pre-Allocation algorithm

as the number of the processing elements of type $\mathcal{A}$ and type $\mathcal{B}$ used to execute each instance. For Platform 1, we increase the number of processing elements by increasing the size of the applications. For Platform 2, we only use one processor of type $\mathcal{A}$ and one processor of type $\mathcal{B}$.

For each task t_i, we generate randomly the values of energy quantities $e_{i,\mathcal{A}}$ if we execute t_i on a CPU and $e_{i,\mathcal{B}}$ if we execute t_i on a GPU. The energy consumption of each task is generated randomly over an interval $[e_{min}, e_{max}]$, e_{min} has been fixed at 10 and e_{max} at 30.

The minimum amount of energy to execute each application is calculated as $\mathbb{E}_{min} = \sum_{i=1}^{n} min\{e_{i,\mathcal{A}}, e_{i,\mathcal{B}}\}$. The maximum amount of energy to execute each application is calculated as $\mathbb{E}_{max} = \sum_{i=1}^{n} max\{e_{i,\mathcal{A}}, e_{i,\mathcal{B}}\}$. Then, we randomly generate a quantity $\mathbb{E}$ between $[\mathbb{E}_{min}, \mathbb{E}_{max}]$. Finally, the transformation explained in Proposition 8 is used for all tests.

According to the experimental results obtained in the previous chapter in section 4.5, list LLP is the most effective. Thus, we use only the list LLP for both methods LSWPAe and PLSWPAe.

5.6.2 PLSWPAe compared to LSWPAe algorithm

Table 5.2 and 5.3 (resp. 5.4 and 5.5) illustrate the comparison between LSWPAe and PLSWPAe for the platform 1 (resp. platform 2). Column Time *CPLEX* presents the average time that was needed to *CPLEX* to provide the optimal solution using the model (*Opte*). We only have the result for the first two instances due to the large running time ($> 4h$).

The next six columns of table 5.2 (and 5.4) concern the results of LSWPAe algorithm using LLP and θe_1. Column **GAP** gives the average ratio between makespan obtained by LSWPAe algorithm and C'_{max} (lower bound obtained by $P1e'$ or $P2e'$):

$$GAP = \frac{\text{LSWPAe makespan} - C'_{max}}{C'_{max}} \times 100$$

Column ***E*-cons** indicates the average energy consumption of each method, calculated for each instance size "test_size" by:

$$\begin{aligned} E\text{-cons(test\_size)} &= \frac{100}{10} * \sum_{I=1}^{10} \frac{\text{Energy consumed by the instance } I}{\text{Energy bound } \mathbb{E} \text{ for the instance } I} \\ &= 10 * \sum_{I=1}^{10} \frac{\text{Energy consumed by the instance } I}{\text{Energy bound } \mathbb{E} \text{ for the instance } I} \end{aligned}$$

Column **Best** presents the number of instances where LSWPAe algorithm provides better or the same solution obtained by LSWPAe using θe_3 or PLSWPAe using θe_2 and θe_4. Column **Opt** presents the number of instances where LSWPAe provides the optimal solution (for small instances). Finally, column $\boldsymbol{P1e'}$ (resp. $\boldsymbol{P2e'}$) gives the average time that was needed for LSWPAe algorithm to provide a solution using the model $P1e'$ (resp. $P2e'$) for the first phase. We use the same six columns to illustrate the results of LSWPAe algorithm using LLP and θe_3.

In the tables that illustrate the results of PLSWPAe algorithm using LLP and θe_2 or θe_4(5.3 and 5.5), we add two columns. Column $\overline{\boldsymbol{E}\textbf{-cons}}$ indicates the average energy consumption of instances that do not respect the energy constraint, calculated for each instance size "test_size" by:

$$\overline{E\text{-cons}}(\text{test\_size}) = 10 * \sum_{I=1}^{10} \frac{\text{Energy consumed } \mathbb{E}c \text{ by the instance } I \text{ if } \mathbb{E}c > \mathbb{E}}{\text{Energy bound } \mathbb{E} \text{ for the instance } I}$$

Thus, for this table, E-cons is calculated only for instances that respect the energy constraint. Furthermore, column **feasible** provides the number of instances that respect the energy constraint.

Platform 1

Table 5.2: *LSWPAe algorithm evaluation using the list LLP with θe_1 and θe_3 for the platform* 1.

Instances	Time	LSWPAe algorithm using LLP and θe_1						LSWPAe algorithm using LLP and θe_3					
	CPLEX	Gap	*E*-cons	Best	Opt	$P1e^{'}$	$P2e^{'}$	Gap	*E*-cons	Best	Opt	$P1e^{'}$	$P2e^{'}$
test_1	8.08s	4.26%	67.33%	7	7	1.05s	0.43s	0%	71.28%	10	10	1.10s	0.49s
test_2	85.02s	2.56%	68.65%	7	6	1.33s	0.82s	0.97%	68.70%	9	8	1.49s	0.97s
test_3	X	8.53%	63.64%	5	X	10.37s	10.02s	6.90%	66.09%	8	X	10.96s	10.28s
test_4	X	6.97%	58.93%	4	X	12.27s	9.02s	3.17%	62.45%	9	X	12.59s	9.34s
test_5	X	13.28%	44.40%	4	X	11.06s	12.74s	10.95%	46.37%	6	X	11.67s	13.36s
test_6	X	4.63%	75.18%	3	X	15.93s	9.85s	3.70%	75.18%	10	X	17.19s	11.01s
test_7	X	6.35%	71.45%	3	X	15.90s	8.57s	4.69%	72.32%	8	X	16.68s	9.29s
test_8	X	7.40%	78.63%	2	X	13.03s	7.77s	3.45%	78.62%	9	X	13.97s	8.6s
test_9	X	0.82%	77.62%	7	X	17.01s	11.36s	0.82%	77.90%	8	X	18.19s	12.40s
test_10	X	4.47%	77.37%	2	X	18.87s	13.98s	2.93%	77.41%	9	X	19.78s	14.79s
Average	/	5.92%	68.32%	44%	/	11.68s	8.45s	3.75%	69.63%	86%	/	12.36s	9.05s

Table 5.3: *PLSWPAe algorithm evaluation using the list LLP with θe_2 and θe_4 for the platform* 1.

Instances	Time	PLSWPAe algorithm using LLP and θe_2						PLSWPAe algorithm using LLP and θe_4					
	CPLEX	Gap	Time	Best	*E*-cons	$\overline{E}$-cons	feasible	Gap	Time	Best	*E*-cons	$\overline{E}$-cons	feasible
test_1_10_3	8.08s	3.56%	0.018s	7	78.26%	103.16%	8	3.50%	0.050s	7	77.04%	103.16%	8
test_2	85.02s	5.34%	0.06s	4	79.15%	103.39%	9	4.39%	0.17s	7	78.46%	0%	10
test_3	X	8.75%	0.08s	2	80.25%	0%	10	6.95%	0.21s	4	78.74%	0%	10
test_4	X	9.37%	0.14s	2	76.56%	0%	10	8.70%	0.32s	2	77.77%	0%	10
test_5	X	13.50%	0.47s	2	78.2%	0%	10	14.7%	0.81s	3	79.65%	0%	10
test_6	X	4.27%	1.34s	2	78.49%	0%	10	3.91%	1.95s	5	78.39%	0%	10
test_7	X	7.45%	1.32s	1	79.08%	0%	10	6.007%	1.8s	3	78.96%	0%	10
test_8	X	7.45%	2.20s	2	78.77%	0%	10	3.87%	2.65s	7	78.67%	0%	10
test_9	X	1.19%	5.83s	5	78.78%	0%	10	1.106%	6.39s	7	78.76%	0%	10
test_10	X	3.76%	9.49s	6	77.8%	0%	10	2.83%	9.95s	5	77.67%	0%	10
Average	/	6.46%	2.09s	33%	78.53%	103.25%	97%	5.59%	2.43s	50%	78.41%	103.16%	98%

From table 5.2, we can notice that LSWPAe algorithm provides better solutions using the rounding θe_3 compared to θe_1 with a GAP of 3.75% and 86% of best solutions. We can also notice that $P2e^{'}$ is more efficient than $P1e^{'}$ in running time using rounding θe_1 or θe_3.

From table 5.3, we can notice that PLSWPAe algorithm provides better solutions using the rounding θe_4 compared to θe_2 with a GAP of 5.59% and 50% of best solutions. Furthermore, PLSWPAe algorithm provides 98% feasible solutions using θe_4 against 97% using θe_2. The energy excess is not very important, with less than 4% using both roundings θe_2 and θe_4.

Comparing tables 5.2 and 5.3, we can notice that LSWPAe is more efficient than PLSWPAe with 86% of best solutions and a GAP of 3.75%. However, PLSWPAe is more efficient in term of running time, with an average of 2.09 seconds using θe_2 against 8.45 seconds for LSWPAe using θe_1 with $P2e^{'}$, and 2.43 seconds using θe_4 against 9.05 seconds for LSWPAe using θe_3 with $P2e^{'}$.

Finally, LSWPAe is more efficient in energy consumption compared to PLSWPAe, with an average consumption equal to 68.32% using θe_1 against 78.53% for PLSWPAe using θe_2 (feasible solutions), and 69.63% using θe_3 against 78.41% for PLSWPAe using θe_4 (feasible solutions).

Platform 2

Table 5.4: *LSWPAe algorithm evaluation using the list LLP with θe_1 and θe_3 for the platform* 2.

Instances	Time	LSWPAe algorithm using LLP and θe_1						LSWPAe algorithm using LLP and θe_3					
	CPLEX	Gap	*E*-cons	Best	Opt	$P1e^{'}$	$P2e^{'}$	Gap	*E*-cons	Best	Opt	$P1e^{'}$	$P2e^{'}$
test_1	0.37s	16.98%	68.46%	10	3	1.20s	1.26s	18.91%	67.15%	9	3	1.27s	1.33s
test_2	157.54s	48.28%	70.16%	6	0	2.69s	2.62s	45.68%	70.86%	9	0	3.06s	2.79s
test_3	X	21.96%	78.40%	10	X	0.92s	0.66s	22.31%	78.67%	9	X	1.16s	0.86s
test_4	X	7.82%	75.02%	10	X	0.71s	0.72s	7.82%	75.02%	10	X	1.02s	1.03s
test_5	X	0.35%	77.90%	8	X	2.10s	1.54s	0.26%	77.92%	10	X	2.68s	2.093s
test_6	X	1.43%	75.79%	9	X	6.89s	4.23s	1.56%	75.80%	7	X	8.055s	5.30s
test_7	X	0.18%	77.05%	5	X	5.25s	2.93s	0.16%	77.04%	7	X	5.99s	3.6s
test_8	X	0.17%	77.13%	6	X	7.19s	3.95s	0.16%	77.12%	7	X	8.08s	4.71s
test_9	X	0.08%	78.12%	6	X	6.14s	3.25s	0.11%	78.10%	6	X	6.76s	3.78s
test_10	X	0.07%	78.10%	4	X	7.95s	5.74s	0.10%	78.12%	5	X	8.76s	6.45s
Average	/	9.73%	75.61%	74%	/	4.10s	2.69s	9.70%	75.58%	79%	/	4.68s	3.19s

Table 5.5: *PLSWPAe algorithm evaluation using the list LLP with θe_2 and θe_4 for the platform* 2.

Instances	Time	PLSWPAe algorithm using LLP and θe_2						PLSWPAe algorithm using LLP and θe_4					
	CPLEX	Gap	Time	Best	*E*-cons	$\overline{E}$-cons	feasible	Gap	Time	Best	*E*-cons	$\overline{E}$-cons	feasible
test_1	0.37s	9.05%	0.02s	8	67.50%	115.41%	7	14.05%	0.05s	6	71.37%	101.86%	9
test_2	157.54s	44.74%	0.04s	5	74.49%	101.07%	9	45.54%	0.13s	4	74.23%	101.07%	9
test_3	X	21.96%	0.06s	10	78.40%	0%	10	22.31%	0.17s	9	78.67%	0%	10
test_4	X	7.82%	0.10s	10	75.02%	0%	10	7.82%	0.25s	10	75.02%	0%	10
test_5	X	0.31%	0.23s	7	78.11%	0%	10	0.31%	0.51s	7	78.11%	0%	10
test_6	X	1.56%	0.9s	7	76.14%	0%	10	1.54%	1.54s	7	76.18%	0%	10
test_7	X	0.24%	0.7s	5	77.60%	0%	10	0.16%	1.5s	5	77.54%	0%	10
test_8	X	0.21%	1.13s	3	77.61%	0%	10	0.15%	1.57s	5	77.64%	0%	10
test_9	X	0.14%	1.08s	5	78.73%	0%	10	0.07%	1.35s	6	78.68%	0%	10
test_10	X	0.12%	1.80s	3	78.35%	0%	10	0.10%	2.15s	3	78.36%	0%	10
Average	/	8.61%	0.60s	63%	76.19%	108.2%	96%	9.20%	0.92s	62%	76.58%	101.45%	98%

From table 5.4, we can notice that LSWPAe algorithm provides better solutions using the rounding θe_3 compared to θe_1 with a GAP of 9.70% and 79% of best solutions. We can also notice that $P2e^{'}$ is more efficient than $P1e^{'}$ in running time using rounding θe_1 or θe_3.

Contrary to the platform 1, we can notice from table 5.5 that PLSWPAe algorithm provides better solutions using the rounding θe_2 compared to θe_4 with a GAP of 8.61% and 63% of best solutions. However, PLSWPAe algorithm provides 98% feasible solutions using θe_4 against 96% using θe_2. The energy excess is not very important using θe_4, with less than 2% of authorised energy. Using θe_2, the difference may be important, where $\overline{E}$-cons = 115.41% for the first instance size.

Comparing tables 5.4 and 5.5, we can notice that LSWPAe is more efficient than PLSWPAe with 79% of best solutions. However, by using more energy than the authorized quantity, PLSWPAe has a better average GAP, with 8.61% obtained using θe_2 against 9.70% for LSWPAe obtained using θe_3. Furthermore, PLSWPAe is more efficient in term of running time, with an average of 0.60 seconds using θe_2 against 2.69 seconds for LSWPAe using θe_1 with $P2e^{'}$, and 0.92 seconds using θe_4 against 3.19 seconds for LSWPAe using θe_3 with $P2e^{'}$. Finally, LSWPAe is more efficient in energy consumption compared to PLSWPAe also on this platform, with an average consumption equal to 75.61% using θe_1 against 76.19% for PLSWPAe using θe_2 (feasible solutions), and 75.58% using θe_3 against 76.58% for PLSWPAe using θe_4 (feasible solutions).

5.6.3 PLSWPAe compared to LSWPAe algorithm if the execution time of each task is related to its energy consumption

To be as close as possible to reality, we have added a hypothesis to the generation of energy quantities, assuming that each task t_i consumes more on its best processor type (the fastest for task t_i), i.e., if $w_{i,\mathcal{A}} > w_{i,\mathcal{B}}$ then $e_{i,\mathcal{A}} < e_{i,\mathcal{B}}$. This hypothesis keeps right the properties previously defined.

Platform 1

Table 5.6: *LSWPAe algorithm evaluation using the list LLP with θe_1 and θe_3 for the platform* 1.

Instances	Time	LSWPAe algorithm using LLP and θe_1						LSWPAe algorithm using LLP and θe_3					
	CPLEX	Gap	*E*-cons	Best	Opt	$P1e'$	$P2e'$	Gap	*E*-cons	Best	Opt	$P1e'$	$P2e'$
test_1	0.29s	0.13%	79.92%	10	10	0.04s	0.27s	0.13%	81.42%	10	10	0.14s	0.44s
test_2	27.63s	5.72%	67.32%	4	2	0.70s	0.39s	1.40%	72.04%	10	7	0.65s	1.17s
test_3	X	8.47%	62.72%	3	X	1.09s	0.89s	4.27%	69.17%	8	X	1.35s	1.66s
test_4	X	9.63%	61.31%	1	X	1.25s	1.78s	3.89%	68.16%	10	X	1.33s	3.54s
test_5	X	13.69%	59.42%	1	X	3.62s	8.45s	10.55%	65.29%	9	X	4.71s	12.02s
test_6	X	5.55%	99.22%	6	X	19.42s	38.86s	5.21%	98.96%	8	X	21.75s	55.69s
test_7	X	5.99%	99.82%	3	X	63.30s	103.9s	5.31%	99.67%	9	X	65.77s	148.45s
test_8	X	2.43%	93.23%	6	X	25.41s	41.43s	2.19%	93.92%	6	X	27.42s	45.02s
test_9	X	0.48%	99.96%	5	X	20.67s	15.02s	0.37%	99.97%	9	X	21.45s	30.61s
test_10	X	1.97%	99.95%	8	X	59.34s	37.59s	2.41%	99.93%	5	X	63.75s	99.43s
Average	/	5.40%	82.28%	47%	/	19.48s	24.85s	3.57%	84.85%	84%	/	20.83s	39.80s

Table 5.7: *PLSWPAe algorithm evaluation using the list LLP with θe_2 and θe_4 for the platform* 1.

Instances	Time	PLSWPAe algorithm using LLP and θe_2						PLSWPAe algorithm using LLP and θe_4					
	CPLEX	Gap	Time	Best	*E*-cons	$\overline{E}$-cons	feasible	Gap	Time	Best	*E*-cons	$\overline{E}$-cons	feasible
test_1_10_3	0.29s	0.87%	0.003s	9	83.98%	107.43%	7	1.71%	0.008s	9	86.09%	103.93%	8
test_2	27.63s	9.77%	0.009s	3	82.06%	102.85%	9	7.17%	0.02s	4	84.16%	102.85%	9
test_3	X	9.44%	0.01s	1	91.09%	100.38%	8	8.49%	0.02s	2	93.29%	100.42%	9
test_4	X	9.91%	0.02s	2	90.01%	102.95%	7	5.96%	0.04s	2	92.48%	0%	10
test_5	X	11.91%	0.08s	3	96.78%	101.15%	7	12.35%	0.13s	2	97.64%	100.32%	8
test_6	X	8.41%	0.23s	0	99.60%	100.14%	6	6.66%	0.35s	3	99.82%	0%	10
test_7	X	7.16%	0.39s	1	99.75%	100.19%	4	5.27%	0.52s	5	99.84%	100.10%	6
test_8	X	4.90%	0.73s	0	99.83%	100.03%	6	4.03%	0.95s	1	99.91%	100.04%	7
test_9	X	0.49%	1.99s	2	99.91%	100.09%	6	0.53%	2.19s	2	99.90%	0%	10
test_10	X	3.68%	5.86s	4	99.94%	100.08%	4	3.48%	6.09s	4	99.86%	100.05%	8
Average	/	6.65%	0.93s	25%	94.29%	101.52%	64%	5.56%	1.03s	34%	95.29%	101.1%	85%

From table 5.6, we can notice that LSWPAe algorithm provides better solutions using the rounding θe_3 compared to θe_1 with a GAP of 3.57% and 84% of best solutions. We can also notice that $P1e'$ is more efficient than $P2e'$ in running time using rounding θe_1 or θe_3.

From table 5.7, we can notice that PLSWPAe algorithm provides better solutions using the rounding θe_4 compared to θe_2 with a GAP of 5.56% and 34% of best solutions. Furthermore, PLSWPAe algorithm provides 85% feasible solutions using θe_4 against 64% using θe_2. The energy excess is not very important, with less than 8% using both roundings θe_2 and θe_4. Compared to the table 5.3, we notice here that we have more solutions that are not feasible. This is possible, because sometimes, PLSWPAe prefers to assign a task to its best processor, even if it consumes more energy. From Lemma 20, the energy consumption of PLSWPAe algorithm does not exceed $2\mathbb{E}'$.

Comparing tables 5.6 and 5.7, we can notice that LSWPAe is more efficient than PLSWPAe with 84% of the best solutions and a GAP of 3.57%. However, PLSWPAe is more efficient in term of running time, with an average of 0.93 seconds using θe_2 against 19.48 seconds for LSWPAe using θe_1 with $P2e'$, and 1.03 seconds using θe_4 against 20.83 seconds for LSWPAe using θe_3 with $P2e'$.

Finally, LSWPAe is very efficient in energy consumption compared to PLSWPAe, with an average consumption equal to 82.28% using θe_1 against 94.29% for PLSWPAe using θe_2 (feasible solutions), and 84.85% using θe_3 against 95.29% for PLSWPAe using θe_4 (feasible solutions).

Platform 2

Table 5.8: *LSWPAe algorithm evaluation using the list LLP with θe_1 and θe_3 for the platform 2.*

Instances	Time	LSWPAe algorithm using LLP and θe_1						LSWPAe algorithm using LLP and θe_3					
	CPLEX	Gap	*E*-cons	Best	Opt	$P1e^{'}$	$P2e^{'}$	Gap	*E*-cons	Best	Opt	$P1e^{'}$	$P2e^{'}$
test_1	0.12s	18.95%	81.76%	8	3	0.7s	0.49s	17.21%	84.47%	9	3	0.75s	1.41s
test_2	27.14s	45.15%	91.13%	7	0	0.62s	0.63s	44.66%	90.94%	8	0	0.82s	1.32s
test_3	X	22.16%	99.27%	10	X	0.86s	0.67s	22.16%	99.27%	10	X	0.35s	1.35s
test_4	X	5.17%	99.70%	10	X	0.97s	1.008s	5.51%	99.67%	9	X	1.06s	2.75s
test_5	X	1.46%	99.93%	9	X	2.94s	2.23s	1.43%	99.89%	8	X	3.12s	7.08s
test_6	X	0.056%	99.95%	9	X	8.04s	4.26s	0.06%	99.95%	7	X	8.82s	11.22s
test_7	X	0.061%	99.96%	7	X	6.84s	5.58s	0.06%	99.97%	8	X	7.4s	19.86s
test_8	X	1.06%	99.97%	6	X	10.08s	4.91s	1.19%	99.97%	7	X	11.04s	18.66s
test_9	X	0.04%	99.96%	8	X	11.05s	9.007s	0.059%	99.96%	5	X	15.26s	24.81s
test_10	X	0.04%	99.97%	10	X	9.63s	8.64s	0.05%	99.96%	7	X	11.22s	24.57s
Average	/	9.41%	97.16%	84%	/	5.17s	3.74s	9.23%	97.40%	78%	/	6.04s	11.30s

Table 5.9: *PLSWPAe algorithm evaluation using the list LLP with θe_2 and θe_4 for the platform 2.*

Instances	Time	PLSWPAe algorithm using LLP and θe_2						PLSWPAe algorithm using LLP and θe_4					
	CPLEX	Gap	Time	Best	*E*-cons	$\overline{E}$-cons	feasible	Gap	Time	Best	*E*-cons	$\overline{E}$-cons	feasible
test_1	0.12s	11.65%	0.002s	7	82.39%	115.38%	6	19.83%	0.007s	6	88.80%	103.19%	8
test_2	27.14s	39.92%	0.007s	5	88.90%	103.83%	5	40.43%	0.01s	5	94.08%	101.77%	8
test_3	X	23.05%	0.01s	4	99.59%	100.74%	4	23.24%	0.03s	5	98.82%	100.30%	9
test_4	X	4.84%	0.01s	4	99.36%	100.55%	5	5.40%	0.04s	4	99.08%	100.60%	8
test_5	X	1.43%	0.03s	4	99.81%	100.47%	6	1.30%	0.09s	5	99.81%	100.32%	8
test_6	X	0.22%	0.15s	4	99.87%	100.16%	6	0.25%	0.27s	3	99.86%	100.11%	8
test_7	X	0.19%	0.24s	4	99.89%	100.15%	6	0.24%	0.38s	3	99.88%	100.13%	8
test_8	X	1.15%	0.38s	4	99.85%	100.11%	6	1.14%	0.57s	4	99.88%	100.11%	6
test_9	X	0.085%	0.69s	5	99.92%	100.05%	4	0.07%	0.89s	5	99.89%	100.04%	6
test_10	X	0.10%	1.17s	6	99.96%	100.04%	6	0.10%	1.45s	5	99.94%	100.03%	8
Average	/	8.26%	0.26s	47%	96.95%	102.14%	54%	9.2%	0.37s	45%	98.004%	100.66%	77%

From table 5.8, we can notice that LSWPAe algorithm provides 84% of best solutions using the rounding θe_1 against 78% using θe_1. LSWPAe provides a better GAP by using θe_3 with 9.23% against 9.41% using θe_1. We can also notice that $P1e^{'}$ (resp. $P2e^{'}$) is more efficient than $P2e^{'}$ ($P1e^{'}$) in running time using rounding θe_3 (resp. θe_1).

Contrary to the platform 1, we can notice from table 5.9 that PLSWPAe algorithm provides better solutions using the rounding θe_2 compared to θe_4 with a GAP of 8.26% and 47% of best solutions. However, PLSWPAe algorithm provides 77% feasible solutions using θe_4 against 47% using θe_2. The energy excess is not very important using θe_4, with less than 4% of authorised energy. Using θe_2, the difference may be important, where $\overline{E\text{-cons}} = 115.38\%$ for the first instance size.

Comparing tables 5.8 and 5.9, we can notice that LSWPAe is more efficient than PLSWPAe with 84% of best solutions. However, by using more energy than the authorized quantity, PLSWPAe has a better average GAP, with 8.26% obtained using θe_2 against 9.23% for LSWPAe obtained using θe_3. Furthermore, PLSWPAe is more efficient in term of running time. Its average running time is 0.26 seconds using θe_2 against 3.74 seconds for LSWPAe using θe_1 with $P2e^{'}$. Using θe_4, its average running time 0.37 seconds against 6.04 seconds for LSWPAe using θe_3 with $P2e^{'}$. In this case, LSWPAe and PLSWPAe consume almost the same amount of energy.

Comparing the results of the two platforms 1 and 2, we can notice that for the platform 1, the average value of E-cons is given by 82.28% for LSWPAe algorithm using θe_1 and 84.85%

using θe_3. However, for the platform 2, the average value of E-cons is given by 97.16% for LSWPAe algorithm using θe_1 and 97.40% using θe_3. This can be explained by the fact that on the platform 2, we have only one processor p_1 of type $\mathcal{A}$ and one processor p_2 of type $\mathcal{B}$. Only one task in execution can make the processor p_1 or p_2 busy, and this can make the switch from p_1 to p_2 or from p_2 to p_1 more profitable, where communication delays are compensated by waiting times on each processor. Indeed, it is sometimes more profitable to switch from p_1 to p_2 or from p_2 to p_1, to execute a task on its favourite processor (best execution time), than to wait until the end of the task that already occupies p_1 or p_2. Thus, in the platform 1, we consume less energy, because there are more processors. In this case, executing the tasks in parallel without changing the type of processor is sometimes more effective, which is less expensive in term of communication delays, even if some tasks are not executed on their best processors, and thus consume less energy.

5.6.4 comparing PLSWPAe and LSWPAe algorithms using only one processor of type $\mathcal{A}$

In tables 5.10 and 5.11, we compare LSWPAe to PLSWPAe on a platform with only one processor of type $\mathcal{A}$ and $k > 2$ processors of type $\mathcal{B}$.

Table 5.10: *LSWPAe algorithm evaluation using the list LLP with θe_1 and θe_3 if $\ell = 1$ and $k > 2$.*

Instances	Time	LSWPAe algorithm using LLP and θe_1						LSWPAe algorithm using LLP and θe_3					
	CPLEX	Gap	*E*-cons	Best	Opt	$P1e'$	$P2e'$	Gap	*E*-cons	Best	Opt	$P1e'$	$P2e'$
test_1	3.06s	11.53%	70.79%	8	5	1.06s	1.05s	12.02%	67.73%	8	5	1.12s	1.11s
test_2	4019.22s	21.60%	69.69%	5	0	15.99s	20.57s	17.34%	73.34%	8	0	16.27s	20.90s
test_3	X	15.37%	72.22%	2	X	3.72s	4.16s	10.33%	71.89%	9	X	3.91s	4.35s
test_4	X	34.47%	76.44%	5	X	10.36s	8.71s	37.47%	77.87%	5	X	10.76s	9.07s
test_5	X	10.71%	72.28%	9	X	9.51s	4.08s	11.65%	72.39%	5	X	10.43s	4.91s
test_6	X	0.61%	79.52%	9	X	5.53s	4.33s	0.60%	79.55%	10	X	6.71s	5.41s
test_7	X	0.37%	77.34%	9	X	6.67s	5.16s	0.29%	77.28%	9	X	8.11s	6.44s
test_8	X	0.24%	79.74%	9	X	11.65s	6.91s	0.24%	79.64%	8	X	13.70s	8.67s
test_9	X	0.26%	76.07%	8	X	9.61s	8.60s	0.18%	76.14%	10	X	10.85s	9.68s
test_10	X	0.18%	77.11%	9	X	10.52s	10.68s	0.19%	77.14%	9	X	11.43s	11.47s
Average	/	9.53%	75.12%	73%	/	8.46s	7.42s	9.03%	75.29%	81%	/	9.32s	7.05s

Table 5.11: *PLSWPAe algorithm evaluation using the list LLP with θe_2 and θe_4 if $\ell = 1$ and $k > 2$.*

Instances	Time	PLSWPAe algorithm using LLP and θe_2						PLSWPAe algorithm using LLP and θe_4					
	CPLEX	Gap	Time	Best	*E*-cons	$\overline{E\text{-cons}}$	feasible	Gap	Time	Best	*E*-cons	$\overline{E\text{-cons}}$	feasible
test_1	3.06s	7.94%	0.02s	9	60.05%	103.46%	7	10.28%	0.05s	8	58.18%	102.07%	8
test_2	4019.22s	27.62%	0.08s	4	80.22%	0%	10	28.55%	0.24s	3	80.42%	0%	10
test_3	X	21.23%	0.07s	0	79.24%	0%	10	21.89%	0.17s	0	78.45%	0%	10
test_4	X	29.25%	0.16s	4	80.11%	0%	10	26.52%	0.38s	3	79.75%	0%	10
test_5	X	11.76%	0.30s	5	75.04%	0%	10	11.02%	0.70s	4	75.05%	0%	10
test_6	X	0.57%	0.94s	9	79.65%	0%	10	0.57%	1.50s	9	79.65%	0%	10
test_7	X	0.30%	1.54s	9	77.37%	0%	10	0.30%	2.27s	9	77.37%	0%	10
test_8	X	0.19%	2.26s	7	79.90%	0%	10	0.19%	3.35s	7	79.90%	0%	10
test_9	X	0.25%	4.53s	8	76.22%	0%	10	0.19%	5.10s	9	76.20%	0%	10
test_10	X	0.22%	8.23s	7	77.39%	0%	10	0.22%	8.72s	7	77.39%	0%	10
Average	/	9.93%	1.81s	62%	76.51%	103.46%	97%	9.97%	2.24s	59%	76.23%	102.07%	98%

From table 5.10, we can notice that LSWPAe algorithm provides better solutions using the rounding θe_3 compared to θe_1 with a GAP of 9.03% and 81% of best solutions. We can also notice that $P2e'$ is more efficient than $P1e'$ in running time using rounding θe_1 or θe_3.

From table 5.11, we can notice that PLSWPAe algorithm provides better solutions using the rounding θe_2 compared to θe_4 with a GAP of 9.93% and 62% of best solutions. However, PLSWPAe algorithm provides 98% feasible solutions using θe_4 against 97% using θe_2. The energy excess is not very important, with less than 4% using both roundings θe_2 and θe_4.

Comparing tables 5.10 and 5.11, we can notice that LSWPAe is more efficient than PLSWPAe with 81% of the best solutions and a GAP of 9.03%. However, PLSWPAe is more efficient in term of running time, with an average of 1.81 seconds using θe_2 against 7.42 seconds for LSWPAe using θe_1 with $P2e^{'}$, and 2.24 seconds using θe_4 against 7.05 seconds for LSWPAe using θe_3 with $P2e^{'}$.

Finally, LSWPAe is more efficient in energy consumption compared to PLSWPAe also on this platform, with an average consumption equal to 75.12% using θe_1 against 76.51% for PLSWPAe using θe_2 (feasible solutions), and 75.29% using θe_3 against 76.23% for PLSWPAe using θe_4 (feasible solutions).

5.6.5 PLSWPAe compared to LSWPAe algorithm when $\mathbb{E}$ is tight

Now we assume that $\mathbb{E}$ is tight, i.e., the generated value $\mathbb{E}$ is very close to the minimum energy $\mathbb{E}_{min}$ needed to execute an application. The objective is to see if the non-polynomial method LSWPAe can manage complicated instances and provide a solution with a reasonable running time.

Platform 1

Table 5.12: *LSWPAe algorithm evaluation using the list LLP with θe_1 and θe_3 for the platform* 1.

Instances	Time	LSWPAe algorithm using LLP and θe_1						LSWPAe algorithm using LLP and θe_3					
	CPLEX	Gap	*E*-cons	Best	Opt	$P1e^{'}$	$P2e^{'}$	Gap	*E*-cons	Best	Opt	$P1e^{'}$	$P2e^{'}$
test_1	5.53s	2.75%	80.49%	10	9	4.84s	9.04s	2.75%	80.49%	10	9	4.91s	9.12s
test_2	5948.45s	8.34%	84.93%	6	1	32.93s	18.09s	5.59%	88.48%	9	3	33.45s	18.53s
test_3	X	17.38%	86.46%	5	X	54.95s	63.86s	13.50%	91.11%	6	X	55.57s	64.41s
test_4	X	10.57%	79.77%	2	X	41.77s	45.77s	5.22%	87.15%	8	X	44.58s	46.18s
test_5	X	13.43%	86.71%	3	X	104.72s	83.60s	11.83%	92.7262%	7	X	104.90s	83.77s
test_6	X	3.23%	99.95%	8	X	128.50s	108.65s	3.38%	99.93%	8	X	128.87s	109.03s
test_7	X	2.11%	93.67%	6	X	138.84s	89.74s	1.80%	94.19%	8	X	139.23s	90.09s
test_8	X	2.89%	99.89%	7	X	193.88s	160.92s	2.67%	99.93%	8	X	194.46s	161.52s
test_9	X	0.56%	95.21%	5	X	57.06s	148.30s	0.45%	95.69%	9	X	58.17s	149.41s
test_10	X	0.93%	99.97%	6	X	277.51s	146.31s	1.01%	99.97%	8	X	278.07s	146.82s
Average	/	6.21%	90.70%	58%	/	103.5s	87.42s	4.82%	92.96%	81%	/	104.22s	87.88s

Table 5.13: *PLSWPAe algorithm evaluation using the list LLP with θe_2 and θe_4 for the platform* 1.

Instances	Time	PLSWPAe algorithm using LLP and θe_2						PLSWPAe algorithm using LLP and θe_4					
	CPLEX	Gap	Time	Best	*E*-cons	$\overline{E}$-cons	feasible	Gap	Time	Best	*E*-cons	$\overline{E}$-cons	feasible
test_1	5.53s	-4.48%	0.03s	10	75.72%	114.64%	7	-1.21%	0.08s	9	76.34%	113.23%	7
test_2	5948.45s	7.79%	0.11s	5	92.81%	107.68%	3	10.94%	0.316s	5	95.4722%	105.516%	5
test_3	X	16.08%	0.10s	4	97.91%	104.03%	5	20.40%	0.25s	1	96.94%	100.38%	9
test_4	X	13.57%	0.19s	1	98.90%	102.65%	5	14.65%	0.41s	0	97.83%	100.46%	8
test_5	X	16.84%	0.49s	0	98.67%	101.25%	5	16.64%	0.87s	0	98.67%	100.23%	7
test_6	X	6.67%	0.58s	1	99.62%	100.47%	6	5.44%	1.24s	1	99.72%	100.14%	6
test_7	X	5.61%	0.56s	1	99.60%	100.33%	8	4.48%	0.78s	1	99.80%	100.14%	9
test_8	X	3.28%	1.28s	5	99.85%	100.37%	6	2.53%	1.55s	4	99.90%	100.10%	8
test_9	X	0.55%	3.61s	4	99.85%	100.06%	5	0.70%	3.78s	4	99.79%	100.14%	7
test_10	X	2.28%	10.15s	5	99.79%	100.16%	3	1.80%	10.73s	5	99.89%	100.09%	7
Average	/	6.81%	1.71s	36%	96.27%	103.16%	53%	7.63%	2.01s	30%	96.43%	102.04%	73%

From table 5.12, we can notice that LSWPAe algorithm provides better solutions using the rounding θe_3 compared to θe_1 with a GAP of 4.82% and 81% of best solutions. We can also notice that $P2e^{'}$ is more efficient than $P1e^{'}$ in running time using rounding θe_1 or θe_3.

From table 5.13, we can notice that PLSWPAe algorithm provides better solutions using the rounding θe_2 compared to θe_4 with a GAP of 6.81% and 36% of best solutions. However, PLSWPAe algorithm provides 73% feasible solutions using θe_4 against 53% using θe_2. The energy excess becomes more important in this case, with more than 13% using both roundings

θe_2 and θe_4 for the first instance size. The negative GAP is justified by the fact that PLSWPAe used more energy than the authorized quantity $\mathbb{E}$ ($> 13\%$), thus we obtain a good solution even if we compare to the lower bound.

Comparing tables 5.12 and 5.13, we can notice that LSWPAe is more efficient than PLSWPAe with 81% of the best solutions and a GAP of 4.82%. However, PLSWPAe is more efficient in term of running time, with an average of 1.71 seconds using θe_2 against 87.42 seconds for LSWPAe using θe_1 with $P2e^{'}$, and 2.01 seconds using θe_4 against 87.88 seconds for LSWPAe using θe_3 with $P2e^{'}$.

Finally, LSWPAe is more efficient in energy consumption compared to PLSWPAe, with an average consumption equal to 90.70% using θe_1 against 96.27% for PLSWPAe using θe_2 (feasible solutions), and 92.96% using θe_3 against 96.43% for PLSWPAe using θe_4 (feasible solutions).

Platform 2

Table 5.14: *LSWPAe algorithm evaluation using the list LLP with θe_1 and θe_3 for the platform 2.*

Instances	Time	LSWPAe algorithm using LLP and θe_1						LSWPAe algorithm using LLP and θe_3					
	CPLEX	Gap	*E*-cons	Best	Opt	$P1e^{'}$	$P2e^{'}$	Gap	*E*-cons	Best	Opt	$P1e^{'}$	$P2e^{'}$
test_1	0.91s	22.66%	77.71%	9	4	1.88s	1.62s	20.64%	80.39%	10	4	1.94s	1.67s
test_2	8018.95s	45.41%	89.85%	9	0	20.88s	19.78s	43.73%	92.78%	10	0	21.07s	19.99s
test_3	X	9.14%	98.42%	10	X	41.76s	62.78s	9.14%	98.42%	10	X	42.14s	63.11s
test_4	X	4.94%	99.35%	10	X	52.42s	53.08s	4.94%	99.35%	10	X	52.88s	53.50s
test_5	X	0.25%	99.93%	10	X	75.25s	58.69s	0.25%	99.93%	10	X	76.45s	59.81s
test_6	X	0.02%	99.96%	7	X	74.55s	79.21s	0.02%	99.96%	9	X	75.41s	80.01s
test_7	X	0.01%	99.96%	8	X	147.42s	117.47s	0.01%	99.97%	9	X	148.13s	118.11s
test_8	X	0.01%	99.97%	9	X	104.45s	108.23s	0.01%	99.97%	8	X	105.37s	109.12s
test_9	X	0.01%	99.98%	0	X	129.34s	101.74s	0.01%	99.97%	0	X	130.71s	103.01s
test_10	X	0.02%	99.96%	0	X	51.66s	51.66s	0.02%	99.96%	0	X	53.06s	54.75s
Average	/	8.24%	96.50%	72%	/	69.96s	65.42s	7.87%	97.07%	76%	/	70.71s	66.30s

Table 5.15: *PLSWPAe algorithm evaluation using the list LLP with θe_2 and θe_4 for the platform 2.*

Instances	Time	PLSWPAe algorithm using LLP and θe_2						PLSWPAe algorithm using LLP and θe_4					
	CPLEX	Gap	Time	Best	*E*-cons	$\overline{E}$-cons	feasible	Gap	Time	Best	*E*-cons	$\overline{E}$-cons	feasible
test_1	0.91s	22.16%	0.01s	7	87.02%	109.27%	4	26.28%	0.04s	5	85.45%	108.05%	6
test_2	8018.95s	38.10%	0.06s	7	95.02%	101.96%	3	37.62%	0.20s	7	96.76%	101.98%	3
test_3	X	9.28%	0.13s	6	97.79%	102.12%	5	9.31%	0.39s	7	98.17%	102.33%	9
test_4	X	7.14%	0.17s	3	99.02%	100.97%	6	7.14%	0.48s	3	99.02%	100.97%	6
test_5	X	0.2%	0.55s	5	99.62%	100.68%	6	0.49%	1.41s	3	99.54%	100.06%	9
test_6	X	0.21%	0.63s	2	99.72%	100.26%	4	0.31%	1.10s	0	99.67%	100.08%	8
test_7	X	0.18%	0.64s	1	99.80%	100.26%	6	0.25%	1.10s	1	99.78%	100.18%	8
test_8	X	0.13%	0.87s	1	99.86%	100.15%	5	0.18%	1.43s	0	99.81%	100.15%	7
test_9	X	0.07%	1.92s	0	99.82%	100.11%	5	0.09%	2.85s	0	99.83%	100.10%	6
test_10	X	0.09%	3.49s	0	99.94%	100.12%	7	0.09%	4.10s	0	99.94%	100.12%	7
Average	/	7.75%	0.84s	32%	97.76%	101.59%	51%	8.17%	1.31s	26%	97.79%	101.40%	69%

From table 5.14, we can notice that LSWPAe algorithm provides better solutions using the rounding θe_3 compared to θe_1 with a GAP of 7.87% and 76% of best solutions. We can also notice that $P2e^{'}$ is more efficient than $P1e^{'}$ in running time using rounding θe_1 or θe_3.

From table 5.15, we can notice that PLSWPAe algorithm provides better solutions using the rounding θe_2 compared to θe_4 with a GAP of 7.75% and 32% of best solutions. However, PLSWPAe algorithm provides 69% feasible solutions using θe_4 against 51% using θe_2. The energy excess becomes more important in this case, with more than 08% using both roundings θe_2 and θe_4 for the first instance size.

Comparing tables 5.14 and 5.15, we can notice that LSWPAe is more efficient than PLSWPAe with 76% of best solutions. However, by using more energy than the authorized quantity, PLSWPAe has a better average GAP, with 7.75% obtained using θe_2 against 7.87% for LSWPAe

obtained using θe_3. Furthermore, PLSWPAe is more efficient in term of running time, with an average of 0.84 seconds using θe_2 against 65.42 seconds for LSWPAe using θe_1 with $P2e^{'}$, and 1.31 seconds using θe_4 against 66.30 seconds for LSWPAe using θe_3 with $P2e^{'}$. Finally, LSWPAe is more efficient in energy consumption compared to PLSWPAe, with an average consumption equal to 96.50% using θe_1 against 97.76% for PLSWPAe using θe_2 (feasible solutions), and 97.07% using θe_3 against 97.79% for PLSWPAe using θe_4 (feasible solutions).

For both platforms, we can see that LSWPAe algorithm is impacted by the tightness of the value of $\mathbb{E}$, where the running time become very important (ex: $>$ 4 minutes for an instance of 1000 tasks in the platform 1).

5.6.6 Average

In table 5.16, we give the average of the tables 5.2, 5.4, 5.6, 5.8, 5.10, 5.12 and 5.14. In table 5.17, we give the average of the tables 5.3, 5.5, 5.7, 5.9, 5.11, 5.13 and 5.15.

Table 5.16: *LSWPAe algorithm evaluation using the list LLP with θe_1 and θe_3.*

Instances	LSWPAe algorithm using LLP and θe_1					LSWPAe algorithm using LLP and θe_3				
	Gap	E-cons	Best	$P1e^{'}$	$P2e^{'}$	Gap	E-cons	Best	$P1e^{'}$	$P2e^{'}$
Table 5.2	5.92%	68.32%	44%	11.68s	8.45s	3.75%	69.63%	86%	12.36s	9.05s
Table 5.4	9.73%	75.61%	74%	4.10s	2.69s	9.70%	75.58%	79%	4.68s	3.19s
Table 5.6	5.40%	82.28%	47%	19.48s	24.85s	3.57%	84.85%	84%	20.83s	39.80s
Table 5.8	9.41%	97.16%	84%	5.17s	3.74s	9.23%	97.40%	78%	6.04s	11.30s
Table 5.10	9.53%	75.12%	73%	8.46s	7.42s	9.03%	75.29%	81%	9.32s	7.05s
Table 5.12	6.21%	90.70%	58%	103.5s	87.42s	4.82%	92.96%	81%	104.22s	87.88s
Table 5.14	8.24%	96.50%	72%	69.96s	65.42s	7.87%	97.07%	76%	70.71s	66.30s
Average	7.77%	83.67%	64.57%	31.76s	28.57s	6.85%	84.68%	80.71%	32.59s	32.08 s

From table 5.16, we can notice that in general, LSWPAe algorithm provides better solutions using the rounding θe_3 compared to θe_1 with a GAP of 6.85% and 80.71% of best solutions. We can also notice that $P2e^{'}$ is more efficient than $P1e^{'}$ in running time using rounding θe_1 or θe_3.

Table 5.17: *PLSWPAe algorithm evaluation using the list LLP with θe_2 and θe_4.*

Instances	PLSWPAe algorithm using LLP and θe_2						PLSWPAe algorithm using LLP and θe_4					
	Gap	Time	Best	E-cons	$\overline{E\text{-cons}}$	feasible	Gap	Time	Best	E-cons	$\overline{E\text{-cons}}$	feasible
Table 5.3	6.46%	2.09s	33%	78.53%	103.25%	97%	5.59%	2.43s	50%	78.41%	103.16%	98%
Table 5.5	8.61%	0.60s	63%	76.19%	108.2%	96%	9.20%	0.92s	62%	76.58%	101.45%	98%
Table 5.7	6.65%	0.93s	25%	94.29%	101.52%	64%	5.56%	1.03s	34%	95.29%	101.1%	85%
Table 5.9	8.26%	0.26s	47%	96.95%	102.14%	54%	9.2%	0.37s	45%	98.004%	100.66%	77%
Table 5.11	9.93%	1.81s	62%	76.51%	103.46%	97%	9.97%	2.24s	59%	76.23%	102.07%	98%
Table 5.13	6.81%	1.71s	36%	96.27%	103.16%	53%	7.63%	2.01s	30%	96.43%	102.04%	73%
Table 5.15	7.75%	0.84s	32%	97.76%	101.59%	51%	8.17%	1.31s	26%	97.79%	101.40%	69%
Average	7.78 %	1.17s	42.57%	88.07%	103.3%	73.14%	7.90%	1.47s	43.71%	88.39%	101.69%	85.42%

From table 5.17, we can notice that PLSWPAe algorithm provides better solutions using the rounding θe_2 compared to θe_4 with a GAP of 7.78%. However, PLSWPAe algorithm provides 43.71% of the best solutions and 98% feasible solutions using θe_4 against 42.57% of the best solutions and 73.14% feasible solutions using θe_2.

Comparing tables 5.16 and 5.17, we can notice that LSWPAe is more efficient than PLSWPAe with 80.71% of the best solutions and a GAP of 6.85%. However, PLSWPAe is more efficient in term of running time, with an average of 1.17 seconds using θe_2 against 28.57 seconds for LSWPAe using θe_1 with $P2e^{'}$, and 1.47 seconds using θe_4 against 32.08 seconds for LSWPAe using θe_3 with $P2e^{'}$.

Furthermore, LSWPAe is more efficient in energy consumption compared to PLSWPAe, with an average consumption equal to 83.67% using θe_1 against 88.07% for PLSWPAe using θe_2 (feasible solutions), and 87.68% using θe_3 against 88.39% for PLSWPAe using θe_4 (feasible solutions).

In general, it can be said that both methods LSWPAe and PLSWPAe are effective methods with a general GAP smaller than 8 compared to the lower bound, and thus to the optimal solution. However, PLSWPAe algorithm has the advantage of being a polynomial time method, and can handle large instances with an interesting running time. By adding the energy constraint, we can notice that the running time of LSWPA (LSWPAe by adding the energy constraint) becomes more important for both platforms 1 and 2. However, the average running time of PLSWPAe does not differ too much from PLSWPA since both are polynomial.

From table 5.17, the average over-consumption of energy for PLSWPAe is given by $\overline{E\text{-cons}} = 3.3\%$ using θe_2 and $\overline{E\text{-cons}} = 1.69\%$ using θe_4. A solution to address this excess (if it is considered unacceptable), is to use PLSWPAe algorithm by fixing the amount of energy to $\widehat{\mathbb{E}}$ which must be smaller than $\mathbb{E}$. Thus, PLSWPAe may provide a feasible solution with an energy consumption larger than $\widehat{\mathbb{E}}$ but smaller than $\mathbb{E}$. Otherwise, the value of $\widehat{\mathbb{E}}$ must be decreased until a feasible solution is found.

5.7 Conclusion

This chapter presents two efficient algorithms to solve the problem of scheduling parallel applications on hybrid platforms with communication delays. The objective is to minimize the total execution time (makespan) respecting an energy constraint.

First, we modified LSWPA algorithm to provide a 6-approximation algorithm LSWPAe with two phases: mapping then assignment. For the first phase, two models and two rounding strategies have been proposed for the mapping which respects the energy constraint. In the second phase, a list scheduling algorithm has been proposed to generate a feasible schedule. LSWPAe algorithm guarantees a ratio of 6 compared to the optimal solution using the rounding strategy θe_1. This method was published in [110].

Then, we modified PLSWPA algorithm to provide a polynomial three-phase algorithm PLSWPAe: the first two phases consist in solving linear models to find the type of processor assigned to execute each task. We obtain a mapping that consumes at most twice the amount of authorized energy. In the third phase, we compute the starting execution time of each task to generate a feasible schedule.

Tests on large instances demonstrated the efficiency of our methods and shows the limits of solving the problem with a solver such as *CPLEX*.

We proved by lemma 20 that the energy consumption of PLSWPAe algorithm using the mapping θe_2 is smaller than $2\mathbb{E}^{'}$. However, experimental results showed that on average, the energy excess is not very important (less than $0.1\mathbb{E}^{'}$ in general). Thus, it will be interesting to investigate this rounding strategy to provide another polynomial one, which may respect the energy constraint.

As part of the future, it will be also interesting to study the tightness of LSWPAe (resp. PLSWPAe) algorithm using rounding θe_3 (resp.θe_4) which provide interesting solutions.

An extension to more general heterogeneous platforms with more than two types of processors is also interesting. The challenge is to find a good adaptation of the pre-allocation policy if we have more than two types of processing elements.

Chapter

6 Conclusion

In this thesis, we have studied two of the main challenges for parallel computing: makespan and energy performance. The objective is to find a generic approach to schedule parallel applications presented with graphs of type DAG (Directed Acyclic Graph) on a heterogeneous resource system in order to minimize both the total execution time (makespan) and the energy consumption. For this purpose, we introduced a constraint on the total energy consumed by the platform. We designed many efficient heuristics with a performance guarantee on general problems that were shown NP-complete beforehand. We first gave a brief overview of heterogeneous platforms, in particular fully heterogeneous and hybrid platforms, and the challenges associated with their use. We have shown that the fully heterogeneous platforms represent an interesting solution to execute complex applications, enclosing great computation capability. In particular, the hybrid platforms, which are increasingly used in the field of HPC, combining multi-core processors and hardware accelerators such as GPUs. We then discussed the two main scheduling strategies in the literature, namely dynamic and static scheduling. Then, we have discussed the limits of DVFS and DPM techniques, and the reliability problems related to their use. Our main contributions are stated in the following paragraphs.

As a first contribution, we presented in chapter 2 an efficient approximation scheduling algorithm on uniform machines (Consistent model) for the particular case of linear chains of tasks (sequential applications). Each processing element $p_k \in M$ is characterized by its execution frequency $f_k \geqslant 1$, $k = \overline{1..m}$. Each task t_i is characterized by its weight w_i, $i = \overline{1..n}$. The execution of a task t_i on a processing element p_k generates an execution time $w_{i,k} = \frac{w_i}{f_k}$ and an energy consumption $e_{i,k} = w_i * f_k^2$. The objective is to minimize the total execution time (makespan) respecting an energy constraint on the total energy consumed by the system $(Qm|chain, \mathbb{E}, com|C_{max})$. This model is motivated by the fact that computation platforms can contain processors of the same architecture, but with different versions (as an example, Intel Core$^{\text{TM}}$ i3-4150 Processor and Intel Core$^{\text{TM}}$ i7-6700 Processor). The execution time of the different tasks of the same application (same code with the same programming language, same compiler, identical configuration and so on) on different processors may be relative.

This work has shown that finding the optimal scheduling is not easy. Tests on large instances close to reality shows the limits of solving the problem with a solver such as *CPLEX*. The main contribution of this work is an algorithm which provides a good solution (usually very close to the optimal solution) with small running time, and also guarantees the quality of the solution obtained compared to the optimal solution. The ratio obtained depends on the frequencies of two successive processing elements p_k and p_{k+1} used to provide the an optimal preemptive scheduling. We proved that the ratio between $\widehat{C}_{max}$ the solution obtained by using NPS algorithm 2 and the optimal solution $C^{\star}_{max}$ can be bounded by the ratio between the frequencies of these two processing elements, i.e., $\frac{\widehat{C}_{max}}{C^{\star}_{max}} \leqslant \frac{f_{k+1}}{f_k}$.

Then, we studied the problem of scheduling parallel applications onto a particular case of HPC which is known as hybrid platforms. Specifically, our platform consists of two types ($\mathcal{A}$ and $\mathcal{B}$) of processing elements (e.g. CPU+GPU). For any task t_i, $w_{i,\mathcal{A}}$ (resp. $w_{i,\mathcal{B}}$) is the execution

time of t_i on a processor of type $\mathcal{A}$ (resp. $\mathcal{B}$). In this model, we consider the more general case where the relation between the two types of processing elements can differ for different tasks (inconsistent model). The consistent model represents a particular case of the inconsistent model which considered as a more generic model. We assume that there are no communication delays between the processors of the same type. This assumption is motivated by tightly coupled multiprocessor systems which contain multiple processing elements that are connected and may have access to a central shared memory. This may make the communication cost negligible. The aim is to find a schedule that minimizes the completion time by respecting precedence constraints with communication costs and respecting an energy constraint.

In addition to the theoretical interest of scheduling on unrelated parallel machines, this model is motivated by the fact that computation platforms can contain processors of different architectures (the number of registers, the structure of memory hierarchy, the size of each memory level and so on). Even different applications of the same narrow class may be executed by two different processors at significantly different relative speeds. Moreover, not all processing elements can be programmed in the same language. Several programming models were developed for general computing on graphical processing units (GPUs) like CUDA (Compute Unified Device Architecture) and OpenCL (Open Computing Language).

For platforms composed of an unlimited number of two types of processors, we have shown in Chapter 3 the intractability of the problem by reducing this problem to the 3-satisfiability problem. We proved that the scheduling problem $(P, P)|prec, com|C_{max}$ is NP-Complete even for graphs of depth 3. We also proved that there does not exist 3/2-approximation algorithms unless P=NP. Finally, we provided some polynomial time algorithms for special cases of graphs: Bi-partite graphs, trees and series-parallel graphs.

For hybrid platforms composed of a limited number of two types of processing elements, we presented in Chapter 4 two efficient algorithms that minimize the total execution time $((P\ell, Pk)|prec, com|C_{max})$. First, we proposed a non polynomial 6-approximation algorithm LSWPA with two phases: mapping then assignment. Two models and two rounding strategies have been proposed for the mapping. In the second phase, a list scheduling algorithm has been proposed to generate a feasible schedule using several lists. LSWPA algorithm guarantees a ratio of 6 compared to the optimal solution using the rounding strategy θ_1. Then we proposed a polynomial three-phase algorithm PLSWPA: the first two phases consist in solving linear models to find the type of processor assigned to execute each task. In the third phase, we compute the start execution time of each task to generate a feasible schedule.

Finally, we presented in Chapter 5 two efficient algorithms to solve the problem of scheduling parallel applications on hybrid platforms by adding an energy bound $\mathbb{E}$. The objective is to minimize the total execution time (makespan) respecting an energy constraint $((P\ell, Pk)|prec, com, \mathbb{E}|C_{max})$. First, we modified LSWPA algorithm to provide a 6-approximation algorithm LSWPAe with two phases: mapping then assignment. Two models and two rounding strategies have been proposed for the mapping which respects the energy constraint. In the second phase, a list scheduling algorithm has been proposed to generate a feasible schedule. LSWPAe algorithm guarantees a ratio of 6 compared to the optimal solution using the rounding strategy θe_1. Then, we modified PLSWPA algorithm to provide a polynomial three-phase algorithm PLSWPAe: the first two phases consist in solving linear models to find the type of processor assigned to execute each task. We obtain a mapping that consumes at most twice the amount of authorized energy. In the third phase, we compute the start execution time of each task to generate a feasible schedule. We proved that the energy consumption of PLSWPAe algorithm is smaller than $2\mathbb{E}$. However, experimental results showed that on average, the energy excess is not very important (less than $0.1\mathbb{E}^{'}$ in general).

Tests on large instances demonstrated the efficiency of our methods and shows the limits of solving the problem with a solver such as *CPLEX*. Furthermore, we have given in this thesis great attention to performance guarantees. All the proposed methods guarantee performance in

the worst case (with a fixed or relative ratio).

Throughout the thesis, we pointed out at the end of each chapter some future work that remains to be done. Those, along with the following three main directions for each part form the immediate research that could follow the thesis in the short term.

Chapter 2 presents an efficient approximation algorithm to solve scheduling tasks problem on heterogeneous platforms containing uniforms processing elements for the particular case of linear chains of tasks (sequential applications). The objective is to minimize the total execution time (makespan) respecting an energy constraint on the total energy consumed by the system $(Qm|chain, \mathbb{E}, com|C_{max})$.

The use of the preemptive scheduling has proven to be effective for an application represented by a linear chain of tasks. As part of the future, it could be interesting to focus on more general classes of graphs like DAG. Using the same approach on all graph paths (chains of tasks) could provide an effective method. Furthermore, it could be interesting to test NPS algorithm on real sequential applications en real platforms.

For this model, we consider the more general case where the relation between processing elements can differ for different tasks. Thus we have to take into account that the execution time for any task of the application depends on the processor used to execute it. The aim is to find a schedule on m unrelated machines (two types) that minimizes the completion time by respecting precedence constraints with communication costs and respecting an energy constraint.

In Chapter 3, we treated the problem with unlimited processing elements without energy constraint. There are several extensions that we can see to this work. One direction we are interested by, is a version of this problem where only one type has an unbounded number of resources, and where the data is located on the other one. For example, in the context of smartphone applications, we can model the frontend/backend context where the phone (Machine 1) has a limited number of available processors, but can rely on sending some of the computation on a backend machine (cloud-based), with an unbounded number of processors. Similarly to here, the problem is a data and communication problem: given the cost to transfer data from one machine to the other one, what is the most efficient strategy?. In the context of two unbounded platforms, it would be interesting to find some polynomial time algorithms with proven bounds to the optimal solution. We do not expect to be able to find one in the general case, but we hope that with some constraints between the communication costs and computation cost, one may be able to find such algorithms.

In Chapter 4, we treated the problem with limited processing elements. A proof of the performance guarantee for PLSWPA algorithm was initiated. The ratio between the solution $\widehat{C}_{max1}$ obtained by PLSWPA algorithm and the optimal scheduling solution $C^{\star}_{max}$ is given by $\widehat{C}_{max1} < 6\alpha C^{\star}_{max}$, with $\alpha \geqslant 1$. In future works, it will be interesting to find the value of α to have a fixed bound on the ratio between $\widehat{C}_{max1}$ and $C^{\star}_{max}$.

In Chapter 5, we proposed two algorithms LSWPAe and PLSWPAe to solve the problem with an energy constraint. We proved that the energy consumption of PLSWPAe algorithm using the mapping θe_2 is smaller than $2\mathbb{E}$. However, experimental results showed that on average, the energy excess is not very important (less than $0.1\mathbb{E}^{'}$ in general). Thus, it will be interesting to investigate this rounding strategy to provide another polynomial one, which may respect the energy constraint.

Experimental results showed that LSWPA and PLSWPA algorithms using rounding θ_3 and LSWPAe (resp. PLSWPAe) algorithm using rounding θe_3 (resp.θe_4) provide interesting solutions. Thus, it will be also interesting study the tightness of LSWPA and PLSWPA algorithms using rounding θ_3. It will be also interesting study the tightness of LSWPAe (resp. PLSWPAe) algorithm using rounding θe_3 (resp.θe_4).

An extension to more general heterogeneous platforms with more than two types of processor is also interesting. The different algorithms proposed in the literature do not allow the possibility of finding a performance guarantee. We showed that pre-allocation policy is efficient to deal with all graph paths together, and make the decisions that arrange the greatest number of tasks. This technique gives more possibilities to provide a performance guarantee. The challenge is to find a good adaptation of this policy if we have more than two types of processing elements.

Finally, in this thesis we proposed efficient static methods, so that their association with dynamics methods will give the most efficient solution as possible. It will be interesting to test these methods on real applications, with the association to several dynamic methods to find the most efficient hybrid method.

Bibliography

[1] Linshan Shen and Tae-Young Choe. Posterior task scheduling algorithms for heterogeneous computing systems. In *International Conference on High Performance Computing for Computational Science*, pages 172–183. Springer, 2006.

[2] E Ilavarasan, P Thambidurai, and R Mahilmannan. High performance task scheduling algorithm for heterogeneous computing system. In *ICA3PP*, volume 2005, pages 193–203. Springer, 2005.

[3] Shaikhah AlEbrahim and Imtiaz Ahmad. Task scheduling for heterogeneous computing systems. *The Journal of Supercomputing*, 73(6):2313–2338, 2017.

[4] Anne Benoit, Loïc Pottier, and Yves Robert. Resilient co-scheduling of malleable applications. *The International Journal of High Performance Computing Applications*, 32(1): 89–103, 2018.

[5] Stavan Satish Karia. *Alternative Processor within Threshold: Flexible Scheduling on Heterogeneous Systems*. Rochester Institute of Technology, 2017.

[6] Arman Shehabi, Sarah Smith, Dale Sartor, Richard Brown, Magnus Herrlin, Jonathan Koomey, Eric Masanet, Nathaniel Horner, Inês Azevedo, and William Lintner. United states data center energy usage report. 2016.

[7] George Bosilca, Aurelien Bouteiller, Anthony Danalis, Mathieu Faverge, Thomas Hérault, and Jack J Dongarra. Parsec: Exploiting heterogeneity to enhance scalability. *Computing in Science & Engineering*, 15(6):36–45, 2013.

[8] Nikola Rajovic, Alejandro Rico, Filippo Mantovani, Daniel Ruiz, Josep Oriol Vilarrubi, Constantino Gomez, Luna Backes, Diego Nieto, Harald Servat, Xavier Martorell, et al. The mont-blanc prototype: An alternative approach for hpc systems. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 444–455. IEEE, 2016.

[9] René Griessl, Meysam Peykanu, Jens Hagemeyer, Mario Porrmann, Stefan Krupop, Lars Kosmann, Patrick Knocke, Michal Kierzynka, Ariel Oleksiak, et al. Fpga-accelerated heterogeneous hyperscale server architecture for next-generation compute clusters. 2015.

[10] Lilia Zaourar, Massinissa Ait aba, David Briand, and Jean-Marc Philippe. Task management on fully heterogeneous micro-server system: Modeling and resolution strategies. *Concurrency and Computation: Practice and Experience*, 30(23):e4798, 2018.

[11] Fips consortium. *FiPS project website - http://fips-project.eu/* .

[12] L. Lacassagne, D. Etiemble, A Hassan-Zahraee, A. Dominguez, and P. Vezolle. High level transforms for SIMD and low-level computer vision algorithms. In *ACM Workshop on Programming Models for SIMD/Vector Processing (PPoPP)*, pages 49–56, 2014.

[13] H. Ye, L. Lacassagne, J. Falcou, D. Etiemble, L. Cabaret, and O. Florent. High level transforms to reduce energy consumption of signal and image processing operators. In *IEEE International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, pages 247–254, 2013.

[14] A. Petreto, A. Hennequin, , T. Koehler, T. Romera, Y. Fargeaix, B. Gaillard, M. Bouyer, Q. L. Meunier, and L. Lacassagne. Energy and execution time comparison of optical flow algorithms on SIMD and GPU architectures. In *IEEE International Conference on Design and Architectures for Signal and Image Processing (DASIP)*, pages 1–6, 2018.

[15] A. Petreto, T. Romera, F. Lemaitre, I. Masliah, B. Gaillard, M. Bouyer, Q. L. Meunier, and L. Lacassagne. A new real-time embedded video denoising algorithm. In *IEEE International Conference on Design and Architectures for Signal and Image Processing (DASIP)*, pages 1–6, 2019.

[16] A. Petreto, T. Romera, F. Lemaitre, Manuel Bouyer, B. Gaillard, P. Menard, Q. Meunier, and L. Lacassagne. Real-time embedded video denoiser prototype. In *International Symposium on Optronics in Defense and Security (Optro)*, pages 1–8, 2020.

[17] N. Rambaux, J. Vaubaillon, L. Lacassagne, D. Galayko, G. Guignan, M. Birlan, M. Capderou, F. Colas, F. Deleflie, F. Deshours, A. Hauchecorne, P. Keckhut, A.C. Levasseurd-Regourd, J.L. Rault, and B. Zanda. Meteorix: a cubesat mission dedicated to the detection of meteors and space debris. In *ESA Space Safety Programme Office, NEO and Debris Detection Conference (ESA NDDC)*, pages 1–9, 2019.

[18] Hassan Barada, Sadiq M Sait, and Naved Baig. Task matching and scheduling in heterogeneous systems using simulated evolution. 2001.

[19] Shoukat Ali, Jong-Kook Kim, Howard Jay Siegel, Anthony A Maciejewski, Yang Yu, Shriram B Gundala, Sethavidh Gertphol, and Viktor K Prasanna. Greedy heuristics for resource allocation in dynamic distributed real-time heterogeneous computing systems. In *PDPTA*, pages 519–530, 2002.

[20] Oliver Sinnen. *Task scheduling for parallel systems*, volume 60. John Wiley & Sons, 2007.

[21] Emmanuel Agullo, Berenger Bramas, Olivier Coulaud, Eric Darve, Matthias Messner, and Toru Takahashi. Task-based fmm for heterogeneous architectures. *Concurrency and Computation: Practice and Experience*, 28(9):2608–2629, 2016.

[22] Cédric Augonnet, Samuel Thibault, Raymond Namyst, and Pierre-André Wacrenier. Starpu: a unified platform for task scheduling on heterogeneous multicore architectures. *Concurrency and Computation: Practice and Experience*, 2011.

[23] Olivier Beaumont, Lionel Eyrauld-Dubois, Abdou Guermouche, and Thomas Lambert. Comparison of static and runtime resource allocation strategies for matrix multiplication. In *2015 27th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 170–177. IEEE, 2015.

[24] Emmanuel Agullo, Olivier Beaumont, Lionel Eyraud-Dubois, and Suraj Kumar. Are static schedules so bad? a case study on cholesky factorization. In *Parallel and Distributed Processing Symposium*, pages 1021–1030. IEEE, 2016.

[25] Lionel Eyraud-Dubois and Thomas Lambert. Using static allocation algorithms for matrix matrix multiplication on multicores and gpus. In *ICPP 2018-47th International Conference on Parallel Processing*, 2018.

[26] Haluk Topcuoglu, Salim Hariri, and Min-you Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE transactions on parallel and distributed systems*, 13(3):260–274, 2002.

[27] Abdessamad Ait El Cadi, Omar Souissi, Rabie Ben Atitallah, Nicolas Belanger, and Abdelhakim Artiba. Mathematical programming models for scheduling in a cpu/fpga architecture with heterogeneous communication delays. *Journal of Intelligent Manufacturing*, 29(3):629–640, 2018.

[28] R Giroudeau and JC König. Scheduling with communication delays. In *Multiprocessor scheduling, theory and applications*. IntechOpen, 2007.

[29] Jeffrey D. Ullman. Np-complete scheduling problems. *Journal of Computer and System sciences*, 10(3):384–393, 1975.

[30] Damien Prot and Odile Bellenguez-Morineau. A survey on how the structure of precedence constraints may change the complexity class of scheduling problems. *Journal of Scheduling*, 21(1):3–16, 2018.

[31] Ronald L Graham, Eugene L Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of discrete mathematics*, volume 5, pages 287–326. Elsevier, 1979.

[32] Peter Brucker. Scheduling algorithms. new york: Springer. 2007.

[33] Sergio Nesmachnow, Bernabé Dorronsoro, Johnatan E Pecero, and Pascal Bouvry. Energy-aware scheduling on multicore heterogeneous grid computing systems. *Journal of grid computing*, 11(4):653–680, 2013.

[34] Jing Mei and Kenli Li. Energy-aware scheduling algorithm with duplication on heterogeneous computing systems. In *Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing*, pages 122–129. IEEE Computer Society, 2012.

[35] Qingjia Huang, Sen Su, Jian Li, Peng Xu, Kai Shuang, and Xiao Huang. Enhanced energy-efficient scheduling for parallel applications in cloud. In *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, pages 781–786. IEEE, 2012.

[36] Sanjeev Baskiyar and Kiran Kumar Palli. Low power scheduling of dags to minimize finish times. In *International Conference on High-Performance Computing*, pages 353–362. Springer, 2006.

[37] Nirmal Kaur, Savina Bansal, and Rakesh Kumar Bansal. Towards energy efficient scheduling with dvfs for precedence constrained tasks on heterogeneous cluster system. In *2015 2nd International Conference on Recent Advances in Engineering & Computational Sciences (RAECS)*, pages 1–6. IEEE, 2015.

[38] Yan Ma, Bin Gong, and Lida Zou. Energy-efficient scheduling algorithm of task dependent graph on dvs-unable cluster system. In *2009 10th IEEE/ACM International Conference on Grid Computing*, pages 217–224. IEEE, 2009.

[39] Vishnu Swaminathan and Krishnendu Chakrabarty. Pruning-based, energy-optimal, deterministic i/o device scheduling for hard real-time systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 4(1):141–167, 2005.

[40] Johnson D.S. Garey, M.R. Computers and intractability: A guide to the theory of np-completeness. pages 238–239. W.H. Freeman and Co., 1979.

[41] Guillaume Pallez (Aupy) and Anne Benoit. Approximation algorithms for energy, reliability, and makespan optimization problems. *Parallel Processing Letters*, 26(01):1650001, 2016.

[42] Guoqi Xie, Xiongren Xiao, Renfa Li, and Keqin Li. Schedule length minimization of parallel applications with energy consumption constraints using heuristics on heterogeneous distributed systems. *Concurrency and Computation: Practice and Experience*, 2016.

[43] Young Choon Lee and Albert Y Zomaya. Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling. In *Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on*, pages 92–99. IEEE, 2009.

[44] Vasanth Venkatachalam and Michael Franz. Power reduction techniques for microprocessor systems. *ACM Computing Surveys (CSUR)*, 37(3):195–237, 2005.

[45] Gong Chen, Wenbo He, Jie Liu, Suman Nath, Leonidas Rigas, Lin Xiao, and Feng Zhao. Energy-aware server provisioning and load dispatching for connection-intensive internet services. In *NSDI*, volume 8, pages 337–350, 2008.

[46] Nirmal Kaur, Savina Bansal, and Rakesh Kumar Bansal. Duplication-controlled static energy-efficient scheduling on multiprocessor computing system. *Concurrency and Computation: Practice and Experience*, 29(12):e4124, 2017.

[47] Etienne Le Sueur and Gernot Heiser. Dynamic voltage and frequency scaling: The laws of diminishing returns. In *Proceedings of the 2010 international conference on Power aware computing and systems*, pages 1–8, 2010.

[48] Adam Manzanares. *Energy Efficient Pre-Fetching-Models to Implementation.* PhD thesis, 2010.

[49] Milad Ghorbani Moghaddam, Alexandre Yamamoto, and Cristinel Ababei. Investigation of dvfs based dynamic reliability management for chip multiprocessors. In *2015 International Conference on High Performance Computing & Simulation (HPCS)*, pages 563–568. IEEE, 2015.

[50] Jack Dongarra and Alexey L Lastovetsky. *High performance heterogeneous computing*, volume 78. John Wiley & Sons, 2009.

[51] Wieslaw Kubiak. Optimal scheduling of unit-time tasks on two uniform processors under tree-like precedence constraints. *Zeitschrift für Operations Research*, 33(6):423–437, 1989.

[52] Wei-Chang Yeh, Mei-Chi Chuang, and Wen-Chiung Lee. Uniform parallel machine scheduling with resource consumption constraint. *Applied Mathematical Modelling*, 39(8):2131–2138, 2015.

[53] Nvidia. embedded systems developer kits and modules. online at http://www.nvidia.com/object/embeddedsystemsdev-kits-modules.html.

[54] John E Stone, David Gohara, and Guochun Shi. Opencl: A parallel programming standard for heterogeneous computing systems. *Computing in science & engineering*, 12(3):66, 2010.

[55] Min Ji and TC Edwin Cheng. An fptas for parallel-machine scheduling under a grade of service provision to minimize makespan. *Information Processing Letters*, 108(4):171–174, 2008.

[56] Christos Koulamas and George J Kyparisis. A modified lpt algorithm for the two uniform parallel machine makespan minimization problem. *European Journal of Operational Research*, 196(1):61–68, 2009.

[57] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[58] Min Ji, Jen-Ya Wang, and Wen-Chiung Lee. Minimizing resource consumption on uniform parallel machines with a bound on makespan. *Computers & Operations Research*, 40(12): 2970–2974, 2013.

[59] Eugene Leighton Lawler. Preemptive scheduling of. precedence-constrained jobs on parallel machines. In *Deterministic and stochastic scheduling*, pages 101–123. Springer, 1982.

[60] Teofilo Gonzalez and Sartaj Sahni. Preemptive scheduling of uniform processor systems. *Journal of the ACM (JACM)*, 25(1):92–101, 1978.

[61] Peter Brucker, Johann Hurink, and Wieslaw Kubiak. Scheduling identical jobs with chain precedence constraints on two uniform machines. *Mathematical methods of operations research*, 49(2):211–219, 1999.

[62] Fabián A Chudak and David B Shmoys. Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds. *Journal of Algorithms*, 30(2):323–343, 1999.

[63] Wieslaw Kubiak, Bernard Penz, and Denis Trystram. Scheduling chains on uniform processors with communication delays. *Journal of Scheduling*, 5(6):459–476, 2002.

[64] Te C Hu. Parallel sequencing and assembly line problems. *Operations research*, 9(6): 841–848, 1961.

[65] Vadim G Timkovsky. Identical parallel machines vs. unit-time shops and preemptions vs. chains in scheduling complexity. *European Journal of Operational Research*, 149(2): 355–376, 2003.

[66] Chandra Chekuriy and Michael Benderz. An efficient approximation algorithm for minimizing makespan on uniformly related machines. *Optimization (IPCO)*, page 1, 1998.

[67] Gerhard J Woeginger. A comment on scheduling on uniform machines under chain-type precedence constraints. *Operations Research Letters*, 26(3):107–109, 2000.

[68] Klaus Jansen and Roberto Solis-Oba. Approximation algorithms for scheduling jobs with chain precedence constraints. In *International Conference on Parallel Processing and Applied Mathematics*, pages 105–112. Springer, 2003.

[69] IBM. Ibm ilog cplex v12.5 user's manual for cplex, http://www.ibm.com. 2013.

[70] Massinissa Ait Aba, Lilia Zaourar, and Alix Munier. Approximation algorithm for scheduling a chain of tasks on heterogeneous systems. In *In Proceedings of the 23rd International European Conference on Parallel and Distributed Computing (Euro-Par'17), Parallel Processing Workshops*, pages 353–365. Springer, 2017.

[71] M Asch, T Moore, R Badia, M Beck, P Beckman, T Bidot, F Bodin, F Cappello, A Choudhary, B de Supinski, et al. Big data and extreme-scale computing: Pathways to convergence-toward a shaping strategy for a future software and data ecosystem for scientific inquiry. *The International Journal of High Performance Computing Applications*, 32(4):435–479, 2018.

[72] Xi Yang, Chengkun Wu, Kai Lu, Lin Fang, Yong Zhang, Shengkang Li, Guixin Guo, and YunFei Du. An interface for biomedical big data processing on the tianhe-2 supercomputer. *Molecules*, 22(12), 2017. ISSN 1420-3049. doi: 10.3390/molecules22122116.

[73] Ioan Raicu, Ian T Foster, and Yong Zhao. Many-task computing for grids and supercomputers. In *Many-Task Computing on Grids and Supercomputers, 2008. MTAGS 2008. Workshop on*, pages 1–11. IEEE, 2008.

[74] Csanad Imreh. Scheduling problems on two sets of identical machines. *Computing*, 70(4): 277–294, 2003.

[75] Louis-Claude Canon, Loris Marchal, and Frédéric Vivien. Low-Cost Approximation Algorithms for Scheduling Independent Tasks on Hybrid Platforms. In *Euro-Par 2017: 23rd International European Conference on Parallel and Distributed Computing*, Santiago de Compostela, Spain, August 2017. Springer. URL https://hal.inria.fr/hal-01559898.

[76] Safia Kedad-Sidhoum, Florence Monna, Grégory Mounié, and Denis Trystram. A family of scheduling algorithms for hybrid parallel platforms. *International Journal of Foundations of Computer Science*, 29(01):63–90, 2018.

[77] Safia Kedad-Sidhoum, Florence Monna, and Denis Trystram. Scheduling tasks with precedence constraints on hybrid multi-core machines. In *IPDPSW*, pages 27–33. IEEE, 2015.

[78] Marcos Amaris, Giorgio Lucarelli, Clément Mommessin, and Denis Trystram. Generic algorithms for scheduling applications on hybrid multi-core machines. In *European Conference on Parallel Processing*, pages 220–231. Springer, 2017.

[79] TH Cormen, CE Leiserson, RL Rivest, and C Stein. Introduction to algorithms, chap. 16. *Greedy Algorithms. MIT Press, Cambridge*, 2001.

[80] Rashmi Bajaj and Dharma P Agrawal. Improving scheduling of tasks in a heterogeneous environment. *IEEE Transactions on Parallel and Distributed Systems*, 15(2):107–118, 2004.

[81] Jean-Yves Colin and Philippe Chrétienne. Cpm scheduling with small communication delays and task duplication. *Operations Research*, 39(4):680–684, 1991.

[82] Sekhar Darbha and Dharma P Agrawal. Optimal scheduling algorithm for distributed-memory machines. *IEEE transactions on parallel and distributed systems*, 9(1):87–95, 1998.

[83] Chan-Ik Park and Tae-Young Choe. An optimal scheduling algorithm based on task duplication. In *Parallel and Distributed Systems, 2001. ICPADS 2001. Proceedings. Eighth International Conference on*, pages 9–14. IEEE, 2001.

[84] Annie S Wu, Han Yu, Shiyuan Jin, K-C Lin, and Guy Schiavone. An incremental genetic algorithm approach to multiprocessor scheduling. *IEEE Transactions on parallel and distributed systems*, 15(9):824–834, 2004.

[85] Yu-Kwong Kwok and Ishfaq Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys (CSUR)*, 31(4):406–471, 1999.

[86] Denis Barthou and Emmanuel Jeannot. Spaghetti: Scheduling/placement approach for task-graphs on heterogeneous architecture. In *Euro-Par 2014 Parallel Processing*, pages 174–185. Springer International Publishing, 2014.

[87] Cristina Boeres, Vinod EF Rebello, et al. A cluster-based strategy for scheduling task on heterogeneous processors. In *Computer Architecture and High Performance Computing. SBAC-PAD 2004. 16th Symposium on*, pages 214–221. IEEE, 2004.

[88] Tao Yang and Apostolos Gerasoulis. Dsc: Scheduling parallel tasks on an unbounded number of processors. *IEEE Transactions on Parallel and Distributed Systems*, 5(9):951–967, 1994.

[89] Michael R Garey and David S. Johnson. Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal on Computing*, 4(4):397–411, 1975.

[90] Luiz F Bittencourt, Rizos Sakellariou, and Edmundo RM Madeira. Dag scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm. In *Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on*, pages 27–34. IEEE, 2010.

[91] Minhaj Ahmad Khan. Scheduling for heterogeneous systems using constrained critical paths. *Parallel Computing*, 38(4-5):175–193, 2012.

[92] Hamid Arabnejad and Jorge G Barbosa. List scheduling algorithm for heterogeneous systems by an optimistic cost table. *IEEE Transactions on Parallel and Distributed Systems*, 25(3):682–694, 2014.

[93] Sunita Kushwaha and Sanjay Kumar. An investigation of list heuristic scheduling algorithms for multiprocessor system. *IUP Journal of Computer Sciences*, 11(2), 2017.

[94] Huijun Wang and Oliver Sinnen. List-scheduling vs. cluster-scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 2018.

[95] Jan Karel Lenstra, David B Shmoys, and Eva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical programming*, 46(1-3):259–271, 1990.

[96] Rodolphe Giroudeau, Jean-Claude Konig, Farida Kamila Moulai, and Jérôme Palaysi. Complexity and approximation for precedence constrained scheduling problems with large communication delays. *Theoretical Computer Science*, 401(1-3):107–119, 2008.

[97] Tao Yang and Apostolos Gerasoulis. A fast static scheduling algorithm for dags on an unbounded number of processors. In *Proceedings of the 1991 ACM/IEEE conference on Supercomputing*, pages 633–642. ACM, 1991.

[98] Olivier Beaumont, Louis-claude Canon, Lionel Eyraud-Dubois, Giorgio Lucarelli, Loris Marchal, Clément Mommessin, Bertrand Simon, and Denis Trystram. Scheduling on two types of resources: a survey. *arXiv preprint arXiv:1909.11365*, 2019.

[99] Donald E Knuth. *Art of computer programming, volume 2: Seminumerical algorithms*. Addison-Wesley Professional, 2014.

[100] Gary Gordon and Elizabeth McMahon. A greedoid polynomial which distinguishes rooted arborescences. *Proceedings of the American Mathematical Society*, 107(2):287–298, 1989.

[101] Robert E. Tarjan Valdes, Jacobo and Eugene L. Lawler. The recognition of series parallel digraphs. *SIAM Journal on Computing*, 11(2):298–313, 1982. doi: 10.1137/0211023. URL https://doi.org/10.1137/0211023.

[102] B. Schoenmakers. *A new algorithm for the recognition of series parallel graphs*. CWI report. CS-R. CWI, 1995.

[103] Massinissa Ait aba, Guillaume Pallez (Aupy), and Alix Munier-Kordon. Scheduling on two unbounded resources with communication costs. In *In International European Conference on Parallel and Distributed Computing (Euro-Par)*, 2019.

[104] Massinissa Ait aba, Guillaume Pallez (Aupy), and Alix Munier-Kordon. Scheduling on Two Unbounded Resources with Communication Costs. Research Report RR-9264, Inria, March 2019. URL https://hal.inria.fr/hal-02076473.

[105] Henan Zhao and Rizos Sakellariou. An experimental investigation into the rank function of the heterogeneous earliest finish time scheduling algorithm. In *European Conference on Parallel Processing*, pages 189–194. Springer, 2003.

[106] Minhaj Ahmad Khan. Task scheduling for heterogeneous systems using an incremental approach. *The Journal of Supercomputing*, 73(5):1905–1928, 2017.

[107] Massinissa Ait aba, Lilia Zaourar, and Alix Munier. Approximation algorithm for scheduling applications on hybrid multi-core machines with communications delays. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 36–45. IEEE, 2018.

[108] Bruno Bodin, Youen Lesparre, Jean-Marc Delosme, and Alix Munier-Kordon. Fast and efficient dataflow graph generation. In *Proceedings of the 17th International Workshop on Software and Compilers for Embedded Systems*. ACM, 2014.

[109] Michael R Garey and David S Johnson. Computers and intractability. a guide to the theory of np-completeness. a series of books in the mathematical sciences, 1979.

[110] Massinissa Ait aba, Lilia Zaourar, and Alix Munier. Efficient algorithm for scheduling parallel applications on hybrid multi-core machines with communications delays and energy constraint. *Concurrency and Computation: Practice and Experience*, 2019.