Master of Science in Informatics at Grenoble
Master Mathématiques Informatique - spécialité Informatique
Option Parallel, Distributed and Embedded Systems

Scheduling Parallel Programs on Hybrid Machines

Mommessin Clément

June 24, 2016

Research project performed at INRIA-Grenoble

Under the supervision of:
Prof. D. Trystram, Grenoble INP
Dr. G. Lucarelli, Grenoble INP

Defended before a jury composed of:
Prof. N. Depalma, University Grenoble Alpes
Prof. M. Heusse, Grenoble INP
Prof. C. Berrut, University Grenoble Alpes
Prof. O. Gruber, University Grenoble Alpes
Prof. S. Kedad, University Paris 6

June 2016

Abstract

In this paper, we are interested in scheduling an application, *i.e.*, a set of tasks linked by precedence relations, on an heterogeneous parallel machine composed of classical processors (CPUs) with accelerators (GPUs). Such architectures are commonly used in modern high-performance computing platforms. The objective is to minimize the completion time of the entire application, denoted by *makespan*.

We focus in this work on the design of generic methods for scheduling any application, in opposition to most existing dedicated methods developed for specific structures. First, we analyse the recent approximation algorithm proposed by Kedad-Sidhoum *et al.* [10] which achieves a ratio of 6 for solving this problem. This approach is based on a two-phases method where the first phase is based on a rounding of the solution provided by a linear programming formulation for determining the assignment of the tasks to the resources. The second phase uses a classical greedy list algorithm to schedule the tasks according to this assignment. We first construct an instance which shows that the 6-approximation ratio achieved by the proposed algorithm is tight.

Then, we introduce some new constraints in the linear program to improve the assignment step. However, the weakness of the existing algorithm relies mainly in the policy used in the second phase. Thus, we propose a new rule inspired by the ranking function of the well-known HEFT algorithm for improving the second phase. Nevertheless, we show that the approximation ratio is still equal to 6.

We also extend the model for $Q \geq 2$ heterogeneous types of computing units. We generalize the new algorithm defined for $Q = 2$ and we prove that it achieves a $Q(Q+1)$-approximation ratio, which is tight.

An experimental study is reported for studying the practical behavior of the various methods. For this purpose, we developed a new benchmark based on actual library components of Chameleon. The set of experiments shows that the performance of the linear program-based algorithms are in practice much better than the theoretical upper bound of 6. Moreover, our modified algorithm outperforms the original one, while it is, on average, competitive to HEFT.

Résumé

Dans ce papier, nous nous intéressons à l'ordonnancement d'une application, *i.e.*, un ensemble de tâches liées par des relations de précédence, sur une machine parallèle et hétérogène composée de processeurs traditionnels (CPUs) et d'accélérateurs de calculs (GPUs). De telles architectures sont beaucoup employées dans les plateformes modernes de calculs à haute performance. L'objectif est de minimiser le temps de complétion de l'application dans son ensemble, noté *makespan*.

Nous nous concentrons dans ce travail sur la conception de méthodes génériques pour l'ordonnancement d'une application quelconque, en opposition à la plupart des méthodes qui sont dédiées à l'ordonnancement de structures spécifiques. Premièrement, nous analysons le récent algorithme d'approximation proposé par Kedad-Sidhoum *et al.* [10] qui atteint un rapport d'approximation de 6 pour la résolution de ce problème. Cette approche est basée sur une méthode à deux étapes. La première phase est basée sur l'arrondissement de la solution d'un programme linéaire pour déterminer l'affectation d'une tache à une ressource. La seconde phase utilise un algorithme de liste glouton pour ordonnancer les tâches suivant les affectations. Nous construisons tout d'abord une instance du problème qui montre que le ratio d'approximation à 6 de l'algorithme est strict.

Ensuite, nous définissons des nouvelles contraintes pour le programme linéaire pour améliorer l'étape d'affectation. Toutefois, la faiblesse dans l'algorithme existant vient principalement de la méthode utilisée dans la seconde phase. Ainsi, nous proposons une nouvelle règle inspirée de la fonction de classement de l'algorithme HEFT pour améliorer la seconde étape. Néanmoins, nous montrons que le rapport d'approximation de l'algorithme reste égal à 6.

Nous étendons également le modèle pour $Q \geq 2$ types hétérogènes d'unités de calculs. Nous généralisons le nouvel algorithme défini pour $Q = 2$ et nous prouvons que l'algorithme généralisé atteint un strict rapport d'approximation de $Q(Q+1)$.

Une étude expérimentale est menée pour étudier le comportement des différentes méthodes en pratique. Pour cela, nous avons développé un benchmark basé sur les composants réels de la librairie Chameleon. L'ensemble des expériences montre que les performances des algorithmes basés sur des programmes linéaires sont en pratique bien meilleurs que les bornes supérieures théoriques à 6. De plus, notre algorithme modifié surpasse l'algorithme original et est, en moyenne, comparatif à HEFT.

Contents

— 1 —
Introduction

Since the emergence of vector machines in the late seventies, the scheduling of tasks that compose an application has always been an important issue in the domain of High Performance Computing. The evolution of parallel platforms with the increase of the number of nodes on one side, the augmentation of heterogeneity within a node on the other side, tend to complexify more and more the distributed architectures, and as a consequence, render their use less efficient. Since the current scheduling policies are not sufficient, we need to adapt to the new landscape.

Originally, a node was only composed of one or several computing units and, then, the number of components within a node increased to become more diversified with CPUs, accelerators and, more recently, specialized units for analytics, I/O manipulations and checkpointing.

During the last decade, the development of graphical processor units (GPUs) made possible the acceleration of computations by executing some part of the code (the most regular one) of an application on one or more GPU. The programmers first designed their applications to take into account the speedup that can be offered by GPUs and tuned their own code directly deciding which part of the application should be executed on a GPU instead of a classical CPU. Specific scheduling algorithms obtained by *ad-hoc* algorithms have also been proposed to determine the assignment of each job on a CPU or a GPU in order to optimise a specific type of application, such as biological sequence comparison [11].

Since the last 5 years, more general and heuristic algorithms have been proposed for scheduling applications on hybrid machines composed of multiple CPUs and GPUs. These algorithms are defined for any parallel application and some of them provide worst case performance guarantees on the produced schedule.

1.1 Problem Definition

In this work, we are interested in the problem of efficiently scheduling dependent tasks of an application on a hybrid parallel platform composed of several identical CPUs and GPUs, in order to minimize the overall completion time of the application (also called *makespan*). This optimization problem can be summarized in the two following questions: given a set of tasks, each one characterized by different processing times on CPU and GPU, on which CPU or which

GPU and at what time each task should start its execution in order to minimize the completion time of the last finishing task?

The assumption about known processing times is relevant since the platforms where applications are executed usually have models to estimate the execution times of the tasks.

As the number of computing resources increases to thousands and even hundreds thousands of cores, achieving good efficiency in executing large scale applications is an important issue and is to be more and more complicated in the race to exascale[1]. The problem of designing algorithms to efficiently schedule the tasks of a parallel application on the various processors of a platform has been extensively studied and is still a big challenge nowadays.

As parallel and distributed platforms are growing, the sizes of applications that are scheduled on these platforms are also growing in size and a space exploration of all possible schedules to find the optimal one for a given application is too costly, in terms of time, to be used.

Thus, we are looking for generic algorithms with good performances and limited time complexity (polynomial in the size of the problem). Another interesting feature is to design algorithms with bounded worse case guarantees in term of approximation ratio.

1.2 Main Constributions and Outline

Our principal objective is to design a generic algorithm for the scheduling problem on hybrid CPU/GPU environments.

First, we study the heuristic algorithm HEFT and we improve the lower bound on its approximation ratio by giving an example in which the solution of HEFT is $(1-\frac{1}{e})m$ times far from the optimal solution.

We then study the different steps of the algorithm presented by Kedad-Sidhoum *et al.* [10], denoted by HLP, and propose a worst case example, proving that the approximation ratio of the algorithm has tight bounds. We also propose a refinement of the linear program used in the first step as well as a new scheduling method, with a ranking function inspired by HEFT, to assign each task to a processor following the assignment given in the previous step. We explain that the new defined algorithm has an approximation ratio of at most 6 and provide a worst case example showing that the approximation ratio is also bounded below by 6, with any List Scheduling method, concluding by the tightness of the ratio.

Then, we extend the addressed problem to parallel platforms composed of multiple types resources. We define a generic algorithm for scheduling dependent tasks on Q different sets of identical resources and prove that $Q(Q+1)$ is an upper bound for the approximation ratio of the algorithm. We strongly believe that the ratio is tight with a worst case example derived from the case where $Q=2$ but no study is given here.

Finally, we apply the original algorithm HLP, the new defined algorithm and HEFT to a set of 6 applications of linear algebra with dense matrix and compare the produced schedules. The experiments shows that, although our modified algorithms have better performances than HLP, HEFT remains the best algorithm in practice. However, as we said before, HEFT does not guarantee any worst-case ratio.

[1] *i.e.*, when parallel platforms will achieve a computing power of 10^{18} FLOPS (Floating-point Operations per Second)

The outline of this report is as follows. In Chapter 2 we give a formal definition of the addressed problem and explain the notations that are used in the other Chapters, as well as a brief example of the problem. Chapter 3 presents a state of the art of the addressed problem, as well as some other models of scheduling problems.

The HEFT algorithm is presented in Chapter 4. Chapter 5 shows the studies made on the algorithm HLP and the new algorithm is defined in Chapter 6. The algorithm defined for the extension of the problem to Q resources is presented in Chapter 7.

The results on the experiments and comparisons of the algorithms are presented in Chapter 8. Finally, we conclude and discuss about the future work in Chapter 9.

— 2 —
Preliminaries

In this Chapter, we give a formal definition of the problem and the model of the parallel architecture considered. We also introduce all the notations and conventions that are used in the following Chapters and we provide a simple example of the problem.

2.1 Problem Definition

We consider the problem of scheduling a parallel application consisted in a set of sequential tasks which are linked by precedence constraints on a hybrid platform composed of multiple identical CPUs and GPUs.

We model the parallel architecture as a single machine composed of m identical CPUs and k identical GPUs, assuming no hierarchy, no data management nor communication delays among any pair of processors. We also omit the management of the GPUs which is done by some external CPUs. Notice that the number of processors m and k are also part of the problem instance.

An instance of the application to be scheduled on this environment is a set T of n tasks. Each task is denoted by T_j, with the index j going from 1 to n, and has two processing times, $\overline{p_j}$ and $\underline{p_j}$, depending whether the task is executed on a CPU or a GPU, respectively. The tasks are linked by precedence constraints modeled by a Directed Acyclic Graph $G = (V, E)$, where each node $j \in V$ corresponds to the task T_j and each arc $(i, j) \in E$ corresponds to a precedence relation between the tasks T_i and T_j. In other words, the task T_j cannot start its execution before the completion of all its predecessors T_i with $(i, j) \in E$. For simplicity, we may refer to a task only by its index j instead of T_j. Moreover, the use of the words *job* and *task* is done interchangeably.

The objective is to minimize the length of the schedule, known as *makespan* and denoted as $C_{\max}$. Specifically, the makespan corresponds to the completion time of the last task on any processor (assuming that all tasks are available at time zero). A critical assumption is that the preemption of tasks is not allowed, i.e., its task is executed without interruptions.

Following the Graham's 3-fields notation for scheduling problems, the above problem can be denoted by $(P_1, P_2) \mid prec \mid C_{\max}$, where P_1 and P_2 correspond to the set of identical CPUs and identical GPUs, respectively. Note that our problem is a special case of the problem of scheduling on unrelated machines, denoted by $R \mid prec \mid C_{max}$, for which no constant factor approximation algorithm is known. On the other hand, if each task has the same processing time on all CPUs and GPUs (i.e., $\overline{p_j} = \underline{p_j}$), $(P_1, P_2) \mid prec \mid C_{\max}$ reduces to the problem of scheduling on a single set of $m+k$ identical machines, denoted by $P \mid prec \mid C_{max}$, which is known to be $\mathcal{NP}$-hard. Hence, the $\mathcal{NP}$-hardness of our problem is directly implied since it generalizes $P \mid prec \mid C_{max}$.

We also extend our problem to the problem $(P_1,\dots,P_Q) \mid prec \mid C_{\max}$ of scheduling on Q different types of resources, instead of only 2 types (CPUs and GPUs). In this case, we denote by $p_{j,q}$ the processing time of a task T_j if it is executed on a processor of type q. Let also M_q be the number of identical processors of type q. This is an even harder to approximate problem which however is still a special case of $R \mid prec \mid C_{max}$ since each type of resources contains several identical processors.

2.2 Notations and Other Definitions

In general, we denote by C_{max} the makespan of a schedule created by an algorithm, while OPT denotes the makespan of an optimal schedule. A task is called *ready* at a time t if it has not yet been scheduled but all of its predecessors are already completed at t.

We call a task T_i *predecessor*, resp. *successor*, of T_j if there exists an arc $(i,j) \in E$, resp. $(j,i) \in E$, in the directed acyclic graph G. Moreover, a task T_i is called *ancestor* of T_j if there exists a directed path from i to j in G. We denote by $\Gamma^-(j)$, $\Gamma^+(j)$ and $A(j)$ the sets of predecessors, successors and ancestors of a task T_j, respectively.

For simplicity of presentation, we occasionally consider that the input instance contains two fictive tasks T_{start} and T_{end}, whose processing times are set equal to zero for any type of resources. Moreover, we add an arc of precedence between the node corresponding to T_{start} and every node that has no predecessor as well as between every node that has no successor and the node corresponding to T_{end}.

In what follows, we use the notion of *approximation ratio* in order to describe the performance of an algorithm. The approximation ratio ρ of an algorithm is defined as the maximum ratio, over all instances I of the problem, between the objective value of the schedule created by the algorithm and the objective value of an optimal schedule. More specifically, we have:

$$\rho = \max_I \left\{ \frac{C_{max}(I)}{OPT(I)} \right\}$$

In the case where the instance is not known in advance by the algorithm, we say that we are in the *online* mode. More specifically, we consider that the tasks arrive online the one after the other together with their characteristics (processing times, predecessors) and the algorithm has to irrevocably decide at the arrival of each task on which processor and during which period it will be executed. The performance of an online algorithm is measured by an adaptation of the approximation ratio, known as *competitive ratio*, which is defined as the maximum ratio between the objective value of the schedule produced by the online algorithm and the objective value of an optimal off-line schedule, over all instances.

2.3 An Example

In this section we give a simple example to show the execution of tasks in an hybrid CPU/GPU environment. Our example also shows that the well-known *List Scheduling* policy introduced by Graham [7] which achieves a constant-factor approximation ratio for $P \mid prec \mid C_{max}$, can produce a schedule with arbitrarily bad performance for our problem. List Scheduling is a greedy algorithm that does not introduce unnecessary idle times and *whenever a processor becomes idle, it selects for executing the first ready task.*

Consider an application with three tasks which has to be processed on a single CPU and a single GPU. The task T_3 is a successor of both T_1 and T_2, while T_1 and T_2 are independent. The processing times of the tasks are as follows:

$$\begin{array}{lll} \overline{p_1} = 2 & \text{and} & \underline{p_1} = 1 \\ \overline{p_2} = 10 & \text{and} & \underline{p_2} = 1 \\ \overline{p_3} = 1 & \text{and} & \underline{p_3} = 1 \end{array}$$

Assume that the List Scheduling policy will schedule first the task T_1 on the GPU, prioritizing the processor of the smallest processing time for it. Then, the CPU processor is idle at time zero and hence the task T_2 is scheduled on it. When both tasks have completed, the task T_3 will be scheduled on the GPU. The constructed schedule is shown in Figure 2.1.

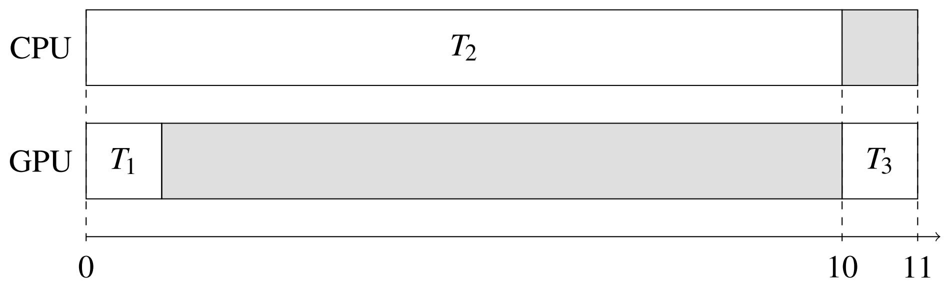

Figure 2.1: Schedule of the List Scheduling algorithm. Note that the gray areas correspond to (mandatory) idle times.

As we can see in this example, the choice of not introducing unnecessary idle times in the schedule is not well-suited for the scheduling on hybrid platforms. In our example, the task T_2 is misplaced on the idle CPU by the List Scheduling algorithm but it is easy to see that in the optimal schedule it should be executed on the GPU. Concluding, this example shows that the choice of the type of resources for the execution of each task is critical when scheduling in non-identical platforms.

— 3 —
State of the art

Scheduling tasks of a parallel program is an old topic and there exist a lot of studies from the 1970s and the first vector machines. Since then, the subject received a particular attention with the successive generations of parallel and distributed machines and, during the last decade, the emergence of hardware accelerators.

In this chapter, we describe the state of the art of the addressed problem by presenting general theoretical results in both off-line and on-line mode as well as general heuristics for scheduling independent and dependent tasks on identical, hybrid or unrelated parallel machines. Notice that all the studies presented here are for the makespan minimization problem when the preemption of tasks is not allowed, all processing times are deterministically defined and no communication costs exist between tasks, unless it is clearly specified.

3.1 Identical Platforms

One of the first and most known result in scheduling theory is Graham's List Scheduling algorithm [7] for the problem $P \mid prec \mid C_{max}$ of scheduling tasks with precedence constraints on a set of identical machines. Note that the List Scheduling algorithm is in fact an online algorithm. The competitive ratio of this algorithm is $2 - \frac{1}{m}$, where m is the number of processors. Svensson [18] has showed that it is $\mathcal{NP}$-hard to improve this ratio assuming a new variant of the unique games conjecture, even in the case of unit processing times.

In the case of independent tasks ($P \mid\mid C_{max}$), Graham [8] has also proposed an off-line variant of List Scheduling where the tasks are presented to the algorithm by non-increasing order of their processing time, known as the *Largest Processing Time* (LPT) policy. The approximation ratio of this algorithm is $\frac{4}{3} - \frac{1}{3m}$ and it is tight. Hochbaum and Shmoys [9] reduce this ratio by proposing a *Polynomial Time Approximation Scheme* (PTAS) using a dual approximation approach. Specifically, for any $\varepsilon > 0$, they describe a $(1+\varepsilon)$-approximation algorithm with a running time in $O((\frac{n}{\varepsilon})^{\frac{1}{\varepsilon^2}})$. Moreover, two faster algorithms have been also presented for values of ε close to $\frac{1}{5}$ and $\frac{1}{6}$.

The online problem $P \mid r_j - online \mid C_{max}$ where a set of independent tasks that arrive over time has to be scheduled on a set of identical processors, has been also studied in the bibliography. In this online model, we assume that a task T_j arrives at an unknown time r_j together with its characteristics. The algorithm cannot modify the execution of tasks that is already done by the time r_j; however, it can modify the schedule after time r_j. For this problem, Chen and

Vestjens [4] proposed an adaptation of LPT, achieving a tight competitive ratio of $\frac{3}{2}$. They have also provided an absolute lower bound of 1.3473 for the competitive ratio of this problem.

3.2 Unrelated Platforms

In the case of unrelated platforms, where tasks have a different processing time on each processor, Lenstra *et al.* [13] proposed a linear program and a rounding method to schedule a set of independent tasks off-line ($R \mid\mid C_{max}$). They showed that the described algorithm achieves an approximation ratio of 2 and they proved that it is $\mathcal{NP}$-hard to have a polynomial algorithm with a ratio strictly less than $\frac{3}{2}$. The authors also proposed a PTAS in the case of a fixed number of processors, i.e., the complexity of this algorithm is exponential to the number of processors. Shchepin and Vakhania [16] proposed a better rounding method used with the linear program of Lenstra *et al.* [13] which improves the approximation ratio of the algorithm from 2 to $2 - \frac{1}{m}$. This is so far the best known ratio for this problem.

For the on-line version of the problem ($R \mid prec \mid C_{max}$), Azar and Epstein [2] studied the model of scheduling tasks with precedence constraints, where a task becomes known only when all its predecessors have been completed. They showed a lower bound on the competitive ratio of $\Omega(\sqrt{m})$, for both known and unknown processing times.

Many heuristics have been also designed for the off-line scheduling problem with precedence constraints on unrelated processors. Unlike the previous algorithms with worst-case approximation guarantees, these heuristics take also into account communication costs between tasks. The addressed problem is denoted by $R \mid prec, c_{jk} \mid C_{max}$, where c_{jk} is the additional communication cost if the tasks T_j and T_k, where $(j,k) \in E$, are not executed on the same processor. Sakellariou and Zhao [15] proposed a hybrid heuristic taking into account both the processing times of the tasks and the structure of the graph of precedences, while the well-known Heterogeneous Earliest Finish Time (HEFT) algorithm, presented by Topcuoglu *et al.* [19], uses a ranking method to order the tasks before scheduling. Several other heuristics are also presented in the literature (see for example [1, 12]).

3.3 Hybrid Platforms

Since the emergence of hardware accelerators, algorithms have been designed for scheduling tasks on hybrid machines composed of identical processors (CPU) and one or several accelerators (GPU). Kedad-Sidhoum *et al.* presented two off-line algorithms for scheduling independent and dependent tasks on m CPUs and k GPUs. The first algorithm [3] is based on dynamic programming and gives an approximation ratio of $\frac{4}{3} - \frac{1}{3k}$ for the case of independent tasks; notice that this ratio is independent on the number of CPUs. The second algorithm [10] is the first generic method that achieves a constant-factor approximation ratio for scheduling tasks linked by precedence constraints on hybrid platforms. Its approximation ratio is 6 and, for the rest of the work, we will call this algorithm HLP (Heterogeneous Linear Program). Note also that a counter-example showing that HEFT cannot have an approximation ratio better than $\frac{m}{2}$ has been provided in [3].

For the on-line case, Chen *et al.* [5] proposed the first algorithm for scheduling independent tasks over a list, achieving a competitive ratio of 3.85. Two other algorithms are proposed in the case where there is an equal number of CPUs and GPUs and for the case where there is only one GPU achieving better competitive ratios. The authors also show that there exist no algorithm with a competitive ratio smaller than 2 for this problem and that the classical List Scheduling algorithm does not provide any performance guarantee on hybrid machines.

If we increase the degree of heterogeneity, other algorithms have been proposed for scheduling tasks on platforms with $Q \geq 2$ different types of resources, where each type is composed of several identical processing units. Gehrke *et al.* [6] presented a PTAS for scheduling independent tasks if the number of resource types is fixed.

Moreover, Liu and Liu [14] studied the restricted assignment model, where a task can only be processed on a particular type of processors. The authors showed that the approximation ratio obtained by any list scheduling algorithm adapted to Q different types of resources is at most $1 + Q - \min_q(\frac{1}{M_q})$, where M_q is the number of identical processors of type q, with $q = 1, \cdots, Q$. This result is consistent with Graham's List Scheduling of ratio $2 - \frac{1}{m}$ with only one type of processors.

Finally, Shmoys *et al.* [17] presented a method to transform an off-line algorithm for scheduling tasks on parallel systems of any type to an algorithm for the same problem in on-line mode with release dates for the tasks. They proved that the competitive ratio of the produced algorithm is no more than twice the approximation ratio of the off-line algorithm used.

— 4 — The HEFT Algorithm

In this chapter, we present HEFT (Heterogeneous Earliest Finishing Time), a heuristic algorithm proposed by Topcuoglu *et al.* [19] for the problem of scheduling tasks on unrelated processors, taking into account precedence relations and communications between tasks. In the following, we describe the algorithm and we propose an example slightly improving the lower bound for HEFT by Kedad-Sidhoum *et al.* [3].

4.1 The Algorithm

The HEFT algorithm is defined to construct a schedule using a simple heuristic and is composed of two steps:

1. Task Prioritizing: The priority rank of each task is computed and the list of tasks is sorted by decreasing order of the ranks.
2. Processor Selection: Tasks are removed one by one from the list and are scheduled on the processor which minimizes the finishing time of the task.

The algorithm is defined for the model of unrelated machines, where a task may have as many processing times as the number of processors, with additional communication costs if the tasks linked by precedence are not executed on the same processor. Let $\gamma_p(j)$ be the average of the processing times of a task T_j over all processors, *i.e.*, $\gamma_p(j) = \frac{\sum_{i=1}^{m} p_{i,j}}{m}$. The communication cost incurred by the precedence constraint between the tasks T_i and T_j, with T_i executed on processor m_i and T_j on m_j, is denoted by $c_{i,j}$ and the average among all communication costs, with any combination of m_i and m_j, is denoted by $\gamma_c(i,j)$. Notice that the communication cost is reduced to zero if tasks T_i and T_j are executed on the same processor.

The ranks are recursively computed, starting from the tasks with no successors, as follows:

$$rank(T_j) = \gamma_p(j) + \max_{i \in \Gamma^+(j)} \{\gamma_c(i,j) + rank(T_i)\} \quad \forall j \in V$$

HEFT considers the tasks in non-increasing order of their ranks. For each task T_j in this order, HEFT determines the pair of processor and starting time which minimizes the completion time of T_j with respect to the already created schedule for the previous tasks of the order, and schedules appropriately T_j. Notice that a task can be placed between two already scheduled tasks if the idle period is sufficient to process the task, taking into account the communication costs, if any. Moreover, based on the way of computation of ranks, each task is scheduled after all its ancestors. Hence, the schedule created by HEFT is feasible.

4.2 A Counter-Example of at Least $(1-\frac{1}{e})m$

In this section, we present an example slightly improving the lower bound for the worst case ratio of HEFT.

In order to do this, we propose a simplified instance of independent tasks to be scheduled on m identical processors (CPUs) and one processor different from the others (GPU). For this instance, the algorithm will produce a schedule with the value of the resulting makespan being a function of m while the makespan of the optimal solution is constant.

Theorem 4.2.1 *The approximation ratio of the HEFT algorithm is at least* $(1-\frac{1}{e})(m+1)+\frac{1}{m}$.

Recall that, in the case of only two different types of processors (CPU and GPU), we denote by $\overline{p_j}$ and $\underline{p_j}$ the processing times of the task T_j on a CPU and a GPU, respectively, so that $p_{i,j} = \overline{p_j}$ for $i = 1,\cdots,m$ and $p_{m+1,j} = \underline{p_j}$. The set of tasks of the instance is as follows, with i going from 1 to $m-1$:

Type	Number of task	Processing time on CPU/GPU
A_0	1	$1/1$
B_0	m	$1/\frac{1}{m^2}$
A_i	1	$\frac{m^i}{m(m+1)^{i-1}} - \frac{1}{m}/\frac{m^i}{m(m+1)^{i-1}} - \frac{1}{m}$
B_i	m	$\frac{m^i}{m(m+1)^{i-1}} - \frac{1}{m}/\frac{1}{m^2}$

It is easy to check that every processing time is greater than zero. We consider independent tasks, hence all communication costs are zero. The rank of each task is then the average computation time of the task over all processors and is calculated as follows:

$$rank(T_j) = \frac{m\overline{p_j} + \underline{p_j}}{m+1}$$

The rank of tasks A_0 and A_i, $\forall i = 1,\cdots,m-1$ are clearly:

$$rank(A_0) = 1$$

$$rank(A_i) = \frac{m^i}{m(m+1)^{i-1}} - \frac{1}{m} = \frac{m^i - (m+1)^{i-1}}{m(m+1)^{i-1}}$$

The rank of tasks B_0 and B_i, $\forall i = 1,\cdots,m-1$ are as follows:

$$rank(B_0) = \frac{m+\frac{1}{m^2}}{m+1} = \frac{m^3+1}{m^2(m+1)}$$

$$rank(B_i) = \frac{\frac{m^i}{(m+1)^{i-1}} - 1 + \frac{1}{m^2}}{(m+1)} = \frac{m^i - (m+1)^{i-1}}{(m+1)^i} + \frac{1}{m^2(m+1)}$$

4.2.1 Task Ordering

Before scheduling, the algorithm sorts the tasks in decreasing order of their rank.

Proposition 4.2.2 *For the proposed instance, the order of the tasks created by HEFT is as follows:*

$$A_0 \rightarrow B_0 \rightarrow A_1 \rightarrow B_1 \rightarrow \cdots \rightarrow A_i \rightarrow B_i \rightarrow A_{i+1} \rightarrow \cdots \rightarrow B_{m-1}$$

Proof:
For the order on the tasks A_0 and B_0:

$$\begin{aligned} rank(A_0) &\geq rank(B_0) \\ 1 &\geq \frac{m^3+1}{m^2(m+1)} \\ m^3+m^2 &\geq m^3+1 \end{aligned}$$

For the tasks B_0 and A_1:

$$\begin{aligned} rank(B_0) &\geq rank(A_1) \\ \frac{m^3+1}{m^2(m+1)} &\geq \frac{m-1}{m} \\ m^3+1 &\geq (m^2+m)(m-1) \\ m^3+1 &\geq m^3-m \end{aligned}$$

For the tasks A_i and B_i, $\forall i = 1, \cdots, m-1$:

$$\begin{aligned} rank(A_i) &\geq rank(B_i) \\ \frac{m^i-(m+1)^{i-1}}{m(m+1)^{i-1}} &\geq \frac{m^i-(m+1)^{i-1}}{(m+1)^i} + \frac{1}{m^2(m+1)} \\ m(m+1)(m^i-(m+1)^{i-1}) &\geq m^2(m^i-(m+1)^{i-1})-(m+1)^{i-1} \\ m^{i+2}+m^{i+1}-m^2(m+1)^{i-1}-m(m+1)^{i-1} &\geq m^{i+2}-m^2(m+1)^{i-1}-(m+1)^{i-1} \\ m^{i+1}-m(m+1)^{i-1} &\geq -(m+1)^{i-1} \\ m^{i+1} &\geq (m-1)(m+1)^{i-1} \end{aligned}$$

For the tasks B_i and A_{i+1}, $\forall i = 1, \cdots, m-2$:

$$\begin{aligned} rank(B_i) &\geq rank(A_{i+1}) \\ \frac{m^i-(m+1)^{i-1}}{(m+1)^i} + \frac{1}{m^2(m+1)} &\geq \frac{m^{i+1}-(m+1)^i}{m(m+1)^i} \\ m^{i+2}-m^2(m+1)^{i-1}+(m+1)^{i-1} &\geq m^{i+2}-m(m+1)^i \\ (1-m^2)(m+1)^{i-1} &\geq -m(m+1)(m+1)^{i-1} \\ 1-m^2 &\geq -m^2-m \end{aligned}$$

Therefore, the order claimed over the ranks is correct and the Proposition 4.2.2 follows. □

4.2.2 Task Scheduling

To schedule a task, the algorithm selects the processor which minimizes the completion time of that task. Let assume that in case of a tie, the algorithm selects in priority the GPU before any CPU and chooses between two CPUs arbitrarily. Notice that, in our case, all tasks are independent and then the scheduling method does not introduce idle times in the schedule. The schedule can then be constructed by knowing the list of tasks assigned to each processor.

The algorithm will schedule the task A_0 on the GPU and the m tasks B_0 on the m CPUs. The task A_1 will then be scheduled on the GPU and tasks B_1 on the CPUs and so on until all tasks are scheduled. In the resulting schedule, all tasks of type A are on the GPU and all tasks of type B are on the m CPUs. All processors become idle at the same time and the makespan is as follows:

$$\begin{aligned}
C_{max} &= 1 + \sum_{i=1}^{m-1} \frac{m^i}{m(m+1)^{i-1}} - \frac{1}{m} = 1 - \frac{m-1}{m} + \sum_{i=1}^{m-1} \frac{m^{i-1}}{(m+1)^{i-1}} \\
&= \frac{1}{m} + \sum_{j=0}^{m-2} (\frac{m}{m+1})^j = \frac{1}{m} + \frac{1-(\frac{m}{m+1})^{m-1}}{1-\frac{m}{m+1}} \\
&= \frac{1}{m} + \frac{1-(\frac{m}{m+1})^{m-1}}{\frac{1}{m+1}} = \frac{1}{m} + (m+1)(1-(\frac{m}{m+1})^{m-1})
\end{aligned}$$

Taking a value of m sufficiently large we have $(\frac{m}{m+1})^{m-1} \to \frac{1}{e}$ and the value of C_{max} tends to $\frac{1}{m} + (m+1)(1-\frac{1}{e})$. Thus, we have:

$$C_{max} = O(m)$$

The optimal makespan of the instance is achieved when all tasks of type A are on the m CPUs and all tasks of type B are on the GPU. The resulting makespan is $OPT = 1$.

The left side of Figure 4.1 shows a schedule produced by the algorithm HEFT while the right side shows the optimal schedule of the given instance.

Summarizing, we can deduce that there is an instance for which HEFT has an approximation ratio at least $(1-\frac{1}{e})(m+1) + \frac{1}{m}$, completing the proof of Theorem 4.2.1.

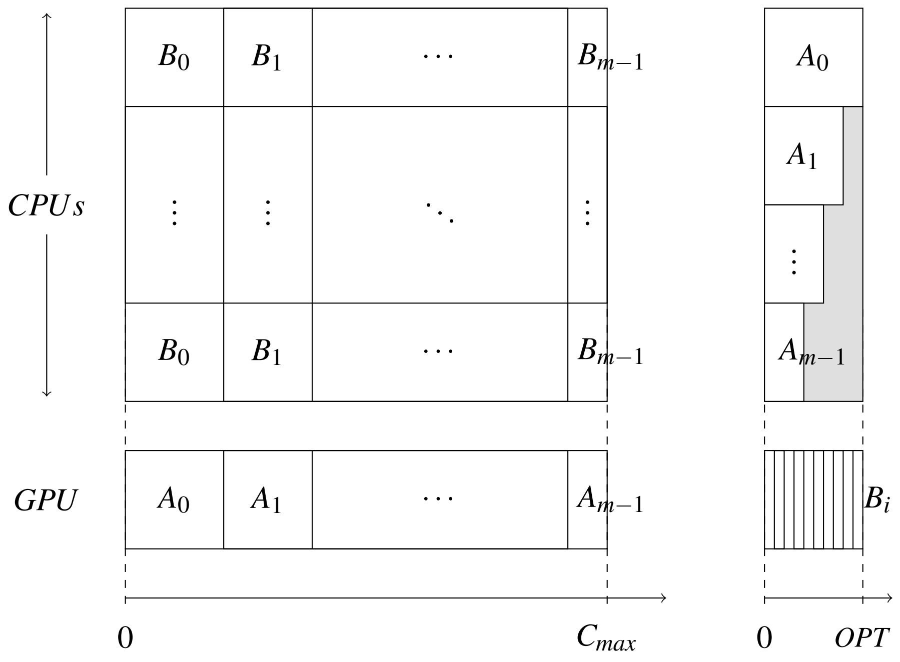

Figure 4.1: Possible schedule of HEFT (left) and optimal schedule (right). Notice that the gray area represents idle time.

— 5 —

The HLP Algorithm

In this chapter, we study the 6-approximation algorithm HLP presented by Kedad-Sidhoum *et al.* [10] designed for the problem of scheduling dependent tasks on hybrid platforms and we propose a worst case example showing that its approximation ratio is tight.

5.1 The Algorithm

In order to solve the addressed scheduling problem, the HLP algorithm proceeds in two main steps:

- Assignment Step: Each task is assigned to a type of processor, either CPU or GPU, by solving a Linear Program and by rounding the obtained fractional optimal solution to an integral one.
- Scheduling Step: Each task is scheduled with a variant of the List Scheduling algorithm, taking into account the assignment obtained in the previous step.

5.1.1 Assignment Step

To solve the assignment problem, Kedad-Sidhoum *et al.* [10] proposed an integer linear program that provides a lower bound to the optimal makespan based on the two classical lower bounds: the total load of each type of processors and the critical path. The critical path is the path from the *start* node to the *end* node which have the maximal total processing time and that cannot be reduced, no matter how many processors are considered. In order to do this, they defined a binary assignment variable for each task T_j as follows:

$$x_j = \begin{cases} 1 & \text{if } T_j \text{ is processed on a CPU} \\ 0 & \text{otherwise} \end{cases}$$

Moreover, for each task T_j, they introduced a variable C_j representing the completion time of T_j. Finally, let λ be a variable corresponding to the lower bound of the makespan.

The integer linear program is defined as follows:

minimize λ subject to

$$C_i + \overline{p_j} x_j + \underline{p_j}(1 - x_j) \leq C_j \qquad \forall j = 1, \cdots, n,\ \forall i \in \Gamma^-(j) \tag{5.1}$$

$$C_j \leq \lambda \qquad \forall j = 1, \cdots, n \tag{5.2}$$

$$\sum_{j=1}^{n} \overline{p_j} x_j \leq m\lambda \tag{5.3}$$

$$\sum_{j=1}^{n} \underline{p_j}(1 - x_j) \leq k\lambda \tag{5.4}$$

$$x_j \in \{0, 1\} \qquad \forall j = 1, \cdots, n \tag{5.5}$$

Constraints 5.1 represent the precedence relation between two tasks, imposing that a task cannot start its execution before the completion of all its predecessors. Constraints 5.2 are used to lower bound the makespan λ, which is the maximum completion time among all the tasks. In this way, Constraints 5.1 and 5.2 imply that the makespan cannot be smaller than the critical path. Constraints 5.3 and 5.4 correspond to the load. They impose that the total work load of all the tasks executed on the CPUs, resp. GPUs, is not greater than $m\lambda$, resp. $k\lambda$. Finally, Constraints 5.5 are the integrality constraints, ensuring that a task is executed either on a CPU or on a GPU.

Since an optimal solution for the above program cannot be computed in polynomial time, we relax the integrality constraints. The assignment variables x_j are now allowed to be real values between 0 and 1, corresponding to the assignment of a fraction of the task T_j to a CPU and the other fraction to a GPU.

The relaxed linear program, denoted by $(LP1)$, is as follows:

minimize λ subject to

$$C_i + \overline{p_j} x_j + \underline{p_j}(1 - x_j) \leq C_j \qquad \forall j = 1, \cdots, n,\ \forall i \in \Gamma^-(j) \tag{5.6}$$

$$C_j \leq \lambda \qquad \forall j = 1, \cdots, n \tag{5.7}$$

$$\sum_{j=1}^{n} \overline{p_j} x_j \leq m\lambda \tag{5.8}$$

$$\sum_{j=1}^{n} \underline{p_j}(1 - x_j) \leq k\lambda \tag{5.9}$$

$$x_j \in [0, 1] \qquad \forall j = 1, \cdots, n \tag{5.10}$$

Once an optimal solution of $(LP1)$ is computed, each assignment variable is rounded to either 1 or 0, in order to have each task fully assigned to either a CPU or a GPU, respectively. The fractional assignment variables of the optimal solution are denoted by x_j^R, $\forall j = 1, \cdots, n$. For each T_j, the rounding policy is as follows:

$$x_j = \begin{cases} 1 & \text{if } x_j^R \geq \frac{1}{2} \\ 0 & \text{otherwise} \end{cases}$$

5.1.2 Scheduling Step

Once the assignment has been defined for all tasks, a variant of List Scheduling is used. Specifically, at each time where a CPU, resp. GPU, is idle, the algorithm schedules on it one of the ready tasks assigned to the CPU resources, resp. GPU resources, at the previous step.

We denote by S the schedule produced by the algorithm and by α the assignment of the tasks obtained by the previous step. The scheduling method used is as follows:

Algorithm 1

```
1: $S \leftarrow \emptyset$
2: while $S \neq T$ do
3:     Let $R \leftarrow \{T_j \mid \Gamma^-(j) \subseteq S\}$ be the set of ready tasks
4:     Compute the earliest possible starting time for all tasks in $R$ with respect to the precedence constraints and the assignment $\alpha$
5:     Schedule the task $T_j \in R$ with the smallest possible starting time
6:     $S \leftarrow S \cup \{T_j\}$
```

5.2 Worst Case Example

In this section, we propose an instance of the problem for which the algorithm HLP builds a schedule with a makespan being close to 6 times greater than the makespan of the optimal solution, leading to the following theorem:

Theorem 5.2.1 *The approximation ratio of HLP is at least 6.*

5.2.1 Instance of the Problem

Consider a parallel system with an equal number m of CPUs and GPUs, with m large enough. The list of tasks T, with the graph of precedence between the tasks shown in Figure 5.1, is defined as follows:

Type	Number of tasks	Processing time on CPU/GPU
A	1	$1 \;/\; \infty$
B	1	$\infty \;/\; 1$
C	$2(m+1)(m-2)$	$\frac{1}{2m-1} \;/\; 1$
D	1	$2m-1 \;/\; 1$
E	1	$\infty \;/\; m+2$
J	$2(m+1)(m-1)$	$1 \;/\; \frac{1}{2m-1}$

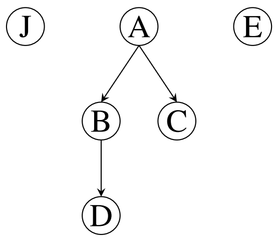

Figure 5.1: Precedence graph of the tasks

5.2.2 Solving the Linear Program

Proposition 5.2.2 *A possible optimal solution of $(LP1)$ gives $\lambda = m+2$ and the fractional assignment variables, with $\varepsilon > 0$:*

$$\begin{cases} x_A &= 1 \\ x_B &= 0 \\ x_C &= \frac{1}{2} - \varepsilon \\ x_D &= \frac{1}{2} \\ x_E &= 0 \\ x_J &= \frac{1}{2} \end{cases}$$

Proof:
It is easy to see that no solution with $\lambda < m+2$ is possible since the task E is forced to be scheduled on a GPU with a processing time $m+2$. Thus, $\lambda \geq m+2$.

We then have to make sure that all constraints of $(LP1)$ are verified in order to prove that $\lambda \leq m+2$.

For constraint 5.6:

$$\begin{aligned}
&C_A = \overline{p_A} x_A + \underline{p_A}(1-x_A) = 1 \\
&C_B = C_A + \overline{p_B} x_B + \underline{p_B}(1-x_B) = 2 \\
&C_C = C_A + \overline{p_C} x_C + \underline{p_C}(1-x_C) = 1 + \frac{1}{2m-1}(\frac{1}{2}-\varepsilon) + (\frac{1}{2}+\varepsilon) = \frac{3}{2} + \frac{1}{4m-2} + \varepsilon - \frac{\varepsilon}{2m-1} \\
&C_D = C_B + \overline{p_D} x_D + \underline{p_D}(1-x_D) = 2 + \frac{2m-1}{2} + \frac{1}{2} = m+2 \\
&C_E = \overline{p_E} x_E + \underline{p_E}(1-x_E) = m+2 \\
&C_J = \overline{p_J} x_J + \underline{p_J}(1-x_J) = \frac{1}{2} + \frac{1}{2m-1}\frac{1}{2} = \frac{2m}{4m-2}
\end{aligned}$$

For constraint 5.7, the relation $C_j \leq m+2, \forall j$ is straightforward since $\varepsilon > 0$.

For constraint 5.8:

$$\begin{aligned}
m\lambda &\geq \sum_{j=1}^{n} \overline{p_j} x_j \\
m(m+2) &\geq \overline{p_A} x_A + 2(m+1)(m-2)\overline{p_C} x_C + \overline{p_D} x_D + 2(m+1)(m-1)\overline{p_J} x_J \\
m^2+2m &\geq 1 + 2(m+1)(m-2)\frac{1}{2m-1}(\frac{1}{2}-\varepsilon) + (2m-1)\frac{1}{2} + 2(m+1)(m-1)\frac{1}{2} \\
m^2+2m &\geq 1 + \frac{2(m^2-m-2)}{2m-1}(\frac{1}{2}-\varepsilon) + \frac{2m-1}{2} + m^2 - 1 \\
m^2+2m &\geq m^2 + m - \frac{1}{2} + \frac{m^2-m-2}{2m-1} - 2\varepsilon\frac{m^2-m-2}{2m-1} \\
m &\geq \frac{m^2-m-2}{2m-1} - \frac{1}{2} - 2\varepsilon\frac{m^2-m-2}{2m-1} \\
m(2m-1) &\geq m^2 - m - 2 - \frac{2m-1}{2} - 2\varepsilon(m^2-m-2) \\
2m^2 - m &\geq m^2 - 2m - \frac{3}{2} - 2\varepsilon(m^2-m-2) \\
0 &\leq m^2 + m + \frac{3}{2} + 2\varepsilon(m^2-m-2)
\end{aligned}$$

The last inequality is true since m is large enough and $\varepsilon > 0$.

For constraint 5.9:

$$\begin{aligned}
k\lambda &\geq \sum_{j=1}^{n} \underline{p_j}(1-x_j) \\
m\lambda &\geq \underline{p_B}(1-x_B) + 2(m+1)(m-2)\underline{p_C}(1-x_C) + \underline{p_D}(1-x_D) + \underline{p_E}(1-x_E) \\
&\quad + 2(m+1)(m-1)\underline{p_J}(1-x_J) \\
m(m+2) &\geq 1 + 2(m+1)(m-2)(\frac{1}{2}+\varepsilon) + \frac{1}{2} + (m+2) + 2(m+1)(m-1)\frac{1}{2m-1}\frac{1}{2} \\
m^2+2m &\geq \frac{7}{2} + m + (m^2-m-2) + 2\varepsilon(m^2-m-2) + \frac{m^2-1}{2m-1} \\
2m &\geq \frac{3}{2} + 2\varepsilon(m^2-m-2) + \frac{m^2-1}{2m-1} \\
2m(2m-1) &\geq \frac{6m-3}{2} + 2\varepsilon(m^2-m-2)(2m-1) + m^2 - 1 \\
4m^2 - 2m &\geq 3m - \frac{3}{2} + 2\varepsilon(2m^3-3m^2-3m+2) + m^2 - 1 \\
0 &\leq 3m^2 - 5m + \frac{5}{2} - 2\varepsilon(2m^3-3m^2-3m+2)
\end{aligned}$$

The last inequality is true since m is large enough and $\varepsilon > 0$.
Finally, the constraint 5.10 is easily checked and thus we have $\lambda \leq m+2$.

Since $\lambda \leq m+2$, $\lambda \geq m+2$ and every constraints are satisfied, we can conclude that the claimed proposition 5.2.2 is true. □

5.2.3 Rounding and Scheduling Phases

After solving the linear program, the fractional assignment variables are rounded to the values:

$$\begin{cases} x_A &= 1 \\ x_B &= 0 \\ x_C &= 0 \\ x_D &= 1 \\ x_E &= 0 \\ x_J &= 1 \end{cases}$$

Figure 5.2 shows a possible output schedule for the variant of List Scheduling used. The order on the list of tasks considered for the List Scheduling is $T = \{E, J, A, C, B, D\}$, where the multiple tasks of the types C and J are arbitrarily ordered.

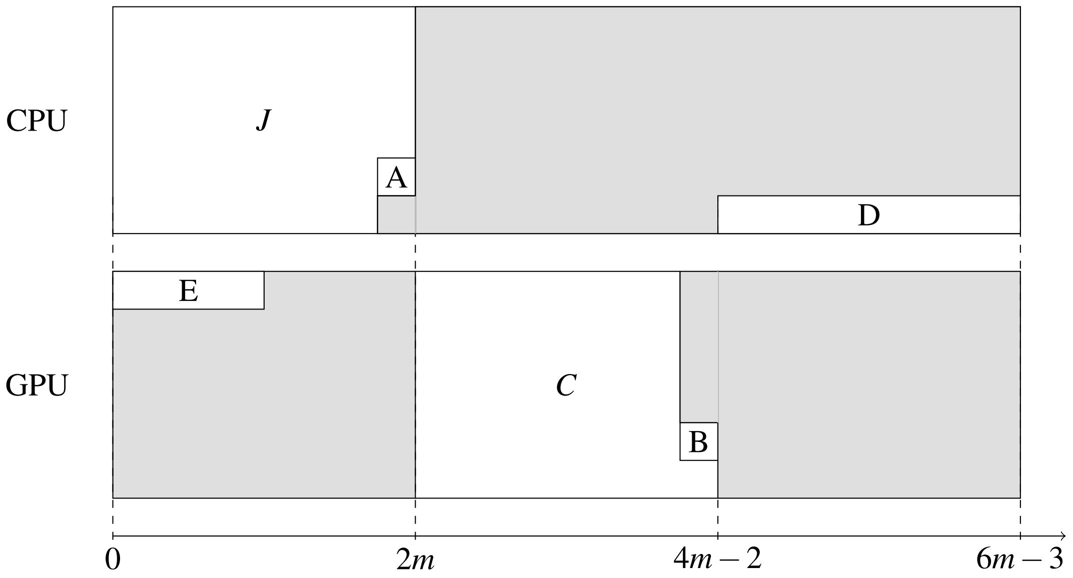

Figure 5.2: Possible schedule for the described instance. Notice that the gray areas represent idle times

We can observe that the resulting makespan is $C_{max} = 6m - 3$, compared to the optimal value $m + 2$ we have the following ratio:

$$\begin{aligned} \rho &= \frac{6m-3}{m+2} \\ &= 6 - \frac{15}{m+2} \end{aligned}$$

Thus, we can conclude that the approximation ratio of HLP is bounded below by 6, validating Theorem 5.2.1.

Combining this result with the Theorem 6 given by Kedad-Sidhoum, we deduce the following statement:

Corollary 5.2.3 *The algorithm HLP has a tight approximation ratio of 6.*

— 6 —
Modifications on HLP

In Kedad-Sidhoum *et al.* [10], it is proved that the used policy of rounding the assignment variables is best possible. Moreover, the counter-example that we gave in the previous chapter is based on the fact that the scheduling algorithm can consider the tasks in an arbitrary bad order non respecting the critical path. For these reasons, we modified the constraints of the linear programs and the scheduling method in order to improve the schedules constructed by the HLP algorithm, as well as its approximation ratio. However, even with this refined and more complicated algorithm, denoted by refined HLP, we cannot achieve an approximation ratio better than 6.

6.1 Modifications of the Linear Program

In this section, we refine the set of constraints of the linear program of HLP, described in Section 5.1.1, in order to give a better lower bound for the assignment problem. The new described linear program is denoted by $(LP2)$.

Recall that we denote by $A(j)$ the set of ancestors of the task T_j and that the extended graph $G = (V,E)$ contains a starting task T_{start} and an ending task T_{end} as described in Section 2.2.

In order to refine the linear program we modify the third and fourth constraints of $(LP1)$, which are two constraints of load on CPUs and on GPUs, to create two constraints of partial load on CPUs and on GPUs for each task. Each task has a non-empty set of ancestors, containing at least the initial task T_{start}, and can only start after the execution of all its ancestors is done.

The new constraints, Constraints 6.2 and 6.3, express the fact that a task cannot start its execution before the work load of all its ancestors is done on the m CPUs and on the k GPUs. Between the two constraints, only the one with the greatest sum will have an impact on the linear program.

Notice that the two original constraints are included in these new sets of constraints because we have $\lambda = C_{end}$ and node $end \in V$.

We also delete the second type of constraint and change the objective value to be $\lambda = C_{end}$. These changes are relevant and do not impact the behavior of the linear program because the task T_{end} is a descendent of every other task and thus it is only required to minimize its completion time.

The variables of the linear programs are x_j and C_j, $\forall j = 1, \cdots, n$, and the variable C_{end} which is minimized in the objective function and corresponds to the makespan. For simplicity, we set $x_{start} = 1$, $C_{start} = 0$ and $x_{end} = 1$. Notice that the integer linear program only differs in the last constraint and is also refined as described above. The new linear program, denoted by $(LP2)$, is as follows:

minimize $\lambda = C_{end}$ subject to

$$
\begin{aligned}
& C_i + \overline{p_j} x_j + \underline{p_j}(1 - x_j) \le C_j && \forall j \in V,\ \forall i \in \Gamma^-(j) && (6.1)\\
& \sum_{i \in A(j)} \frac{\overline{p_i} x_i}{m} + \overline{p_j} x_j + \underline{p_j}(1 - x_j) \le C_j && \forall j \in V && (6.2)\\
& \sum_{i \in A(j)} \frac{\underline{p_i}(1 - x_i)}{k} + \overline{p_j} x_j + \underline{p_j}(1 - x_j) \le C_j && \forall j \in V && (6.3)\\
& x_j \in [0,1] && \forall j = 1, \cdots, n && (6.4)
\end{aligned}
$$

Once an optimal solution is found for $(LP2)$, we use the same rounding method to get integral values of the variables x_j as in HLP.

6.2 Modifications of the Scheduling Method

In this section, we propose a task prioritizing method to assign each task a rank, inspired by the ranking function of HEFT, and define a partial order of the tasks before using a variant of List Scheduling.

The motivation of assigning a priority to each task is to take into account the notion of precedence between the tasks, in order to prioritize the scheduling of critical tasks before the scheduling of independent tasks. A critical task is a task belonging to a critical path.

The rank of each task is computed after the rounding operation of the assignment variables x_j and corresponds to the length, in the sense of processing time, of the longest path between this task and the ending task. Thus, as in HEFT, each task will have a bigger rank than all its successors.

The computation of the rank is similar to the one used during the HEFT algorithm described in Chapter 4:

$$Rank(T_j) = \overline{p_j} x_j + \underline{p_j}(1 - x_j) + \max_{i \in \Gamma^+(j)} \{Rank(T_i)\}$$

After computing all ranks, the list of tasks T, containing every task but T_{start} and T_{end}, is split in two separate lists T^{CPU}, resp. T^{GPU}, containing the tasks to be executed on a CPU, resp GPU, with respect to the assignment variables x_j. The two lists are then separately sorted in non-increasing order of the rank of the tasks and ties are broken by giving priority to the task which have the longest path to the end node, in terms of number of nodes in the extended graph of precedences.

A variant of List Scheduling algorithm is then used to construct the schedule of the instance, with two systems of identical machines and two lists of tasks. The new defined algorithm, denoted by refined HLP, is as follows:

1. Solve $(LP2)$ to have the fractional assignment variables x_j

2. Round the variables x_j to integral values 0 or 1

3. Compute the rank of each task T_j
4. Split the list T in T^{CPU} and T^{GPU} *wrt.* the assignment x_j and sort the two lists by decreasing rank of the tasks
5. Build a feasible schedule by running two List Scheduling algorithms in parallel, one for the CPU side and one for the GPU side, taking into account the precedence relations of the tasks

6.3 Worst Case Example

The refined HLP algorithm keeps the same rounding method than HLP and the new scheduling method only change the order in which the tasks are scheduled by List Scheduling, not changing fundamentally how the method works. The two algorithms only differ in the set of constraints of the linear program used for obtaining the fractional assignment variables x_j.

The constraints of $(LP2)$ are more restrictive than the constraints of $(LP1)$ in a sense that, for any instance of the problem, the optimal solution of $(LP2)$, denoted by λ_2, is greater or equal to the optimal solution of $(LP1)$, denoted by λ_1. Thus, $\lambda_1 \leq \lambda_2$ and it is easy to show, following the argument of Kedad-Sidhoum *et al.*, that $\lambda_2 \leq C^*_{max}$, where C^*_{max} is the optimal makespan with integral assignment.

We can deduce that the proof of Kedad-Sidhoum *et al.* leading to the Theorem 6 is also valid for our algorithm, concluding that the approximation ratio of the refined HLP algorithm is at most 6.

Theorem 6.3.1 *The refined HLP algorithm has a tight approximation ratio of 6.*

In the following, we propose an instance of the problem for which the refined HLP algorithm creates a schedule with a makespan close to 6 times the optimal makespan, with any scheduling method.

6.3.1 Instance of the Problem

Consider a parallel system with an equal number m of CPUs and GPUs, with $m \geq 4$.

The list of tasks is composed of a single independent task T_A that can only be processed on a GPU and l groups of $2m+1$ tasks T_i, $\forall i = 1, \cdots, l$ and $1 < l \leq m$. Figure 6.1 shows the graph of precedence of the tasks. Notice that every task of the same type is independent with each other and that every task of type J_i have to be completed before the execution of any task of type J_{i+1}, $\forall i = 1, \cdots, l-1$.

The processing times of the tasks are as follows:

Type	Number of task	Processing time on CPU/GPU
A	1	∞ / $(l+2)m$
J_i, i odd	$2m+1$	$2m-1$ / 1
J_i, i even	$2m+1$	1 / $2m-1$

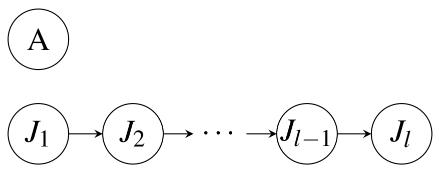

Figure 6.1: Precedence graph of the instance of tasks

6.3.2 Solving the Linear Program

Clearly the optimal makespan with this instance is $OPT = (l+2)m$ by placing task T_A on a GPU and every task T_{J_i} on the processor type which gives a processing time of 1.

Proposition 6.3.2 *A possible optimal solution of* $(LP2)$ *gives* $\lambda = C_{end} = (l+2)m$ *and the fractional assignment, with* $i = 1, \cdots, l$ *and* $\varepsilon > 0$ *small enough:*

$$\begin{cases} x_A = 0 & \\ x_{J_i} = \frac{1}{2} & \text{if } i \text{ is odd} \\ x_{J_i} = \frac{1}{2} - \varepsilon & \text{if } i \text{ is even} \end{cases}$$

The associated variables C_j are as follows:

$$\begin{cases} C_A = (l+2)m \\ C_{J_1} = m \\ C_{J_i} = (i+1)m + \left\lfloor \frac{i-2}{2} \right\rfloor - \frac{1}{2m} + 2\left\lfloor \frac{i}{2} \right\rfloor \varepsilon(m-1) \ \ , \forall i = 2, \cdots, l \end{cases}$$

Proof:
It is easy to see that no solution with $\lambda < (l+2)m$ is possible because of the first constraint (6.1) and the task T_A:

$$C_{start} + \underline{p_A} \leq C_A$$

$$(l+2)m \leq C_A$$

Since $C_{end} \geq C_A$ we have $\lambda = C_{end} \geq (l+2)m$.

It is then required to check that every constraint of $(LP2)$ is satisfied by the variables x_j, C_j and $\lambda = (l+2)m$, to conclude that the solution is an optimal solution of $(LP2)$.
Details on the computations can be found in Appendix A.

Hence, Proposition 6.3.2 is verified. □

6.3.3 Rounding and Scheduling Phases

After solving the linear program, the variables C_j are no longer relevant and only the assignment variables x_j are used to construct a feasible schedule with m CPUs and k GPUs.
During the rounding phase, each variable x_j is either rounded to the integral value 1 or 0 to

determine whether the task T_j will be processed on a CPU or a GPU. The assignment variables corresponding to our instance after the rounding phase are as follows:

$$\begin{cases} x_A = 0 & \\ x_{J_i} = 1 & \text{if i is odd} \\ x_{J_i} = 0 & \text{if i is even} \end{cases}$$

After the rounding phase, there is only one possible makespan with this assignment of tasks, independant of the scheduling method used to construct the schedule.

Figure 6.2 shows the resulting schedule with the given instance and assignment, with an odd value of l in this example. Individual tasks are not explicitly placed in the schedule since tasks of a same type are independent to each other. Instead, we group every task of the same type in a block labelled by the type of task scheduled in that block.

The obtained makespan is $C_{max} = 3l(2m-1)$, compared to the optimal value $(l+2)m$ we have the approximation ratio:

$$\rho = \frac{l(6m-3)}{(l+2)m}$$

The ratio tends to 6 as l grows large and if we set $l = m$ we obtain the simplified approximation ratio:

$$\begin{aligned} \rho &= \frac{m(6m-3)}{m(m+2)} \\ &= 6 - \frac{15}{m+2} \end{aligned}$$

Thus, with m and l large enough we built a schedule with a makespan 6 times greater than the optimal makespan possible with the given instance.

We conclude that the approximation ratio of the refined HLP algorithm is at least 6 and Theorem 6.3.1 holds.

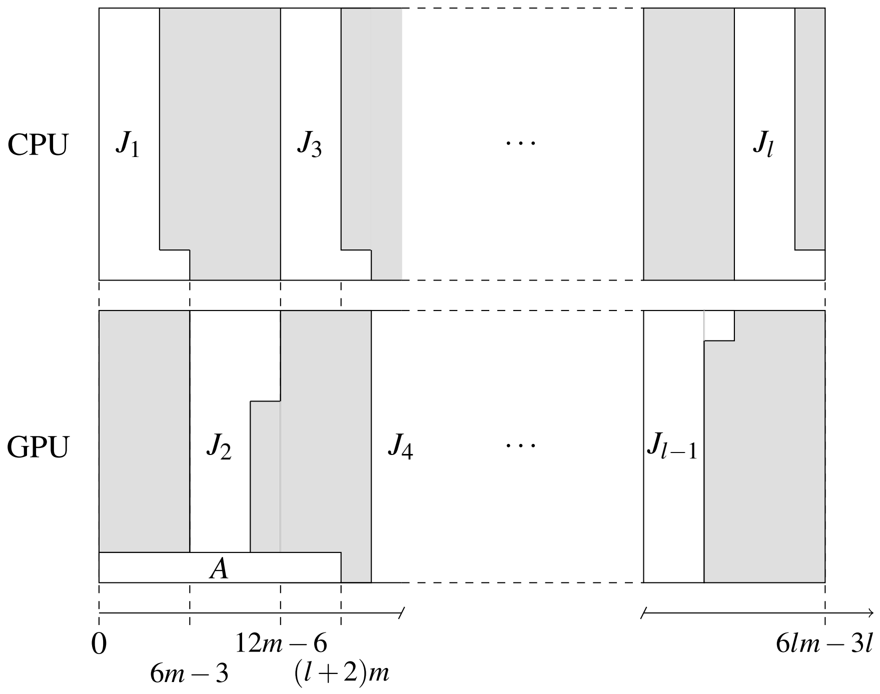

Figure 6.2: Resulting schedule of the algorithm for the instance with an odd value of l. Notice that the gray areas represent idle times.

— 7 —
Generalization on Q Resources Types

In this chapter, we generalize the algorithm presented in Chapter 6 to address the problem of scheduling tasks linked by precedence constraints on a parallel system composed of multiple sets of resources, with each set containing identical processors. We also propose an analysis of the algorithm to give an upper bound of the approximation ratio depending on the number of resource types.

The studied problem only differs from the previous chapters in the parallel system considered. Instead of having only two types of processors, called CPUs and GPUs, we study here a system having Q different types of resources with each type having M_q identical processors. The total number of processors of the system is $m = \sum_{q=1}^{Q} M_q$. Recall that we denote by $p_{j,q}$ the processing time of task T_j if it is executed on a processor of type q.

The algorithm that we analyze is the same as the refined HLP, where each step is adapted to take into account the Q types of resources instead of only a CPU side and a GPU side. In the following sections, we present the modified linear programs and the rounding and scheduling methods used by the algorithm to solve the scheduling problem. We then analyze the structure of the schedule produced by the algorithm and prove that its approximation ratio is at most $Q(Q+1)$.

7.1 Linear Program

In this section, we adapt the integer and relaxed linear programs defined in Section 6.1 for the assignment problem with Q types of resources.

Regarding the set of tasks to be scheduled, the variables keep the same notations except from the assignment variables which are now defined as follows:

$$x_{j,q} = \begin{cases} 1 & \text{if } T_j \text{ is processed on a processor of type } q \\ 0 & \text{otherwise} \end{cases} \qquad \forall j \in V,\ \forall q = 1, \cdots, Q$$

Then, the variables of the linear programs are $x_{j,q}$ and C_j, $\forall j = 1, \cdots, n$ and we aim to minimize C_{end}. The new integer linear program is as follows:

minimize $\lambda = C_{end}$ subject to

$$C_i + \sum_{q=1}^{Q} p_{j,q} x_{j,q} \leq C_j \qquad \forall j \in V,\ \forall i \in \Gamma^-(j) \tag{7.1}$$

$$\sum_{i \in A(j)} \frac{p_{j,q} x_{j,q}}{M_q} + \sum_{q=1}^{Q} p_{j,q} x_{j,q} \leq C_j \qquad \forall j \in V,\ \forall q = 1, \cdots, Q \tag{7.2}$$

$$\sum_{q=1}^{Q} x_{j,q} = 1 \qquad \forall j \in V \tag{7.3}$$

$$x_{j,q} \in \{0,1\} \qquad \forall j \in V,\ \forall q = 1, \cdots, Q \tag{7.4}$$

As for the refined HLP algorithm, Constraints 7.1 and 7.2 represent the precedence constraints between the tasks and the constraints of partial load on each type of resource for each task. Constraint 7.3 ensure that a task is entirely executed and Constraints 7.4 are the integrity constraints, making sure that a task is executed on only one type of resources.

As for HLP and refined HLP, the integrity constraint is relaxed in order to have a linear program that can be solved optimally in polynomial time. The relaxed linear program, denoted by (LP_q), is as follows:

minimize $\lambda = C_{end}$ subject to

$$C_i + \sum_{q=1}^{Q} p_{j,q} x_{j,q} \leq C_j \qquad \forall j \in V,\ \forall i \in \Gamma^-(j) \tag{7.5}$$

$$\sum_{i \in A(j)} \frac{p_{j,q} x_{j,q}}{M_q} + \sum_{q=1}^{Q} p_{j,q} x_{j,q} \leq C_j \qquad \forall j \in V,\ \forall q = 1, \cdots, Q \tag{7.6}$$

$$\sum_{q=1}^{Q} x_{j,q} = 1 \qquad \forall j \in V \tag{7.7}$$

$$x_{j,q} \in [0,1] \qquad \forall j \in V,\ \forall q = 1, \cdots, Q \tag{7.8}$$

7.2 Rounding and Scheduling Methods

We describe in this section the rounding policy, the ranking function and the variant of List Scheduling used by the algorithm.

We denote by $x^R_{j,q}$ the assignment variables given by the optimal solution of (LP_q). These variables have possible fractional values between 0 and 1, allowing a task to be fractionally executed on several types of processors. Since our model does not allow preemption of tasks, it is required to round the variables $x^R_{j,q}$ to either 0 or 1 to have each task executed by exactly one type of processor.

The rounding method proceeds in two steps. We first determine the type of resource on which each task T_j is mostly assigned to and then we set the corresponding assignment variable to 1 and every other variables to 0. The computation is as follows:

1. $r_j = \arg\max_{q=1,\cdots,Q}\{x^R_{j,q}\}\ \ \forall j \in V$

2. $x_{j,q} = \begin{cases} 1 & \text{if } q = r_j \\ 0 & \text{otherwise} \end{cases}\ \ \forall j \in V,\ \forall q = 1,\cdots,Q$

In case of ties during the computation of r_j, we give priority to the resource type with the smallest processing time $p_{j,q}$.

Once the assignment variables have been rounded, the rank of each task is computed as follows:

$$Rank(T_j) = \sum_{q=1}^{Q} p_{j,q}x_{j,q} + \max_{i\in\Gamma^+(j)}\{Rank(T_i)\}$$

Before scheduling the tasks, the list of tasks T is split into Q sublists corresponding to the tasks to be scheduled on processors of the same resource type. The schedule is then built by running Q parallel List Scheduling algorithms as described in Section 6.2.

The algorithm is then as follows:

1. Solve (LP_q) to have the fractional assignment variables $x^R_{j,q}$
2. Round the variables $x^R_{j,q}$ to integral values 0 or 1
3. Compute the rank of each task T_j
4. Split the list T in Q lists T_q, $\forall q = 1,\cdots,Q$, *wrt.* the assignment $x_{j,q}$ and sort the lists by decreasing rank of the tasks
5. Build a feasible schedule by running Q List Scheduling algorithms in parallel, one for each sublist T_q, taking into account the precedence relations of the tasks

7.3 Analysis

In this section, we analyze the structure of the schedule produced by the algorithm and show that the approximation ratio of this algorithm is at most $Q(Q+1)$. The analysis of the algorithm and the structure of the schedule is similar to the one of Kedad-Sidhoum *et al.* [10].

Theorem 7.3.1 *The approximation ratio of the algorithm is at most* $Q(Q+1)$.

Proof:
We denote by W_q, $\forall q = 1,\cdots,Q$, the total work load on all processors of type q in the schedule. We also denote by C^R_{max}, W^R_q and L^R the objective value, the total work load on all processors of type q and the length of the longest path in the fractional optimal solution of (LP_q), respectively. Finally, we define by C^*_{max} the optimal makespan of the scheduling problem with integral assignment of tasks to processors.

We have the following inequalities:

$$L^R \leq C^R_{max} \leq C^*_{max} \tag{7.9}$$

$$\frac{W^R_q}{M_q} \leq C^R_{max} \leq C^*_{max}\quad \forall q = 1,\cdots,Q \tag{7.10}$$

To analyze the structure of the schedule, we partition into two disjoint subsets of intervals $\mathscr{T}_{CP}$ and $\mathscr{T}_W$ the time interval of the schedule $\mathscr{T} = [0, C_{max}]$ as follows:

$$\begin{cases} \mathscr{T}_{CP} & = \{t \in \mathscr{T} \mid \text{at least one processor of each type is idle at time } t\} \\ \mathscr{T}_W & = \{\mathscr{T} \backslash \mathscr{T}_{CP}\} \end{cases}$$

We then can divide the set $\mathscr{T}_W$ into Q possibly non-disjoint subsets defined as:

$$\mathscr{T}_q = \{t \in \mathscr{T}_W \mid \text{all processors of type q are busy at time } t\}\ , \forall q = 1, \cdots, Q$$

Notice that there are $Q+1$ subsets $\mathscr{T}_i$, with $i = CP, 1, \cdots, Q$. We define the length $|\mathscr{T}_i|$ to be the number of unitary time slots in $\mathscr{T}_i$.

Clearly we have the following inequalities:

$$C_{max} = |\mathscr{T}| \leq |\mathscr{T}_{CP}| + \sum_{q=1}^{Q} |\mathscr{T}_q|$$

In the following, we will bound above by QC^*_{max} each subset $\mathscr{T}_i$.

Due to the rounding policy, we know that if $x_{j,q} = 1$ then $x^R_{j,q} \geq \frac{1}{Q}$. We then have:

$$x_{j,q} \leq Qx^R_{j,q} \quad \forall j \in V,\ \forall q = 1, \cdots, Q \tag{7.11}$$

Consider the subset $\mathscr{T}_{CP}$. There is a directed path $\mathscr{P}$ of tasks being executed during any time slot in $\mathscr{T}_{CP}$. The construction of $\mathscr{P}$ is the same as described by Kedad-Sidhoum *et al.* in their analysis. Since the directed path $\mathscr{P}$ covers every time slot in $\mathscr{T}_{CP}$, the length of $\mathscr{T}_{CP}$ is smaller than the length of $\mathscr{P}$ and the length of $\mathscr{P}$ in the fractional optimal solution of (LP_q), noted $\mathscr{P}^R$, is smaller than L^R.

Thus, using the inequalities (7.9) and (7.11), we have the following bounds:

$$|\mathscr{T}_{CP}| \leq |\mathscr{P}| \leq \sum_{j \in \mathscr{P}} \sum_{q=1}^{Q} p_{j,q} x_{j,q} \leq Q \sum_{j \in \mathscr{P}} \sum_{q=1}^{Q} p_{j,q} x^R_{j,q} \leq QL^R \leq QC^*_{max}$$

Consider now the subset $\mathscr{T}_q$, with $q = 1, \cdots, Q$. At each time slot of $\mathscr{T}_q$ all processors of type q are busy. Using the inequalities (7.10) and (7.11), we have the following bounds:

$$|\mathscr{T}_q| \leq \frac{W_q}{M_q} \leq \frac{1}{M_q} \sum_{x_{j,q}=1} p_{j,q} \leq \frac{Q}{M_q} \sum_{j \in V} p_{j,q} x^R_{j,q} \leq \frac{QW^R_q}{M_q} \leq QC^*_{max}$$

Thus, by combining the calculated bounds we have the inequality:

$$C_{max} = |\mathscr{T}| \leq |\mathscr{T}_{CP}| + \sum_{q=1}^{Q} |\mathscr{T}_q| \leq Q(Q+1)C^*_{max}$$

Hence, Theorem 7.3.1 follows. □

Using the worst case example proposed in Section 6.3 with $Q = 2$, the following corollary follows.

Corollary 7.3.2 *The algorithm has a tight approximation ratio of $Q(Q+1)$.*

— 8 — Experiments

In this chapter, we use 6 parallel applications of linear albegra to compare the schedules produced by HEFT, HLP and the refined HLP algorithms presented in the previous chapters. We describe the instance inputs of the algorithms, the environment and analyze the results of the experiments.

8.1 Input Data

Each of the three algorithms take as input an application, composed of a DAG of precedence between the tasks and the processing times on CPU and on GPU for each task, as well as the number of CPUs and the number of GPUs that compose the parallel platform on which the application is to be scheduled.

The applications used for our experiments are 6 applications from Chameleon[1], a dense linear algebra software which is part of the MORSE[2] project. The applications were run on a parallel platform and the traces of execution were analyzed to extract the list of generated tasks with their processing times on CPU and on GPU and their predecessors.

The 6 applications, named *sgetrf_nopiv*, *sgetrs_nopiv*, *sposv*, *spotrs*, *spotri* and *spotrs*, are composed of multiple sequential basic tasks of linear algebra such as *SYRK* (symetric rank update), *GEMM* (general matrix-matrix multiply), *TRSM* (triangular matrix equation solver) and *DPOTRF* (computes the Cholesky factorization). Different tiling of the matrices have been used, varying the number of sub-matrices denoted by *nb_blocs* and the size of the sub-matrices denoted by *bloc_size*. The different values of *nb_blocs* were 10, 20, 50 and 100 and the different values of *bloc_size* were 64, 128, 320, 512, 768 and 960, for a total of 24 configurations for each application.

Table 8.1 shows the total number of tasks for each application and each value of *nb_blocs*. Notice that the *bloc_size* does not impact the number of tasks.

From the traces, a file have been created for each combination of application, regrouping on each line the index of a task, its processing time on CPU, its processing time on GPU and the list of indexes of its predecessors. Each combination of application consists of the pair (*nb_blocs*,*bloc_size*), for a total of 24 files per application.

[1]https://project.inria.fr/chameleon/

[2]Marices Over Runime Systems at Exascale

Nb_blocs	sgetrf_nopiv	sgetrs_nopiv	sposv	spotrf	spotri	spotrs
10	385	110	330	220	660	110
20	2870	420	1960	1540	4620	420
50	42925	2550	24650	22100	66300	2550
100	338500	10100	181800	171700	515100	10100

Table 8.1: Total number of task of the application in function of the number of blocs

For the input number of CPUs and GPUs, due to the gap on the number of tasks between 20 and 50 blocs, we determined two different sets of pairs (number_CPU, number_GPU). For the *nb_blocs* values 10 and 20, we used 64, 128, 256 and 512 CPUs with 8, 16 and 32 GPUs for a total of 12 machine configurations. For the *nb_blocs* values 50 and 100, we used 2048, 8192, 16384 and 32768 CPUs with 256, 512, 1024 and 2048 GPUs for a total of 16 machine configurations.

8.2 Experiment Environment and Algorithms

The three algorithms are implemented in Python, version 2.7.6, and the linear program solver used is the *glpsol* command-line solver, version 4.52, of the GLPK package (GNU Linear Programming Kit).

All the experiments were performed using the Froggy platform of the CIMENT infrastructure (https://ciment.ujf-grenoble.fr), which is supported by the Rhône-Alpes region (GRANT CPER07_13 CIRA) and the Equip@Meso project (reference ANR-10-EQPX-29-01) of the programme Investissements d'Avenir supervised by the Agence Nationale pour la Recherche.

Except from HEFT, HLP and refined HLP, a variant of the HLP algorithm have been implemented, denoted by HLP_ranked, using the linear program of HLP and the ranking and scheduling methods of the refined HLP algorithm.

For the *nb_blocs* values 10 and 20, each of the 6 applications with each of the 6 *bloc_size* values and the 12 machine configurations were used as inputs to test the algorithm. The resulting makespan of HEFT, HLP, HLP_ranked and refined HLP were stored, as well as the objective values of the linear programs $(LP1)$ and $(LP2)$.

Each combination of application and machine configuration have been run only once since the algorithms, as well as *glpsol*, are deterministic.

Due to the growing size and complexity of the linear programs used by these algorithms, the experiments with $nb\_blocs = 100$ for all algorithms, as well as $nb\_blocs = 50$ for the refined HLP algorithm, are still running and will be analyzed in a future work.

8.3 Results

In this section, we analyse the results of the experiments. More precisely, we compare the behavior of the HLP algorithm to the theoretical approximation ratio of 6 and compare the performances of HEFT, HLP, HLP_ranked and refined HLP on the described instances.

8.3.1 Analysis of HLP

We study here the ratio between the mean of the makespan of HLP (C_{max}) and the mean of the lower bound provided by the fractional optimal solution of $(LP1)$ (λ) per application, in function of the number of blocs tiling the matrix. The means are computed over the 72 instances of each application for $nb\_blocs = 10$ or 20 and over the 96 instances for $nb\_blocs = 50$.

Figure 8.1 shows the ratio $\frac{\text{mean}(C_{max})}{\text{mean}(\lambda)}$ of each application in function of nb_blocs. We can see that, on average, the ratio does not exceed 1.4.

Looking more closely at the computed values we notice that, over all instances and all applications, the average of the ratios is near 1.1, meaning that the makespan of HLP is on average 10% away from the optimal theoretical makespan for these instances. Moreover, the maximal ratio obtained from these instances is close to 1.62 and is achieved with the application sgetrf_nopiv with $nb\_blocs = 20$, a $bloc\_size$ of 512, 64 CPUs and 8 GPUs.

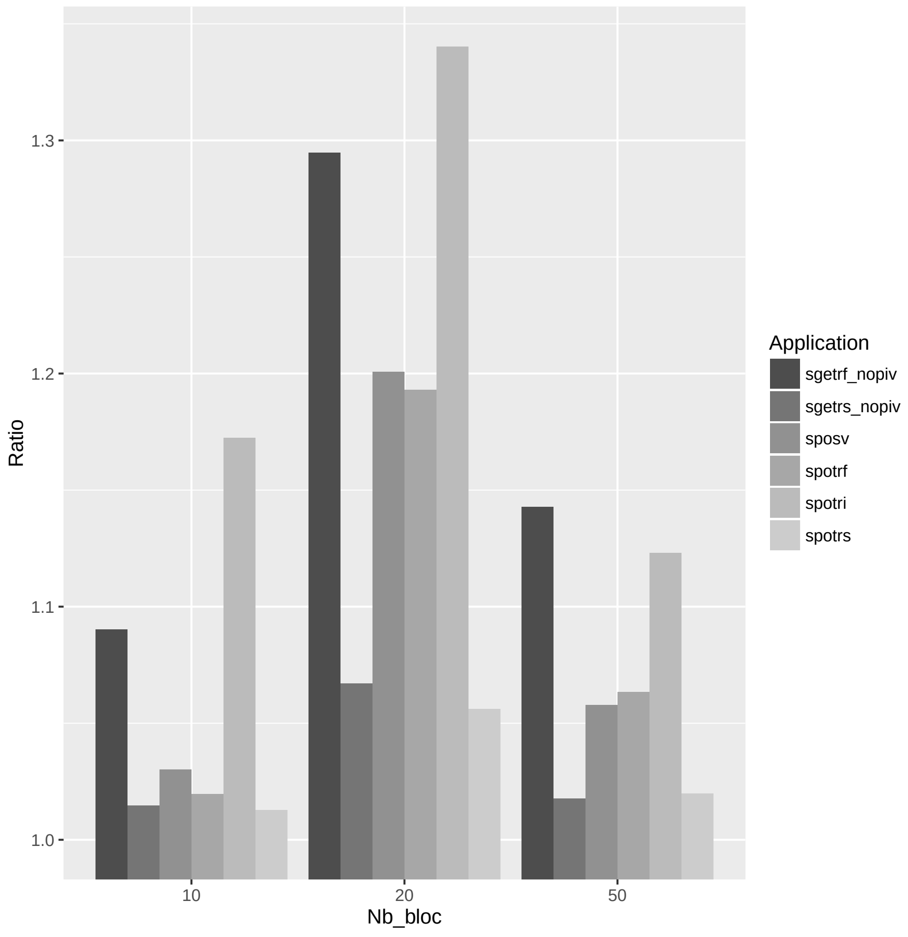

Figure 8.1: Ratio $\frac{\text{mean}(C_{max})}{\text{mean}(\lambda)}$ in function of nb_blocs for the HLP algorithm

An interesting observation, regarding the values of x_j for the optimal solutions of $(LP1)$, is that the proportion of fractional values over the total number of tasks is on average smaller than 4% for the 576 instances. Moreover, only 7 instances of the spotri application have more than 10% of fractional variables, while the maximum is 16% for the application spotri with *nb_blocs* $= 50$, a *bloc_size* of 960, 16384 CPUs and 256 GPUs.

8.3.2 Comparison of the LP-based Algorithms

We study now the performances of the algorithms HLP, HLP_ranked and refined HLP by comparing the average of their makespans over all the instances for the 6 applications. The means are computed over the 144 instances corresponding to *nb_blocs* $= 10$ and 20 for each application.

Figure 8.2 shows the mean makespans of each application for each of the 3 algorithms under study. We can see that the HLP algorithm performs, on average, worse than the two others and that the algorithm HLP_ranked and refined HLP are quite similiar in terms of mean makespan.

For 864 different instances (6 applications and 144 configurations per application), the HLP algorithm strictly outperforms HLP_ranked in only 2 cases and has the same makespan in 325 cases. HLP strictly outperforms refined HLP in 273 cases and has the same makespan in 145 cases. Finally, the HLP_ranked algorithm outperforms refined HLP in 422 cases and has the same makespan in 193 cases over the 864 instances.

Notice that the overall difference of performances between the 3 algorithm is on average smaller than 5%.

8.3.3 Comparison with HEFT

In the previous section we observed that HLP was outperformed by both HLP_ranked and refined HLP in terms of average makespan over all the instances. Since the running time of refined HLP, as well as the amount of memory it requires, is much greater than for HLP_ranked, we decided to compare the performances of HEFT to HLP_ranked.

To do so, we compared the average makespans over the 240 instances for each of the 6 applications. The means are computed over the 144 instances corresponding to *nb_blocs* $= 10$ and 20 and the 96 instances corresponding to *nb_blocs* $= 50$.

Figure 8.3 shows the mean makespans of each application for the algorithm HEFT and HLP_ranked. We can see that HEFT outperforms the other algorithm on the average of the makespans. Looking more in details at the ratio between the makespan of HLP_ranked and the makespan of HEFT we notice that, on average, the makespans of HLP_ranked are less than 4% greater than the makespans of HEFT, considering all applications with all instances.

The instance with the highest ratio between the two makespans is achieved with the application sgetrf_nopiv, with *nb_blocs* $= 20$, a *bloc_size* of 960, 64 CPUs and 8 GPUs, with HLP_refined having a makespan near to 28% worse than HEFT. The instance with the smallest ratio is for the application spotri, with *nb_blocs* $= 20$, a *bloc_size* of 512, 256 CPUs and 8 GPUs, with HLP_refined having a makespan near to 30% better than HEFT.

To conclude, the 3 LP-based algorithms have relative good performance compared to their approximation ratio for these 6 applications, with the different instances defined. Considering the average of the makespans for the applications, HEFT is the best algorithm in practice but, contrary to the 3 other algorithms, there is no constant performance guarantee on its approximation ratio.

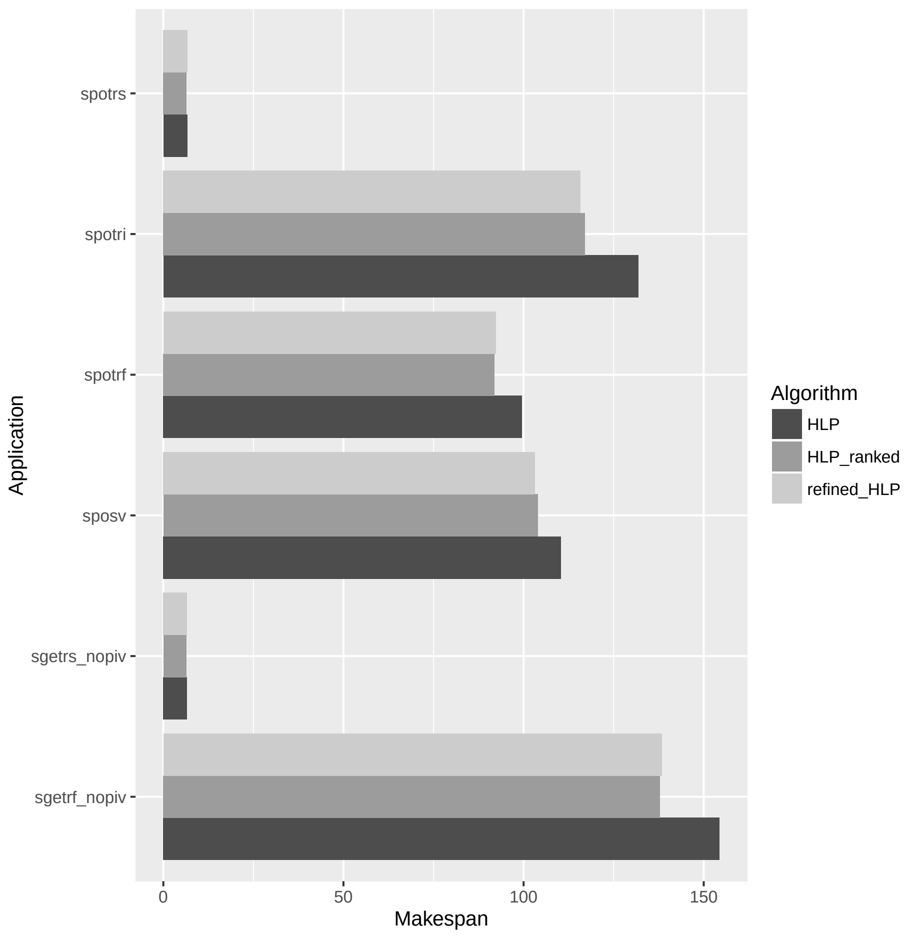

Figure 8.2: Mean makespan of HLP, HLP_ranked and refined HLP for the 6 applications

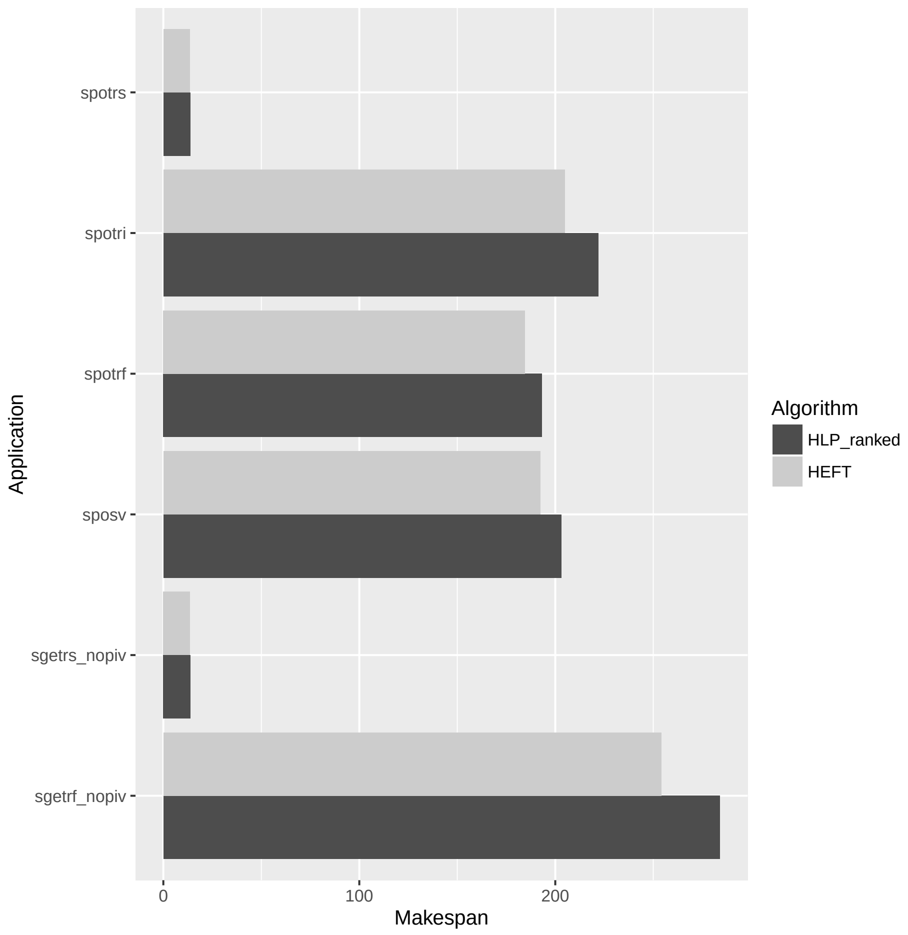

Figure 8.3: Mean makespan of HLP_ranked and HEFT for the 6 applications

— 9 — Conclusion

In this work, we studied the problem of scheduling sequential tasks linked by precedence constraints on heterogeneous multi-core architectures composed of identical CPUs and GPUs in order to minimize the makespan. We focused on two existing generic algorithms, namely HEFT and HLP, for addressing this problem and proposed some refinements of the second one in order to improve the quality of produced schedules.

We improved the known lower bound for HEFT by giving a better $\Omega(m)$ counter-example and we gave a worst-case instance for HLP showing that the approximation ratio of the algorithm was tight at 6.

Then, we proposed new constraints for the linear programs used in the assignment step of HLP, as well as a new scheduling method based on the ranking function of HEFT, in order to give a better lower bound on the optimal makespan and improve the produced schedule. However, the approximation ratio of the new algorithm remains at 6. We also provide a proof of its tightness. This means that the algorithm can only be improved by modifying the assignment method since the provided worst-case example can be adapted for several scheduling policies.

We also presented a new testbed constructed using a representative benchmark of 6 applications of linear algebra routines, issued from the library Chameleon, in order to evaluate the performances of the algorithms in a realistic context. For the experiments, we defined a variant of HLP using the scheduling method based on the ranking of the tasks in addition to the three other algorithms under study. Though the computed approximation ratio of HLP did not exceed 1.62, it appeared that HLP was outperformed by the variant and by the refined HLP algorithm. Both proposed algorithms lead to an improvement of the makespans close to 5% on average. Moreover, the comparison of HEFT and the variant of HLP showed that HEFT was performing better than the variant with a small improvement in average (close to 4%). However, HEFT does not provide any worst-case performance guarantee on the produced schedules.

Finally, we proposed a tight $Q(Q+1)$-approximation algorithm for the problem of scheduling dependent tasks on Q different types of resources. This is an important step towards the increasing degree of heterogeneity of the future parallel platforms since it is the first approximation algorithm taking into account precedence constraints for this kind of architectures with more than two types of resources.

Based on the results of this work, we can conclude that the assignment step is a crucial issue for the performances of scheduling algorithms for hybrid and more general heterogeneous platforms. Thus, it is important to further improve this assignment step both in theory and practice. However, as we observed during the experiment process, the resolution of the linear programs for deciding the assignment of the tasks is not appropriate for practical applications since it significantly increases the execution time, as well as the memory used by the linear program-based algorithms. Thus, a promising idea could be the replacement of the linear programs by a set of rules, like the one used to solve the on-line version of the problem [5], in order to simplify and reduce the execution time of the assignment step as well as improving the approximation ratio.

Another interesting mid-term direction is to deal with the increasing heterogeneity of the platforms, such as the utilization of specific nodes for data analytics or for I/O operations, on the road to the exascale. Here, it is critical to develop efficient greedy algorithms that are adapted to the increasing number of nodes and the complexity of such platforms, as well as to the ever-growing size of the applications and the data transfers.

Bibliography

[1] H. Arabnejad and J. G. Barbosa. List scheduling algorithm for heterogeneous systems by an optimistic cost table. *IEEE Transactions on Parallel and Distributed Systems*, 25(3):682–694, March 2014.

[2] Yossi Azar and Leah Epstein. On-line scheduling with precedence constraints. *Discrete Applied Mathematics*, 119(1-2):169 – 180, 2002.

[3] R. Bleuse, S. Kedad-Sidhoum, F. Monna, G. Mounié, and D. Trystram. Scheduling independent tasks on multi-cores with GPU accelerators. *Concurrency and Computation: Practice and Experience*, 27(6):1625–1638, 2015.

[4] Bo Chen and Arjen P.A. Vestjens. Scheduling on identical machines: How good is lpt in an on-line setting? *Operations Research Letters*, 21(4):165 – 169, 1997.

[5] Lin Chen, Deshi Ye, and Guochuan Zhang. Online scheduling of mixed cpu-gpu jobs. *International Journal of Foundations of Computer Science*, 25(06):745–761, 2014.

[6] Jan Clemens Gehrke, Klaus Jansen, Stefan E. J. Kraft, and Jakob Schikowski. A ptas for scheduling unrelated machines of few different types. In *SOFSEM 2016: Theory and Practice of Computer Science: 42nd International Conference on Current Trends in Theory and Practice of Computer Science*, pages 290–301, Berlin, Heidelberg, 2016.

[7] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45(9):1563–1581, 1966.

[8] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal On Applied Mathematics*, 17(2):416–429, 1969.

[9] Dorit S. Hochbaum and David B. Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *J. ACM*, 34(1):144–162, January 1987.

[10] S. Kedad-Sidhoum, F. Monna, and D. Trystram. Scheduling Tasks with Precedence Constraints on Hybrid Multi-core Machines. In *IPDPSW 2015 - IEEE International Parallel and Distributed Processing Symposium Workshop*, pages 27–33, Hyderabad, India, May 2015.

[11] Safia Kedad-Sidhoum, Fernando Machado Mendonca, Florence Monna, Gregory Mounie, and Denis Trystram. Fast biological sequence comparison on hybrid platforms. In *43rd International Conference on Parallel Processing, ICPP 2014, Minneapolis, MN, USA, September 9-12, 2014*, pages 501–509, 2014.

[12] Minhaj Ahmad Khan. Scheduling for heterogeneous systems using constrained critical paths. *Parallel Comput.*, 38(4-5):175–193, April 2012.

[13] J. K. Lenstra, D. B. Shmoys, and É. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Math. Program.*, 46(3):259–271, February 1990.

[14] Jane W. Liu and C. L. Liu. Performance analysis of multiprocessor systems containing functionally dedicated processors. *Acta Inf.*, 10(1):95–104, March 1978.

[15] R. Sakellariou and H. Zhao. A hybrid heuristic for dag scheduling on heterogeneous systems. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, April 2004.

[16] Evgeny V. Shchepin and Nodari Vakhania. An optimal rounding gives a better approximation for scheduling unrelated machines. *Operations Research Letters*, 33(2):127–133, 2005.

[17] D. B. Shmoys, J. Wein, and D. P. Williamson. Scheduling parallel machines on-line. In *Foundations of Computer Science, 1991. Proceedings., 32nd Annual Symposium on*, pages 131–140, Oct 1991.

[18] Ola Svensson. Hardness of precedence constrained scheduling on identical machines. *SIAM J. Comput.*, 40(5):1258–1274, September 2011.

[19] H. Topcuoglu, S. Hariri, and Min-You Wu. Task scheduling algorithms for heterogeneous processors. In *Heterogeneous Computing Workshop, 1999. (HCW '99) Proceedings. Eighth*, pages 3–14, 1999.

— A —
Computations from Chapter 6

In the following, we will check that every constraint of $(LP2)$, presented in Section 6.1, is satisfied with the variables x_j and C_j by studying the types of tasks, since every task of the same type has the same constraints and values of variables.

The validity of the constraints for T_{start} is straightforward since its processing time is always 0 and $C_{start} = 0$. The last constraint is easily checked for every task so the three first constraints remains to be checked.

For the task T_A and tasks T_{J_1}:
Since tasks of type A or J_1 has no predecessors, except from T_{start} with processing time 0, the three first constraints are equivalent and then:

$$\begin{aligned} C_{start} + \underline{p_A} &\leq C_A \\ 0 + (l+2)m &\leq (l+2)m \end{aligned}$$

$$\begin{aligned} C_{start} + \overline{p_{J_1}} x_{J_1} + \underline{p_{J_1}}(1 - x_{J_1}) &\leq C_{J_1} \\ 0 + (2m-1)\frac{1}{2} + \frac{1}{2} &\leq m \\ m &\leq m \end{aligned}$$

The two inequalities are true. Thus, the constraints are satisfied for the task types A and J_1.

For the tasks T_{J_i}, $i \geq 2$:
Due to the precedence relations, the constraints associated with the tasks T_{J_i} uses the variable C_{i-1}, so they rely on the constraints associated with $T_{J_{i-1}}$, which rely on the constraints associated with $T_{J_{i-2}}$ and so on. Thus, in order to check the constraints associated with the tasks T_{J_i} it is required to check every constraint with lower values of i.

We will show by an induction proof on i that the three constraints associated with the tasks

T_{J_i} are satisfied. To do so, we define the two sums:

$$S_i^{CPU} = \sum_{a \in A(i)} \frac{\overline{p_a} x_a}{m} \qquad \forall i = 1, \cdots, l$$
$$S_i^{GPU} = \sum_{a \in A(i)} \frac{\underline{p_a}(1 - x_a)}{m} \qquad \forall i = 1, \cdots, l$$

The set of ancestors $A(i)$ of a task T_{J_i} contains all tasks of type J_1 to J_{i-1}.
If i is odd, there are $\frac{i-1}{2}$ types J_a, a odd, and $\frac{i-1}{2}$ types J_a, a even.
If i is even, there are $\frac{i}{2}$ types J_a, a odd, and $\frac{i-2}{2}$ types J_a, a even.

The first sum can be simplified to:

$$\begin{aligned}
S_i^{CPU} &= \frac{i-1}{2}\frac{(2m+1)(2m-1)\frac{1}{2}}{m} + \frac{i-1}{2}\frac{(2m+1)(\frac{1}{2}-\varepsilon)}{m} && \text{if i is odd} \\
&= \frac{i-1}{2}(\frac{4m^2-1}{2m} + \frac{m+\frac{1}{2}-\varepsilon(2m+1)}{m}) \\
&= \frac{i-1}{2}(2m+1-\varepsilon(2+\frac{1}{m})) \\
&= (i-1)m + \frac{i-1}{2} - \varepsilon(i-1+\frac{i-1}{2m}) \\
S_i^{CPU} &= \frac{i}{2}\frac{(2m+1)(2m-1)\frac{1}{2}}{m} + \frac{i-2}{2}\frac{(2m+1)(\frac{1}{2}-\varepsilon)}{m} && \text{if i is even} \\
&= \frac{i}{2}(2m - \frac{1}{2m}) + \frac{i-2}{2}(1+\frac{1}{2m} - \varepsilon(2+\frac{1}{m})) \\
&= im - \frac{1}{2m} + \frac{i-2}{2} - \varepsilon(i-2+\frac{i-2}{2m})
\end{aligned}$$

And with the same computation applied for the second sum:

$$\begin{aligned}
S_i^{GPU} &= (i-1)m + \frac{i-1}{2} + \varepsilon((2i-2)m - \frac{i-1}{2m}) && \text{if i is odd} \\
S_i^{GPU} &= (i-2)m + \frac{i}{2} + \frac{1}{2m} + \varepsilon((2i-4)m - \frac{i-2}{2m}) && \text{if i is even}
\end{aligned}$$

Notice that Constraints 6.2 and 6.3 can be rewriten, when task T_{J_i} is considered, as:

$$S_j^{CPU} + \overline{p_{J_i}}(x_{J_i}) + \underline{p_{J_i}}(1 - x_{J_i}) \leq C_{J_i}$$
$$S_j^{GPU} + \overline{p_{J_i}}(x_{J_i}) + \underline{p_{J_i}}(1 - x_{J_i}) \leq C_{J_i}$$

For the base case $i = 2$:

$$\begin{aligned}
C_{J_1} + \overline{p_{J_2}}x_{J_2} + \underline{p_{J_2}}(1 - x_{J_2}) &\leq C_{J_2} \\
m + \frac{1}{2} - \varepsilon + (2m-1)(\frac{1}{2} + \varepsilon) &\leq 3m - \frac{1}{2m} + \varepsilon(2m-2) \\
2m + \varepsilon(2m-2) &\leq 3m - \frac{1}{2m} + \varepsilon(2m-2) \\
0 &\leq m - \frac{1}{2m}
\end{aligned}$$

$$\begin{aligned}
S_2^{CPU} + \overline{p_{J_2}}x_{J_2} + \underline{p_{J_2}}(1 - x_{J_2}) &\leq C_{J_2} \\
2m - \frac{1}{2m} + \frac{1}{2} - \varepsilon + (2m-1)(\frac{1}{2} + \varepsilon) &\leq 3m - \frac{1}{2m} + \varepsilon(2m-2) \\
3m - \frac{1}{2m} + \varepsilon(2m-2) &\leq 3m - \frac{1}{2m} + \varepsilon(2m-2) \\
0 &\leq 0
\end{aligned}$$

$$\begin{aligned}
S_2^{GPU} + \overline{p_{J_2}}x_{J_2} + \underline{p_{J_2}}(1 - x_{J_2}) &\leq C_{J_2} \\
1 + \frac{1}{2m} + \frac{1}{2} - \varepsilon + (2m-1)(\frac{1}{2} + \varepsilon) &\leq 3m - \frac{1}{2m} + \varepsilon(2m-2) \\
m + 1 + \frac{1}{2m} + \varepsilon(2m-2) &\leq 3m - \frac{1}{2m} + \varepsilon(2m-2) \\
0 &\leq 2m - 1 - \frac{1}{m}
\end{aligned}$$

The three inequalities are true. Thus, the constraints are satisfied for the type J_2.

For the general case i, assume the constraints are satisfied from 2 to $i-1$. We need to check that the constraints are also satisfied for i.
If i is odd:

$$\begin{aligned}
C_{J_{i-1}} + \overline{p_{J_i}}x_{J_i} + \underline{p_{J_i}}(1 - x_{J_i}) &\leq C_{J_i} \\
im + \frac{i-3}{2} - \frac{1}{2m} + (i-1)\varepsilon(m-1) + m &\leq (i+1)m + \frac{i-3}{2} - \frac{1}{2m} + \varepsilon(i-1)(m-1) \\
(i+1)m + \frac{i-3}{2} - \frac{1}{2m} + (i-1)\varepsilon(m-1) &\leq (i+1)m + \frac{i-3}{2} - \frac{1}{2m} + (i-1)\varepsilon(m-1) \\
0 &\leq 0
\end{aligned}$$

$$\begin{aligned}
S_i^{CPU} + \overline{p_{J_i}}x_{J_i} + \underline{p_{J_i}}(1 - x_{J_i}) &\leq C_{J_i} \\
(i-1)m + \frac{i-1}{2} - \varepsilon(i-1+\frac{i-1}{2m}) + m &\leq (i+1)m + \frac{i-3}{2} - \frac{1}{2m} + \varepsilon(i-1)(m-1) \\
im + \frac{i-1}{2} - \varepsilon(i-1+\frac{i-1}{2}) &\leq (i+1)m + \frac{i-3}{2} - \frac{1}{2m} + \varepsilon(i-1)(m-1) \\
0 &\leq m - 1 + \varepsilon(i-1)(m + \frac{i-1}{2m})
\end{aligned}$$

$$\begin{aligned}
S_i^{GPU} + \overline{p_{J_i}} x_{J_i} + \underline{p_{J_i}}(1 - x_{J_i}) &\le C_{J_i} \\
(i-1)m + \frac{i-1}{2} + \varepsilon((2i-2)m - \frac{i-1}{2m}) + m &\le (i+1)m + \frac{i-3}{2} - \frac{1}{2m} + \varepsilon(i-1)(m-1) \\
im + \frac{i-1}{2} + \varepsilon((2i-2)m - \frac{i-1}{2m}) &\le (i+1)m + \frac{i-3}{2} - \frac{1}{2m} + \varepsilon(i-1)(m-1) \\
0 &\le m - 1 + \varepsilon(i-1)(-m-1+\frac{i-1}{m})
\end{aligned}$$

The three inequalities are true. Thus, the constraints are satisfied for the type J_i, where $i \ge 3$ and odd.
If i is even:

$$\begin{aligned}
C_{J_{i-1}} + \overline{p_{J_i}} x_{J_i} + \underline{p_{J_i}}(1 - x_{J_i}) &\le C_{J_i} \\
im + \frac{i-4}{2} - \frac{1}{2m} + \varepsilon(i-2)(m-1) + m + 2\varepsilon(m-2) &\le (i+1)m + \frac{i-2}{2} - \frac{1}{2m} + i\varepsilon(m-1) \\
(i+1)m + \frac{i-4}{2} - \frac{1}{2m} + i\varepsilon(m-1) &\le (i+1)m + \frac{i-2}{2} - \frac{1}{2m} + i\varepsilon(m-1) \\
0 &\le 1
\end{aligned}$$

$$\begin{aligned}
S_i^{CPU} + \overline{p_{J_i}} x_{J_i} + \underline{p_{J_i}}(1 - x_{J_i}) &\le C_{J_i} \\
im - \frac{1}{2m} + \frac{i-2}{2} - \varepsilon(i-2+\frac{i-2}{2}) + m + \varepsilon(2m-2) &\le (i+1)m + \frac{i-2}{2} - \frac{1}{2m} + i\varepsilon(m-1) \\
(i+1)m + \frac{i-2}{2} - \frac{1}{2m} + \varepsilon(2m - i - \frac{i-2}{2}) &\le (i+1)m + \frac{i-2}{2} - \frac{1}{2m} + \varepsilon(mi - i) \\
0 &\le \varepsilon(m(i-2) + \frac{i-2}{2})
\end{aligned}$$

$$\begin{aligned}
S_i^{GPU} + \overline{p_{J_i}} x_{J_i} + \underline{p_{J_i}}(1 - c_{J_i}) &= (i-2)m + \frac{i}{2} + \frac{1}{2m} + \varepsilon((2i-4)m - \frac{i-2}{2m}) + m + \varepsilon(2m-2) \\
&= (i-1)m + \frac{i}{2} + \frac{1}{2m} + \varepsilon((2i-2)m - 2 - \frac{i-2}{2m})
\end{aligned}$$

So we have for the third constraint:

$$\begin{aligned}
S_i^{GPU} + \overline{p_{J_i}} x_{J_i} + \underline{p_{J_i}}(1 - x_{J_i}) &\le C_{J_i} \\
(i-1)m + \frac{i}{2} + \frac{1}{2m} + \varepsilon((2i-2)m - 2 - \frac{i-2}{2m}) &\le (i+1)m + \frac{i-2}{2} - \frac{1}{2m} + \varepsilon(mi - i) \\
0 &\le 2m - 1 - \frac{1}{m} + \varepsilon((m+1)(2-i) + \frac{i-2}{2m})
\end{aligned}$$

The three inequalities are true. Thus, the constraints are satisfied for the type J_i, where $i \ge 3$ and even.
The constraints are then satisfied for every task of type J_2 to J_l.

Finally, for the task T_{end}, which is a successor of the tasks of type A and J_l, Constraint 6.1 with the task T_A is:

$$\begin{aligned}
C_A &\le C_{end} \\
(l+2)m &\le (l+2)m \\
0 &\le 0
\end{aligned}$$

For the tasks T_{J_l}, with l odd:

$$\begin{aligned} C_{J_l} &\leq C_{end} \\ (l+1)m+\frac{l-3}{2}-\frac{1}{2m}+(l-1)\varepsilon(m-1) &\leq (l+2)m \\ 0 &\leq m-\frac{l-3}{2}-\varepsilon(l-1)(m-1) \end{aligned}$$

If l is even:

$$\begin{aligned} C_{J_l} &\leq C_{end} \\ (l+1)m+\frac{l-2}{2}-\frac{1}{2m}+l\varepsilon(m-1) &\leq (l+2)m \\ 0 &\leq m-\frac{l-2}{2}-\varepsilon(m-1)l \end{aligned}$$

The inequalities are true since $l \leq m$. Thus, Constraint 6.1 is satisfied. For the two remaning constraints, we can see the task T_{end} as a fictitious task $T_{J_{l+1}}$ with no processing time. Constraints 6.2Â and 6.3 reduce then to the following:

$$\begin{aligned} S_{l+1}^{CPU} &\leq C_{end} \\ S_{l+1}^{GPU}+\frac{p_A}{m} &\leq C_{end} \end{aligned}$$

And then for Constraint 6.2 if l is odd:

$$\begin{aligned} S_{l+1}^{CPU} &\leq C_{end} \\ (l+1)m+\frac{l-1}{2}-\frac{1}{2m}-\varepsilon(l-1+\frac{l-1}{2m}) &\leq (l+2)m \\ \frac{l-1}{2}-\frac{1}{2m}-\varepsilon(l-1+\frac{l-1}{2m}) &\leq m \end{aligned}$$

If l is even:

$$\begin{aligned} S_{l+1}^{CPU} &\leq C_{end} \\ lm+\frac{l}{2}-\varepsilon(l+\frac{l}{2}) &\leq (l+2)m \\ \frac{l}{2}-\varepsilon(l+\frac{l}{2}) &\leq 2m \end{aligned}$$

The two inequalities are true since $l \leq m$. Thus, Constraint 6.2 is satisfied.
For Constraint 6.3, if l is odd:

$$\begin{aligned} S_{l+1}^{GPU}+\frac{p_A}{m} &\leq C_{end} \\ (l-1)m+\frac{l+1}{2}+\frac{1}{2m}+\varepsilon((2l-2)m-\frac{l-1}{2m})+\frac{(l+2)m}{m} &\leq (l+2)m \\ \frac{3l}{2}+\frac{5}{2}+\frac{1}{2m}+\varepsilon((2l-2)m-\frac{l-1}{2m}) &\leq 3m \end{aligned}$$

If l is even:

$$\begin{aligned} S_{l+1}^{GPU} + \frac{p_A}{m} &\leq C_{end} \\ lm + \frac{l}{2} + \varepsilon(2lm - \frac{l}{2m}) + \frac{(l+2)m}{m} &\leq (l+2)m \\ \frac{3l}{2} + 2 + \varepsilon(2lm - \frac{l}{2m}) &\leq 2m \end{aligned}$$

The two inequalities are true since $l \leq m$. Thus, Constraint 6.3 is satisfied.

We conclude that the set of variables x_j and C_j is effectively an optimal solution of $(LP2)$, with $\lambda = (l+2)m$. $\square$