Approximation algorithm for scheduling applications on hybrid multi-core machines with communications delays
Massinissa Ait Aba, Lilia Zaourar, Alix Munier-Kordon

To cite this version:
Massinissa Ait Aba, Lilia Zaourar, Alix Munier-Kordon. Approximation algorithm for scheduling applications on hybrid multi-core machines with communications delays. IPDPS Workshops 2018, 2018 IEEE International Parallel and Distributed Processing Symposium Workshops, May 2018, Vancouver, Canada. pp.36-45, 10.1109/IPDPSW.2018.00016. hal-02073600

HAL Id: hal-02073600
https://hal.science/hal-02073600
Submitted on 20 Mar 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Approximation algorithm for scheduling applications on hybrid multi-core machines with communications delays

Massinissa Ait Aba, Lilia Zaourar
CEA, LIST, Computing and Design Environment Laboratory
91191 GIF SUR YVETTE CEDEX, FRANCE.
Emails:{massinissa.aitaba, lilia.zaourar}@cea.fr

Alix Munier
LIP6-UPMC,
4 place Jussieu,
75005 Paris, France.
Email:alix.munier@lip6.fr

Abstract—This paper presents an efficient algorithm with performance guarantee (approximation algorithm) to solve task scheduling problem on hybrid platform. The underlying platform architecture in this work is composed by two types of resources CPU and GPU, often called hybrid parallel multi-core platforms. We consider here for each type of resource identical nodes with communications delays. We focus in finding a generic approach to schedule applications presented by DAG (Directed Acyclic Graph) that minimizes makespan by considering communication delay between processors and tasks. A 6-approximation scheduling algorithm is proposed and evaluated in comparison to exact solutions and to another method. We demonstrate that the proposed algorithm achieves a close-to-optimal performance. Finally, our algorithm has been experimented on a large number of instances. These tests assess the good practical behavior of the algorithms with respect to the state-of-the-art solutions whenever these exist.

Index Terms—DAG applications, makespan, hybrid CPU GPU, approximation algorithm, scheduling.

1. Introduction

The past few years have seen an increase demand for developing efficient large computational resources. Thus, heterogeneous computing systems become a popular and powerful commercial platform, containing several heterogeneous processing elements such as Central Processing Unit (CPU), Graphics Processing Unit (GPU) and some Field Programmable Array (FPGA) with different computational characteristics. In particular, the number of platforms of the TOP500 [1] equipped with accelerators has significantly increased during the last years. However, using efficiently these platforms became very complicated. Consequently, more and more attention has been focused on scheduling techniques for solving the problem of optimizing the execution of parallel applications on heterogeneous computing systems [4], [18], [25].

The underlying platform architecture in this work is composed by two types of resources CPU and GPU often called hybrid parallel multi-core platforms. We consider here for each type of resource identical platforms. In several applications, we always observe an acceleration of the execution time of tasks if they are executed on a GPU compared to their execution time on a CPU [8]. However, we consider here the more general case where the relation between the two resources can differ for different tasks. Thus we have to take into account that the execution time for any task of the application depends on the type of resource used to execute it. This configuration is known as “inconsistent heterogeneity” and has been well-studied in the literature. The consistent model problem was presented in [3] for the particular case of applications represented by chain of tasks, where the execution time of the task depends on the frequency of the processor to which it is assigned.

We focus here in finding a generic approach to schedule applications presented by DAG (Directed Acyclic Graph) that minimizes finish time of the application by considering communication delay between processors and tasks. 6-approximation scheduling algorithm is proposed and evaluated compared to exact solution and to another method. We demonstrate that the proposed algorithm achieves a close-to-optimal performance. The goal here is to minimize the finish execution time of the last task of the application (usually called makespan).

The rest of the paper is organized as follow: Section 2 investigates previous research in scheduling strategies to minimize makespan on hybrid platforms. Section 3 presents the detailed problem with mathematical formulation. In Section 4, we describe the proposed algorithm for our problem and the approximation ratio we obtain. Section 5 shows some preliminary numerical results. Finally, we conclude and provide insight for future work in Section 6.

2. Related work

There is a plentiful literature about scheduling in heterogeneous and hybrid platforms CPU/GPUs that concerns specific applications [9], [13]. Only few papers deal with generic scheduling strategies in hybrid platforms, and very
few of them consider precedence constraints but usually without communications costs as it is the case in this work.

List Scheduling algorithm [6], [16], [23] was widely used because of their ease of implementation as well as their low complexity. We can find comparison between some basic List Scheduling algorithms in [23], [27] on different environments. However, such heuristic algorithms do not provide performance guarantees and thus, may lead to very bad executions for some instances.

Most of these strategies as in [10], [21] work in two main steps. The first step assigns ranks based on certain properties of the tasks, usually the execution time and/or communication delays. In the second step, the tasks are assigned to the processors.

For unlimited number of processors or homogeneous platforms, clustering algorithms [12], [19], [24] usually provide a good solution, where multiple tasks are combined at each step into a cluster to be assigned. Another method to solve the problem is duplication algorithms [2], [7] which use the concept of multiple copies of a task that can run on multiple processors to reduce communication time between processors but generate additional data transfer costs between tasks and may also increase energy consumption.

Among the algorithms proposed in the literature, Heterogeneous Earliest Finish Time (HEFT) [26] is one of the first work that deal with scheduling problem on heterogeneous platform. It served as a comparison method for most of works. HEFT is a list based approach on two main phases. The first phase uses runtime costs and communication costs to calculate ranks. After rank calculation, the assignment to the processors will take place in the second phase using the earliest finish time of tasks. Each task is then assigned to the processor that produces the minimal finishing time. However, HEFT does not consider more than one task during processor assignment. A different approach has been presented in [6], proposing Predict Earliest Finish Time (PEFT) algorithm. This algorithm has also two main phases: a phase that calculates task priorities, and a processor selection phase to choose the best processor for running the current task. Recently, an algorithm with a new strategy named INCSEFT (Incremental Sub-graph Earliest Finish Time) has been proposed in [22] for the heterogeneous platform scheduling. It incorporates the use of a sub-graph that grows progressively by adding critical paths. Critical paths are calculated dynamically using ranks based on the average execution costs of the tasks. All tasks in a critical path are assigned to the most appropriate processor if the length of the sub-graph scheduling does not exceed the length of the previously computed scheduling. Otherwise, a single task is assigned to the most appropriate processor.

Inspired by research before, we tried to solve our problem on two steps: define the assignment of the tasks, then look for an optimal scheduling. Contrarily to most existing approaches, we propose here to address the problem of assignment tasks on an hybrid platform by considering communications costs between tasks and resources which reflects better the reality.

We took over the work presented in [20] where the two-phases approach has been proposed for the problem of scheduling a parallel application whose tasks are linked by precedence constraints without communications delays. The first phase consists in solving the assignment problem to find the type of processor assigned to execute the tasks (CPU or GPU) using a linear program. In the second phase, a list scheduling algorithm has been proposed to generate a feasible schedule. This algorithm (which we call GPU-CPU Scheduling or simply GCS) achieves an approximation ratio of 6. This ratio has been proven that it is tight in [5]. We tried to keep the same ratio of 6 for the scheduling with communication costs by adding new constraints. We can notice that for the problem without communication costs, a 2(K + 1)-approximation algorithm has been developed in [15] using a platform having processors with K different speeds. To the best of our knowledge, we propose the first algorithm that takes precedence constraints and communications delays into account for scheduling a parallel application on hybrid multi-core machines.

3. Problem definition

We consider in this work a heterogeneous platform composed of m resources of two types: GPU and CPU. Let ℓ be the number of CPU and k the number of GPU, m = ℓ+k. An application A of n tasks is represented by a Directed Acyclic Graph (DAG) oriented G(V, E), each vertex represents a task ti. Each arc e = {ti, tj} represents a precedence constraint between two tasks ti and tj. We associate it with the value ctij which represents the communication delay between ti and tj if they are executed on two different resource types. The exact formula to evaluate ctij which takes into consideration latencies and available bandwidth between processors is provided in [28]. We denote by Γ−(i) (resp. Γ+(i)) the sets of the predecessors (resp. successors) of task ti. Figure 1 presents an example of our application.

![Figure 1: DAG application.](image-url)
A task can be executed by a CPU or a GPU. Executing the task \( t_i \) on a CPU (resp. GPU) generates an execution time equal to \( w_{i,0} \) (resp. \( w_{i,1} \)). A task \( t_i \) can be executed only after the complete execution of its predecessors. We do not allow duplication of tasks and preemption. We denote by \( C_{max} \) the completion time of the application (makespan). The aim is to find the minimum makespan of the application.

Our problem can be modeled by mixed integer quadratic constrained program (\( P \)). The decision variables are \( x_{i,j} \) and \( \text{start}_i \) for \( i = 1..n \) and \( j = 1..m_i \), where \( x_{i,j} = 1 \) if \( t_i \) is executed on the CPU for \( j \in [1,\ell] \) (resp. GPU for \( j \in [\ell+1,m] \)), 0 otherwise. \( \text{start}_i \) presents the starting time of the task \( t_i \).

The first constraint simply expresses that each task must be executed only once and on one processor. Constraints (2) describes that the task \( t_{i2} \) must be executed after the completion time of the task \( t_{i1} \) for each task \( t_{i2} \) that precedes task \( t_{i2} \), and the communication cost \( c_{l1,2} \) is added if they are executed on two different processing elements. Constraint (3) is a disjunctive constraint to prohibit overlapping tasks on the same processor using a large constant \( B \), such that if two tasks \( t_{i1} \) and \( t_{i2} \) are executed on the same processor, then either \( t_{i2} \) starts after the completion time of the task \( t_{i1} \) or \( t_{i1} \) starts after the completion time of the task \( t_{i2} \). Constraints (4) describes that \( C_{max} \) is bigger than the completion time of the tasks without successors. Thus, to minimize the finish execution time of the application, we must minimize \( C_{max} \).

\[
\begin{align*}
\sum_{j=1}^{m} x_{i,j} &= 1, \forall i = 1..n \quad (1) \\
\text{start}_i + x_{i,j} w_{i,j} &\leq \text{start}_{i2} + x_{i,j} w_{i,j} + x_{i,j} x_{i,j2} c_{i,j,j2} \quad (2) \\
&\forall i \rightarrow i_2, \forall j_1 = 1..m_i, \forall j_2 = 1..m_i, j_1 \neq j_2 \\
&\text{start}_i + x_{i,j} w_{i,j} &\leq \text{start}_{i2} + B \times (1 - x_{i,j} \times x_{i,j2}) \quad (3) \\
&\forall i, i_2 = 1..n, i_1 \neq i_2, \forall j = 1..m_i, B = Cte \\
&\text{start}_i + \sum_{j=1}^{m} x_{i,j} w_{i,j} &\leq C_{max}, \forall i, \Gamma^+(i) = \emptyset \quad (4) \\
\end{align*}
\]

4. Approximation algorithm for scheduling DAG applications

In this section, we propose a two-phase approximation algorithm, aiming for a ratio of 6. We start by solving an assignment problem to find which processor (CPU or GPU) will execute each task. We propose two models (P1) and (P2) for solving the assignment problem while the precedence constraints are satisfied. The solution obtained by the model (P1) or (P2) represents a lower bound for the final makespan. Then we solve the relaxation (P1') (resp. (P2')) of the model (P1) (resp. (P2)). And in order to obtain a feasible assignment for the tasks, we rounded up the fractional solution of the program (P1') and (P2'). In the second phase, we use the assignment of the tasks and list scheduling algorithm to get a feasible schedule.

4.1. Phase 1: assignment of tasks

4.1.1. Mathematical model.

Let the decision variable \( x_{i,j} \) which is equal to 1 if the task \( t_i \) is assigned to a CPU and 0 otherwise. We introduce two binary variables \( z_{i,j} \) and \( y_{i,j} \) such that \( z_{i,j} = 1 \) if the tasks \( t_i \) and \( t_j \) are assigned to a CPU and \( y_{i,j} = 1 \) if the tasks \( t_i \) and \( t_j \) are assigned to a GPU. Let \( C_{i} \) be the finish time of the task \( t_i \). The goal is to minimize the makespan.

The model is inspired by the model given in [20]. Constraints (1) to (7) describes the critical path, such as if task \( t_i \) precedes \( t_j \), and these two tasks are assigned to different processors, we obtain two cases: either \( x_{i,j} = 1 \) and \( x_{j,i} = 0 \) or \( x_{i,j} = 0 \) and \( x_{j,i} = 1 \). In the two cases, we obtain \( y_{i,j} = 0 \) and \( z_{i,j} = 0 \), which is equal to the four constraints (2), (3), (4) and (5).

If tasks \( t_i \) and \( t_j \) are assigned to the same processor, we obtain also two cases:

- case 1: \( x_{i,j} = 0 \) and \( x_{j,i} = 0 \): in this case, \( z_{i,j} = 0 \) and \( y_{i,j} \) takes the value 1 (minimization problem), then \( 1 - y_{i,j} - z_{i,j} = 0 \).
- case 2: \( x_{i,j} = 1 \) and \( x_{j,i} = 1 \): in this case, \( y_{i,j} = 0 \) and \( z_{i,j} \) takes the value 1 (minimization problem), then \( 1 - y_{i,j} - z_{i,j} = 0 \).

Tasks without predecessors (respectively successors) are considered in the constraint (6) (resp. (7)). Constraint (8) (resp. (9)) simply expresses that the makespan cannot be smaller than the average load of work putted in CPUs (resp. GPUs). Note that the problem of finding the optimal mapping that minimizes makespan is np-hard even for the problem without communications delays [14, 25].

The first constraint contains an absolute value that can be processed in the CPLEX APIs [17]. In the C + + API, Cplex.abs can be used. To see the efficiency of CPLEX in managing the absolute value, we have proposed another model (P2) with absolute value. By adding two binary variables \( a_{i,j} \) and \( b_{i,j} \) and two constraints (5.1) and (5.2), we can get rid of the absolute value, we obtain a second model (P2).

The models (P1) and (P2) are equivalent. Indeed, the value of variable \( b_{i,j} \) replace \( |y_{i,j} - z_{i,j}| \in \{0,1\} \) in (P2), we obtain two cases:
In the following, we focus on the model (P1), the results found for (P1) remain valid for (P2).

4.1.2. Relaxed problem. We obtain the model (P1') by relaxing the integrality variables $x_i, y_{i,j}$ and $z_{i,j}$. We also obtain the model (P2') by relaxing the integrality variables $x_i, y_{i,j}, z_{i,j}$ and $b_{i,j}$. However, $a_{i,j}$ must remain integer. We denote by $y_{i,j}', z_{i,j}', b_{i,j}'$ all in $[0,1]$, the fractional value of $y_{i,j}, z_{i,j}, b_{i,j}$ in the optimal solution of the model (P1') or (P2'), with $i = \overline{1,n}$ and $j = \overline{1,m}$. We denote by $x_i'$ the fractional value of the assignment variable of task $t_i$ in the optimal solution of the model (P1') or (P2'). If $x_i'$ is integer for $i \in \overline{1,m}$, the solution obtained is feasible and optimal for (P1) and (P2), otherwise the fractional values are rounded. We denote by $x_i''$ the rounded value of the fractional value of the assignment variable of task $t_i$ in the optimal solution of (P1') or (P2'). We set $x_i'' = 0$ if $x_i' < \frac{1}{2}$, $x_i'' = 1$ otherwise.

Let $\theta_i$ be the mapping obtained by this rounding. Each task $t_i$ is mapped in either CPU or GPU. Thus, $\theta_i(t_i) \rightarrow \{CPU, GPU\}$.

**Proposition 1.** The rounding previously defined satisfies the following two inequalities:

$$x_i'' \leq 2x_i' \leq 2(1-x_i').$$

**Proof:** If $0 \leq x_i' < \frac{1}{2}$ then $x_i'' = 0 \leq 2x_i'$. Furthermore, $2x_i' \leq 1$, then $0 \leq 1-2x_i'$, follows $-x_i'' = 0 \leq 1-2x_i'$, then $1-x_i'' \leq 2(1-x_i')$. If $\frac{1}{2} \leq x_i'$ then $1 \leq 2x_i'$ follows $x_i'' = 1 \leq 2x_i'$. Furthermore, $x_i' \leq 1$ then $-2x_i' \geq -2$, follows $1-2x_i'' \geq -1$, then $-x_i'' = -1 \leq 1-2x_i'$, then $1-x_i'' \leq 2(1-x_i').$

**Lemma 1.** Let $C_{\text{max}}$ be the optimal solution obtained by solving the programs (P1') or (P2'). We can get another solution $C'_{\text{max}} = C_{\text{max}}$, such that for each two tasks $t_i$ precedes $t_j$:

1) if $\min\{1-x_i', 1-x_j'\} \geq \min\{x_i', x_j'\}$, then $y_{i,j}' = \min\{1-x_i', 1-x_j'\}$ and $z_{i,j}' = 0$.

2) if $\min\{1-x_i', 1-x_j'\} < \min\{x_i', x_j'\}$, then $y_{i,j}' = 0$ and $z_{i,j}' = \min\{x_i', x_j'\}$.

**Proof:** By constraints (2) and (3) from (P1') or (P2'), $z_{i,j}' \leq \min\{x_i', x_j'\}$. By constraints (4) and (5) from (P1') or (P2'), $y_{i,j}' \leq \min\{1-x_i', 1-x_j'\}$. Let $\beta = |y_{i,j}' - z_{i,j}'|$. To minimize the communication cost $(\zeta_{i,j} = (1-|y_{i,j}' - z_{i,j}'|)ct_{i,j})$ between $t_i$ and $t_j$, we have to maximize $\beta$. $\beta = |y_{i,j}' - z_{i,j}'| \leq \max\{y_{i,j}', z_{i,j}'\} \leq \max\{1-x_i', 1-x_j'\} \min\{x_i', x_j'\}$. Then, if $\min\{1-x_i', 1-x_j'\} \geq \min\{x_i', x_j'\}$, we can put $y_{i,j}' = \min\{1-x_i', 1-x_j'\}$ and $z_{i,j}' = 0$ to maximize $\beta$. Otherwise, we can put $z_{i,j}' = \min\{x_i', x_j'\}$ and $y_{i,j}' = 0$. 

In the following, we suppose that the solution obtained by solving the programs (P1') or (P2') follows the properties given by the Lemma 1.

Let two tasks $t_i$ and $t_j$, such that $t_i$ precedes $t_j$. Let $\alpha_{i,j}$ the value given by $\alpha_{i,j} = 1-|y_{i,j}' - z_{i,j}'|$ with replacing $y_{i,j}'$ and $z_{i,j}'$ by their values obtained by model (P1') or (P2'). In the following, we look for the relation between the value of $\alpha_{i,j}$ and the assignment of the tasks $t_i$ and $t_j$.

**Remark 1.** $\alpha_{i,j} \geq 0$. Indeed, if $y_{i,j}' = \min\{1-x_i', 1-x_j'\} \leq 1$ and $z_{i,j}' = 0$, then $\alpha_{i,j} = 1 - y_{i,j}' \geq 0$. Furthermore, if $z_{i,j}' = \min\{x_i', x_j'\} \leq 1$ and $y_{i,j}' = 0$, then $\alpha_{i,j} = 1 - z_{i,j}' \geq 0$.

**Lemma 2.** If $t_i$ and $t_j$ are executed by two different processing elements, then $\alpha_{i,j} \geq \frac{1}{2}$.

**Proof:** Let two tasks $t_i$ and $t_j$ executed on two different processing elements, we obtain two cases:

a. $x_i' < \frac{1}{2}$ and $x_j' \geq \frac{1}{2}$; from constraints (2) and (3), $z_{i,j} < \frac{1}{2}$. Furthermore, $1-x_i' > \frac{1}{2}$ and $1-x_j' \leq \frac{1}{2}$, then, from constraints (4) and (5), $y_{i,j}' \leq \frac{1}{2}$. Finally, $\alpha_{i,j} = 1 - |y_{i,j}' - z_{i,j}'| \geq 1 - \max\{y_{i,j}', z_{i,j}'\} \geq \frac{1}{2}$.

b. $x_j' \geq \frac{1}{2}$ and $x_i' < \frac{1}{2}$; from constraints (2) and (3), $z_{i,j} < \frac{1}{2}$. Furthermore, $1-x_i' \leq \frac{1}{2}$ and $1-x_j' > \frac{1}{2}$, then, from constraints (4) and (5), $y_{i,j}' \leq \frac{1}{2}$. Finally, $\alpha_{i,j} = 1 - |y_{i,j}' - z_{i,j}'| \geq 1 - \max\{y_{i,j}', z_{i,j}'\} \geq \frac{1}{2}$.

Let two tasks $t_i$ and $t_j$, such that $t_i$ precedes $t_j$. We denote by $Cost^i_{t_i,j}$ the value given by $Cost^i_{t_i,j} = 0$ if $x_i' = x_j'$, $Cost^i_{t_i,j} = ct_{i,j}$ otherwise.
Proposition 2. \( \text{Cost}^{t}_{i,j} \leq 2 \alpha_{i,j} c_{t_{i,j}} \).

Proof: If \( t_{j} \) and \( t_{i} \) are executed by the same processing element, \( \text{Cost}^{t}_{i,j} = 0 \leq 2 \alpha_{i,j} c_{t_{i,j}} \), because \( \alpha_{i,j} \geq 0 \). If \( t_{j} \) and \( t_{i} \) are executed by two different processing elements, then \( \text{Cost}^{t}_{i,j} = c_{t_{i,j}} \). Then, from the lemma 2, \( \alpha_{i,j} \geq \frac{1}{2} \), then \( 2 \alpha_{i,j} \geq 1 \), follows \( \text{Cost}^{t}_{i,j} = c_{t_{i,j}} \leq 2 \alpha_{i,j} c_{t_{i,j}} \). \( \square \)

4.2. Phase 2: scheduling algorithm

We note by \( \text{EST}_{i} \) the Earliest Start Time of the task \( t_{i} \) according to the mapping \( \theta_{1} \). The following algorithm builds a feasible schedule. The mapping \( \theta_{1} \) is used to define on which processing element to execute which task (CPU or GPU). The algorithm determines for a task order given by a list \( L \), the corresponding scheduling by executing the first task ready of the list as long as there are free processing elements.

Algorithm 1: List Scheduling (LS) algorithm.

Data: \( T = \{t_{1}, t_{2}, ..., t_{n}\} \), mapping \( \theta_{1} \), list \( L \).
Result: feasible scheduling.

\[
\text{begin} \\
\quad S \leftarrow \emptyset \\
\quad \text{while } S \neq T \text{ do} \\
\quad \quad \text{RD} = \{t_{i} \in T; \Gamma^{-}(t_{i}) \subseteq S\}; \\
\quad \quad \text{EST} = \min\{\text{EST}_{j}, t_{j} \in \text{RD}\}; \\
\quad \quad \text{Let } t_{k} \in \text{RD} \text{ the first task following the order of list } L \text{ such that } \text{EST}_{k} = \text{EST}; \\
\quad \quad \text{Execute } t_{k} \text{ at } \text{EST} \text{ according to } \theta_{1}; \\
\quad \quad S \leftarrow S \cup \{t_{k}\}; \\
\text{end}
\]

The list \( L \) can be defined by different way, \( n! \) lists are possible. In the following, we propose some lists that will be used for the experimentations.

4.2.1. List by using the model \( (P1') \) or \( (P2') \). Two interesting lists extracted from the resolution of model \( (P1') \) or \( (P2') \) can be used for algorithm 2. \( \text{LST} \) (List by Start Time) and \( \text{LFT} \) (List by Finish Time). The \( \text{LST} \) (resp. \( \text{LFT} \)) list can be obtained by sorting the tasks in ascending order of their processing start time (resp. processing finish time) obtained by solving the model \( (P1') \) or \( (P2') \). Let \( \text{Start}_{i} \) (resp. \( C_{i}' \)) be the processing start time (resp. processing finish time) of the task \( t_{i} \) obtained by solving the model \( (P1') \) or \( (P2') \). Let \( \text{Start}_{1} \), \( \text{Start}_{2} \), ..., \( \text{Start}_{n} \) be the ordering of the tasks in the list \( L \) obtained by solving the model \( (P1') \) or \( (P2') \), \( i = 1, ..., n \). Thus, \( \text{LST} = \{\text{Start}_{1} \leq \text{Start}_{2} \leq ... \leq \text{Start}_{n}\} \), \( \text{LFT} = \{\text{Start}_{1} \leq \text{Start}_{2} \leq ... \leq \text{Start}_{n}\} \), \( \text{LFT} = \{\text{Start}_{1} \leq \text{Start}_{2} \leq ... \leq \text{Start}_{n}\} \).

4.2.2. List by longest path (LLP). In the first, we start by defining graph \( G'(V, E) \), with \( V = \{t_{1}, t_{2}, ..., t_{n}\} \) and \( E \) represent the set of graph edges. The vertices are labelled by the execution time of each task according to their assignments. The edges are labelled by the communication costs if \( t_{j} \) precedes \( t_{i} \) and \( x_{j} \neq x_{j} \), 0 otherwise. Then, we can calculate the longest path \( PL_{i} \) from each task \( t_{i} \) to its last successor. The list \( LLP \) is given by \( LLP = \{t_{1}, t_{2}, ..., t_{n}\} \), such that \( PL_{1} \geq PL_{2} \geq ... \geq PL_{n} \).

4.3. Algorithm analysis

4.3.1. Lower bound. We note by \( C_{\text{max}} \) the optimal solution obtained by \( (P1') \) or \( (P2') \), this solution is a lower bound for the optimal solution of our problem \( C'_{\text{max}} \). \( C'_{\text{max}} \) is bounded by:

1) \( L(P_{r}) \): length of the fractional critical path \( P_{r} \) in the optimal solution of the program \( (P1') \) or \( (P2') \).
2) \( W_{CPU}^f \): The fractional weight of the tasks allocated to the CPU in the optimal solution of the program \( (P1') \) or \( (P2') \) divided by \( \ell \), with \( W_{CPU}^{f} = \sum_{i=1}^{n} x_{i} w_{i,0} \).
3) \( W_{GPU}^f \): The fractional weight of the tasks allocated to the GPU in the optimal solution of the program \( (P1') \) or \( (P2') \) divided by \( k \), with \( W_{GPU}^{f} = \sum_{i=1}^{n} (1 - x_{i}) w_{i,1} \).

Furthermore, the solution \( \tilde{C}_{\text{max}} \) obtained by Algorithm 1 is bounded by \( L(P_{r}) \), \( W_{CPU}^{f} \), \( W_{GPU}^{f} \), length of the critical path \( P_{r} \) and the works on CPUs divided by \( \ell \) and the works on GPUs divided by \( k \) in the final scheduling.

4.3.2. Worst case approximation ratio. We note by \( A \) (resp. \( I \)) the cumulative sum of periods of activity (resp. inactivity) where the processors are busy (resp. idle). Let \( A_{1} = \sum_{i=1}^{n} x_{i} w_{i,0} \) (resp. \( A_{2} = \sum_{i=1}^{n} (1 - x_{i}) w_{i,1} \)) be the cumulative sum of periods of activity of all CPUs (resp. GPUs), \( A = A_{1} + A_{2} \). Let \( I_{1} \) (resp. \( I_{2} \)) be the cumulative sum of periods of inactivity where all CPUs (resp. GPUs) are busy and all GPUs (resp. CPUs) are idle. The maximum duration where all CPUs (resp. GPUs) are busy is \( \frac{A}{\ell} \) (resp. \( \frac{A_{1}}{\ell} \)), then \( I_{1} \leq \frac{k A}{\ell} \) (resp. \( I_{2} \leq \ell \frac{A_{1}}{k} \)). Let \( I_{3} \) the cumulative sum of periods of inactivity where at least one CPU and one GPU are idle, \( I = I_{1} + I_{2} + I_{3} \). Figure 2 represents the occupation of processing elements, during the scheduling of an application.

![Figure 2: Occupation of processing elements.](image)

By multiplying \( \tilde{C}_{\text{max}} \) by the number of processors, we find the cumulative sum of the periods of activity and inactivity, \( (\ell + k)\tilde{C}_{\text{max}} = A + I \).

We look now for the ratio between \( \tilde{C}_{\text{max}} \) and \( C_{\text{max}}' \). For this purpose, we try to limit \( A \) and \( I \) with formulas in functions of \( C_{\text{max}}' \).
Proposition 3.

\[ A_1 \leq 2IC_{max} \]
\[ A_2 \leq 2kC_{max} \]

Proof: By definition, \( A_1 = \sum_{i=1}^{n} x_i^0w_{i,0} \). From Proposition 1, \( x_i^0 \leq 2x_i \). Then, \( A_1 = \sum_{i=1}^{n} x_i^0w_{i,0} \leq 2\sum_{i=1}^{n} x_i w_{i,0} \leq 2W_{CPU} \leq 2IC_{max} \leq 2IC_{max} \). Furthermore, by definition, \( A_2 = \sum_{i=1}^{n} (1 - x_i)w_{i,1} \). From Proposition 1, \( (1 - x_i) \leq 2(1 - x_i) \). Then, \( A_2 = \sum_{i=1}^{n} (1 - x_i)w_{i,1} \leq \sum_{i=1}^{n} 2(1 - x_i)w_{i,1} = 2W'_{GPU} \leq 2kC_{max} \leq 2kC_{max} \).

Corollary 1. If for each task \( t_i \), \( x_i^0 \) is integer, such that \( x_i^0 = x_i \) and \( (1 - x_i^0) = (1 - x_i) \) for \( i = 1, n \), then the mapping \( \theta_1 \) is optimal. Follow, \( A_1 = \sum_{i=1}^{n} x_i^0w_{i,0} = \sum_{i=1}^{n} x_i w_{i,0} \leq IC_{max} \) and \( A_2 = \sum_{i=1}^{n} (1 - x_i^0)w_{i,1} = \sum_{i=1}^{n} (1 - x_i)w_{i,1} \leq kC_{max} \).

Corollary 2.

\[ I_1 \leq 2kC_{max} \]
\[ I_2 \leq 2IC_{max} \]

Proof:

\[ I_1 \leq kA_1 \leq k2IC_{max} = 2kC_{max} \]. Furthermore, \( I_2 \leq \frac{IA_2}{k} \leq \frac{2kC_{max}}{k} = 2C_{max} \).

Proposition 4. \( I_3 \leq 2(\ell + k)C_{max} \).

Proof: There exists a critical path \( \gamma \) in the final scheduling such that the sum of the instants where at least one CPU and one GPU are idle is less than 2L(\( P \)). Indeed, we assume that the tasks are stalled on the left. Let \( t_0 \) be the last task, such as during the execution of \( t_0 \), there is an idle CPU and an idle GPU. Let \( \theta_0 \) be the processing start time of the task \( t_0 \). If there is an idle CPU and idle GPU before \( \theta_0 \), then \( t_0 \) has a predecessor \( t_1 \) that ends before \( \theta_0 \), the idle slots between \( \theta_1 \) and \( \theta_0 \) are covered either by the execution of the task between \( t_1 \) and \( \theta_0 \) or by the communication cost between \( t_1 \) and \( \theta_0 \) and its successor. \( t_0 \) can be \( t_0 \) or a task on the path from \( t_1 \) to \( t_0 \). If there is an idle CPU and idle GPU before \( \theta_1 \), then \( t_1 \) has a predecessor \( t_2 \) that ends before \( \theta_1 \) which can be obtained in the same precedent way. Let \( t_0 \), \( t_1 \), \( t_1 \), ..., \( t_n \) be the maximum sequence of tasks obtained. There is no more slots before \( \theta_n \) where at least one CPU and one GPU are idle. If \( \gamma \) the path containing all these tasks which covers all periods when at least one CPU and one GPU are idle, let \( L(\gamma) \) its length. From Proposition 1, for any task \( t_i \) in \( P \), the processing time of \( t_i \) in the final scheduling will be at most twice the fractional solution obtained by the program \((P_1)\) or \((P_2)\).

For any two tasks \( t_i \), \( t_j \) in \( P \), from the Proposition 2, the communication cost \( Cost_{i,j} \) between \( t_i \) and \( t_j \) in the final scheduling will increase by at most twice the fractional communication cost \( \alpha_{i,j}c_{t_i,j} \) obtained by the program \((P_1)\) or \((P_2)\). Then, \( L(\gamma) \leq L(P) = \sum_{i \in P}(x_iw_{i,0} + (1 - x_i^0w_{i,1}) + \sum_{(t_i,t_j) \in P} Cost_{i,j} \leq \sum_{i \in P}(2x_iw_{i,0} + 2(1 - x_i^0)w_{i,1} + \sum_{(t_i,t_j) \in P} 2\alpha_{i,j}c_{t_i,j} \leq 2L(P)). \) Finally, \( I_3 \leq (\ell + k)L(P) \leq 2(\ell + k)L(P) \leq 2(\ell + k)C_{max} \).

Theorem 1. The ratio between the solution \( \hat{C}_{max} \) obtained by Algorithm 1 (LS) and the optimal scheduling solution \( C_{max} \) is given by \( \frac{\hat{C}_{max}}{C_{max}} \leq 6 \).

Proof:

\[ \hat{C}_{max} = \frac{A_1 + A_2 + 1 + 1}{\ell + k} \leq \frac{A_1 + A_2 + 1 + 1}{\ell + k} \leq 6C_{max} \]. Then, from Proposition 3, 4 and Corollary 2, \( \hat{C}_{max} \leq \frac{2(\ell + 2k + 2(\ell + k))C_{max}}{\ell + k} = 6C_{max} \). Finally, \( \frac{\hat{C}_{max}}{C_{max}} \leq 6 \).

Corollary 3. If the mapping \( \theta_1 \) is optimal, then \( \frac{\hat{C}_{max}}{C_{max}} \leq 5 \).

Proof:

\[ \frac{\hat{C}_{max}}{C_{max}} \leq \frac{A_1 + A_2 + 1 + 1}{\ell + k} \leq \frac{\ell^2 + 2k + 2(\ell + k)C_{max}}{\ell + k} \leq 5C_{max} \]. Finally, \( \frac{\hat{C}_{max}}{C_{max}} \leq 5 \).

4.4. Iterative mapping

In order to find a more efficient rounding than \( \theta_1 \) described previously, we try to assign the tasks progressively. Let \( \theta_2 \) be the rounding obtained by the following algorithm 2.

Algorithm 2: Rounding algorithm.

\begin{algorithm}
\begin{algorithmic}
\STATE Data: model \( (P) \) (\( (P_1) \) or \( (P_2) \)), \( v \geq 2 \).
\STATE \textbf{Result:} mapping \( \theta_2 \).
\end{algorithmic}
\end{algorithm}

\begin{algorithm}
\begin{algorithmic}
\STATE begin
\FOR {\( i = 1 \) to \( \frac{v}{2} \)}
\STATE Solve \( P \);
\FOR {\( j = 1 \) to \( n \)}
\IF {\( x_j^i < \frac{1}{v} \)}
\STATE Set in \( P : x_j = 0 \)
\ENDIF
\IF {\( x_j^i \geq 1 - \frac{1}{v} \)}
\STATE Set in \( P : x_j = 1 \)
\ENDIF
\FOR {\( l = 1 \) to \( n \)}
\IF {\( x_j^i \leq \frac{1}{2} \)}
\STATE \( x_j^i = 0 \)
\ELSE
\STATE \( x_j^i = 1 \)
\ENDIF
\ENDFOR
\ENDFOR
\end{algorithmic}
\end{algorithm}

For a given integer \( v \geq 2 \), we solve the model \( (P_1) \) or \( (P_2) \) \((\frac{v}{2})\) times adding new assignment constraint at each resolution. Contrary to \( \theta_1 \), we try to assign the tasks progressively, starting by setting \( x_j^i \) to 0 if \( x_j^i < \left(\frac{1}{2}\right) \) and \( x_j^i \) to 1 if \( x_j^i \geq 1 - \left(\frac{1}{v}\right) \) for the first resolution of \( (P_1) \) or \( (P_2) \), and finishing by setting \( x_j^i \) to 0 if \( x_j^i < \left(\frac{1}{2}\right) \) and \( x_j^i \) to 1 if \( x_j^i \geq 1 - \left(\frac{1}{v}\right) \), where \( \left(\frac{1}{2}\right) \leq \frac{1}{2} \) according to the Lemma 3.
Remark 2. For \( v = 2 \), we obtain the rounding \( \theta_1 \).

Lemma 3. For an integer \( v \geq 2 \), \( \frac{\lfloor \frac{v}{2} \rfloor}{v} \leq \frac{1}{2} \).

Proof:
1) \( v \) is even, \( v = 2\lambda : \frac{\lfloor \frac{v}{2} \rfloor}{v} = \frac{\lambda}{2\lambda} = \frac{1}{2} \).
2) \( v \) is odd, \( v = 2\lambda + 1 : \frac{\lfloor \frac{v}{2} \rfloor}{v} = \frac{\lambda}{2\lambda + 1} < \frac{1}{2} \). \qed

Remark 3.
The specialized accelerator problem can be solved with the same method, where a set of tasks \( T' \in T \) can be executed by either a CPU or a GPU only. We denote by \( w_i \) the execution time of each task \( t_i \in T' \). In fact, assuming that a task \( t_i \in T' \) is executable only by a CPU (resp. GPU), we can put \( w_{i,0} = w_i \) and \( w_{i,1} = Cte \) (resp. \( w_{i,0} = Cte \) and \( w_{i,1} = w_i \)) where \( Cte \) is a big value (we can put \( Cte = \sum_{t_i \in T' \setminus T''} \max \{w_{i,0}, w_{i,1}\} + \sum_{t_i \in T''} w_i + \sum_{(t_j,t_k) \in E} c_{t,j,k} \)). With this transformation, the rounding \( \theta_1 \) or \( \theta_2 \) will automatically assign the task \( t_i \) to the CPU (resp. GPU).

5. Numerical results

In this section, we compare the performance of LS (List Scheduling) algorithm to HEFT using benchmarks generated by Turbine [11]. In what follows, we describe the generation of benchmarks, then we discuss the efficiency of models (\( P1' \) and \( P2' \)), the behavior of the algorithm 2 starting from mapping \( \theta_1 \) and \( \theta_2 \) using the different lists given in section 4.

5.1. Benchmark

TABLE 1: Description of applications and platforms.

<table>
<thead>
<tr>
<th>Instances</th>
<th>Number of tasks</th>
<th>Platform 1</th>
<th>Platform 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \ell )</td>
<td>( k )</td>
<td>( \ell )</td>
</tr>
<tr>
<td>test_1</td>
<td>10</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>test_2</td>
<td>30</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>test_3</td>
<td>60</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>test_4</td>
<td>100</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>test_5</td>
<td>200</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>test_6</td>
<td>400</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>test_7</td>
<td>500</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>test_8</td>
<td>600</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>test_9</td>
<td>800</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>test_10</td>
<td>1000</td>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>

The benchmark is composed of ten parallel DAG applications. We denote by \( test_i \) instance number \( i \), we generate 10 different applications for each \( test_i \) with \( i = 1 \ldots 10 \). The execution times of the tasks are generated randomly over an interval \([w_{min}, w_{max}]\), \( w_{min} \) has been fixed at 5 and \( w_{max} \) at 30. The degree of the tasks are generated randomly over an interval \([d_{min}, d_{max}]\), \( d_{min} \) has been fixed at 1 and \( d_{max} \) at 10.

Furthermore, communication rate for each arc was generated on an interval \([c_{min}, c_{max}]\), we set \( c_{min} \) to 35 and \( c_{max} \) to 50. Table 1 presents the size of each instance generated as well as the number of CPUs and GPUs used to execute each instance. For Platform 1, we add more CPU and GPU by increasing the size of applications. For Platform 2, we only use one CPU and one GPU.

5.2. Environment and algorithms

To study the performance of our method, we compared the ratio between makespan value obtained by LS algorithm with HEFT algorithm, the optimal solution obtained by \( \text{CPLEX} \) and the lower bound \( C'_{max} \) obtained by \( (P1') \) or \( (P2') \).

Table 2 (resp. 3) shows the results of tests of LS algorithm on 10 instances for each application size given in column Inst using three lists (\( LST, LFP, LLP \)) with the rounding \( \theta_1 \) (resp. \( \theta_2 \)) in platform 1. For the rounding \( \theta_2 \), we set \( v = 10 \). The next three columns concern the result of LS algorithm using \( LST \), where the column GAP gives the average ratio between makespan obtained by LS algorithm and \( C'_{max} \), \( \text{GAP} = \frac{\text{LS makespan} - C'_{max}}{C'_{max}} \times 100 \). Column Best presents the number of instances where LS algorithm provides better or the same solution obtained by using list \( LST \) instead of \( LFT \) or \( LLP \). Column Opt presents the number of instances where LS provides optimal solution using \( LST \). We take the number of solutions equal to the optimal solution provided by \( \text{CPLEX} \) or to the value of the lower bound \( C'_{max} \) obtained by \( (P1') \) or \( (P2') \) if we have not optimal solution. Thus, we do the same thing for the lists \( LFT \) and \( LLP \). Column Best LS with \( \theta_1 \) (resp. Best LS with \( \theta_2 \)) presents the number of instances where LS algorithm provides better or the same solution obtained by using the rounding \( \theta_1 \) (resp. \( \theta_2 \)) and the best list of three lists (\( LST, LFT, LLP \)) compared to the solution obtained by HEFT and LS algorithm using \( \theta_2 \) (resp. \( \theta_1 \)) and all lists (\( LST, LFT, LLP \)). Finally, column Time (\( P1' \)) (resp. Time (\( P1' \))) gives the average time that was needed for LS algorithm to provide a solution using the model \( (P1') \) (resp. \( (P2') \)) for the first phase.

Table 4 shows the results of tests of HEFT algorithm and running time of \( \text{CPLEX} \) in the same platform. Column Time \( \text{CPLEX} \) presents average time that was needed to \( \text{CPLEX} \) to provide the optimal solution using \( (P) \). We only have the result for the first two instances due to the large running time (> 14 h). The next four columns concern the HEFT algorithm, where the column GAP gives the average ratio between makespan obtained by HEFT algorithm and \( C'_{max} \), \( \text{GAP} = \frac{\text{HEFT makespan} - C'_{max}}{C'_{max}} \times 100 \). Column Opt presents the number of instances where HEFT provides optimal solution. We take the number of solutions equal to the optimal solution provided by \( \text{CPLEX} \) or to the value of the lower bound \( C'_{max} \) obtained by \( (P1') \) or \( (P2') \).
Column Best HEFT presents the number of instances where HEFT algorithm provides better or the same solution obtained by using LS algorithm using the rounding \(\theta_1\) or \(\theta_2\) for the three lists (LST, LFT, LLP). Finally, column Time HEFT presents the average time that was needed for HEFT to provide a solution. A line Average is added at the end of each table which represents the average of the values each column. Table 5, 6 and 7 present the same results obtained for platform 2.

5.3. Results analysis

5.3.1. Platform 1. Obtaining the optimal solution using CPLEX is very expensive in term of running time. For example, instance test_3 with 60 tasks and 4 GPU and 4 CPU, CPLEX cannot provide the optimal solution after 14h. Thus, we compare our method to the optimal solution for only the first two instances. From Table 2 and 3, we can notice that list LLP is better than LFT and LST. Comparing to table 4 results, LS algorithm provides better solution than HEFT using rounding \(\theta_1\) or \(\theta_2\) with list LFT or LLP. Furthermore, LS algorithm using rounding \(\theta_2\) and list LLP is the most efficient method, with 78% of best solutions and a ratio of 7.33% comparing to the lower bound.

<table>
<thead>
<tr>
<th>Inst</th>
<th>LST Gap</th>
<th>LFT Gap</th>
<th>LLP Gap</th>
<th>Best LS with (\theta_1)</th>
<th>Time P1'</th>
<th>Time P2'</th>
</tr>
</thead>
<tbody>
<tr>
<td>test_1</td>
<td>0%</td>
<td>10</td>
<td>10</td>
<td>0%</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>test_2</td>
<td>4.51%</td>
<td>3</td>
<td>3</td>
<td>3.31%</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>test_3</td>
<td>17.61%</td>
<td>0</td>
<td>17.98%</td>
<td>0%</td>
<td>11.25%</td>
<td>8</td>
</tr>
<tr>
<td>test_4</td>
<td>13.39%</td>
<td>1</td>
<td>13.43%</td>
<td>1%</td>
<td>7.38%</td>
<td>7</td>
</tr>
<tr>
<td>test_5</td>
<td>27.69%</td>
<td>2</td>
<td>26.06%</td>
<td>2%</td>
<td>15.93%</td>
<td>7</td>
</tr>
<tr>
<td>test_6</td>
<td>25.93%</td>
<td>3</td>
<td>25.36%</td>
<td>3%</td>
<td>12.82%</td>
<td>4</td>
</tr>
<tr>
<td>test_7</td>
<td>28.23%</td>
<td>1</td>
<td>26.6%</td>
<td>2%</td>
<td>14.32%</td>
<td>6</td>
</tr>
<tr>
<td>test_8</td>
<td>22.79%</td>
<td>0</td>
<td>22.39%</td>
<td>1%</td>
<td>11.51%</td>
<td>9</td>
</tr>
<tr>
<td>test_9</td>
<td>14.87%</td>
<td>0</td>
<td>14.07%</td>
<td>0%</td>
<td>2.204%</td>
<td>10</td>
</tr>
<tr>
<td>test_10</td>
<td>20.55%</td>
<td>0</td>
<td>18.9%</td>
<td>1%</td>
<td>5.32%</td>
<td>9</td>
</tr>
<tr>
<td>Average</td>
<td>17.35%</td>
<td>20%</td>
<td>16.81%</td>
<td>22%</td>
<td>8.30%</td>
<td>78%</td>
</tr>
</tbody>
</table>

TABLE 2: LS algorithm results using three lists (LST, LFP, LLP) with rounding \(\theta_1\) in platform 1.

<table>
<thead>
<tr>
<th>Inst</th>
<th>LST Gap</th>
<th>LFT Gap</th>
<th>LLP Gap</th>
<th>Best LS with (\theta_2)</th>
<th>Time P1'</th>
<th>Time P2'</th>
</tr>
</thead>
<tbody>
<tr>
<td>test_1</td>
<td>0%</td>
<td>10</td>
<td>10</td>
<td>0%</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>test_2</td>
<td>2.92%</td>
<td>5</td>
<td>5</td>
<td>1.24%</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>test_3</td>
<td>17.77%</td>
<td>0</td>
<td>14.16%</td>
<td>0%</td>
<td>10.31%</td>
<td>8</td>
</tr>
<tr>
<td>test_4</td>
<td>15.64%</td>
<td>2</td>
<td>11.32%</td>
<td>2%</td>
<td>7.76%</td>
<td>5</td>
</tr>
<tr>
<td>test_5</td>
<td>36.33%</td>
<td>2</td>
<td>25.29%</td>
<td>4%</td>
<td>21.08%</td>
<td>5</td>
</tr>
<tr>
<td>test_6</td>
<td>38.07%</td>
<td>0</td>
<td>20.59%</td>
<td>0%</td>
<td>9.28%</td>
<td>10</td>
</tr>
<tr>
<td>test_7</td>
<td>33.61%</td>
<td>0</td>
<td>21.66%</td>
<td>1%</td>
<td>11.10%</td>
<td>8</td>
</tr>
<tr>
<td>test_8</td>
<td>36.51%</td>
<td>0</td>
<td>18.19%</td>
<td>0%</td>
<td>6.96%</td>
<td>10</td>
</tr>
<tr>
<td>test_9</td>
<td>40.07%</td>
<td>0</td>
<td>13.71%</td>
<td>0%</td>
<td>2.01%</td>
<td>10</td>
</tr>
<tr>
<td>test_10</td>
<td>39.33%</td>
<td>0</td>
<td>18.63%</td>
<td>0%</td>
<td>4.77%</td>
<td>10</td>
</tr>
<tr>
<td>Average</td>
<td>26.02%</td>
<td>19%</td>
<td>14.47%</td>
<td>22%</td>
<td>7.33%</td>
<td>85%</td>
</tr>
</tbody>
</table>

TABLE 3: LS algorithm results using three lists (LST, LFP, LLP) with rounding \(\theta_2\) in platform 1.

There might be some typos or ambiguous data in the document, but the general structure and content are clear. The tables provide a comprehensive comparison of different algorithms and platforms, with detailed results and analysis.
5.3.2. Platform 2. For this platform, we use only one CPU and one GPU. CPLEX is more efficient than on the platform 1, but running time is still large. For instance test_3 with 60 tasks, CPLEX cannot provide the optimal solution after 6h. Thus, we compare our method to the optimal solution for only the first two instances. From Table 5 and 6, we can notice that list LLP is better than LFT and LST. Comparing to table 7 results, LS algorithm provides better solution than HEFT using rounding $\theta_1$ or $\theta_2$ with list LFT or LLP. Furthermore, LS algorithm using rounding $\theta_2$ and list LLP is the most efficient method, with 76% of best solutions and a ratio of 9.86% comparing to the lower bound. For the running time, HEFT algorithm also needs less time than LS algorithm to provide a solution. Finally, unlike the platform 1, we notice that the model $(P1')$ is more efficient than $(P2')$ in running time using rounding $\theta_1$ or $\theta_2$, where the LS algorithm gives a solution in less than 1 second for instances of 1000 tasks for the two models.

<table>
<thead>
<tr>
<th>Instances</th>
<th>Time CPLEX</th>
<th>HEFT Gap</th>
<th>Opt</th>
<th>Best HEFT</th>
<th>Time HEFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>test_1</td>
<td>0.39s 20.58%</td>
<td>4</td>
<td>6</td>
<td>0.01s</td>
<td></td>
</tr>
<tr>
<td>test_2</td>
<td>1h40 44.31%</td>
<td>3</td>
<td>4</td>
<td>0.01s</td>
<td></td>
</tr>
<tr>
<td>test_3</td>
<td>0.77s 29.96%</td>
<td>3</td>
<td>0.01s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>test_4</td>
<td>17.26%</td>
<td>1</td>
<td>0.02s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>test_5</td>
<td>12.17%</td>
<td>0</td>
<td>0.03s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>test_6</td>
<td>11.26%</td>
<td>0</td>
<td>0.05s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>test_7</td>
<td>12.15%</td>
<td>0</td>
<td>0.09s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>test_8</td>
<td>11.86%</td>
<td>2</td>
<td>0.11s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>test_9</td>
<td>11.21%</td>
<td>0</td>
<td>0.14s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>test_10</td>
<td>11.90%</td>
<td>0</td>
<td>0.22s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>&gt;50m 18.26%</td>
<td>/</td>
<td>16%</td>
<td>0.06s</td>
<td></td>
</tr>
</tbody>
</table>

TABLE 5: LS algorithm results using three lists (LST, LFT, LLP) with rounding $\theta_1$ in platform 2.

<table>
<thead>
<tr>
<th>Instances</th>
<th>Time CPLEX</th>
<th>HEFT Gap</th>
<th>Opt</th>
<th>Best HEFT</th>
<th>Time HEFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>test_1</td>
<td>18.57% 7 5</td>
<td>18.57% 7 5</td>
<td>17.61% 8 5</td>
<td>8</td>
<td>0.04s</td>
</tr>
<tr>
<td>test_2</td>
<td>52.72% 1 1</td>
<td>49.55% 1 1</td>
<td>41.05% 6 3</td>
<td>5</td>
<td>0.30s</td>
</tr>
<tr>
<td>test_3</td>
<td>44.14% 0 1</td>
<td>41.18% 0 1</td>
<td>30.69% 6 1</td>
<td>5</td>
<td>0.35s</td>
</tr>
<tr>
<td>test_4</td>
<td>25.00% 0 1</td>
<td>24.02% 0 1</td>
<td>9.80% 7 1</td>
<td>6</td>
<td>0.05s</td>
</tr>
<tr>
<td>test_5</td>
<td>5.94% 2 1</td>
<td>5.93% 3 1</td>
<td>3.3% 10 1</td>
<td>10</td>
<td>0.07s</td>
</tr>
<tr>
<td>test_6</td>
<td>0.31% 8 1</td>
<td>0.31% 8 1</td>
<td>0.31% 10 1</td>
<td>4</td>
<td>0.40s</td>
</tr>
<tr>
<td>test_7</td>
<td>1.59% 8 1</td>
<td>1.54% 8 1</td>
<td>0.34% 10 1</td>
<td>4</td>
<td>0.27s</td>
</tr>
<tr>
<td>test_8</td>
<td>0.19% 10 1</td>
<td>0.19% 10 1</td>
<td>0.19% 10 1</td>
<td>8</td>
<td>0.38s</td>
</tr>
<tr>
<td>test_9</td>
<td>0.61% 9 1</td>
<td>0.39% 9 1</td>
<td>0.05% 10 1</td>
<td>5</td>
<td>0.64s</td>
</tr>
<tr>
<td>test_10</td>
<td>0.16% 9 1</td>
<td>0.16% 10 1</td>
<td>0.16% 10 1</td>
<td>5</td>
<td>0.88s</td>
</tr>
<tr>
<td>Average</td>
<td>14.92% 54%</td>
<td>/</td>
<td>14.18% 56%</td>
<td>/</td>
<td>10.05% 87%</td>
</tr>
</tbody>
</table>

TABLE 6: LS algorithm results using three lists (LST, LFT, LLP) with rounding $\theta_2$ in platform 2.
6. Conclusion

This paper presents an efficient approximation algorithm to solve the task scheduling problem on hybrid platform with communication delays. We have studied the case of scheduling applications presented by DAG (Directed Acyclic Graph), the objective is to minimize the total execution time (makespan). The main contribution of this work is a 6-approximation algorithm (LS) with two phases: mapping then assignment. Two models and two rounding strategies have been proposed for the mapping. In the second phase, a list scheduling algorithm has been proposed to generate a feasible schedule using several lists. LS algorithm guarantees a ratio of 6 compared to the optimal solution using the first strategy of rounding $\theta_1$. Tests on large instances close to reality demonstrated the efficiency of our method and shows the limits of solving the problem with a solver such as CPLEX.

As part of the future, we will try to study the tight of LS algorithm using rounding $\theta_2$ which provide interesting solutions. Then, we will focus on solving the problem with energy constraint due to the significant consumption of these platforms. An extension to more general heterogeneous platforms with more than two types of processor is also planned.

References