Approximation algorithm for scheduling applications on hybrid multi-core machines with communications delays

Massinissa Ait Aba, Lilia Zaourar, Alix Munier-Kordon

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 nodes. 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 guaranties 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 a 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, $m = \ell + k$. An application A of n tasks is represented by a Directed Acyclic Graph (DAG) oriented $G(V, E)$, each vertex represents a task t_i. Each arc $e = \{t_i, t_j\}$ represents a precedence constraint between two tasks t_i and t_j. We associate it with the value $ct_{i,j}$ which represents the communication delay between t_i and t_j if they are executed on two different resource types. The exact formula to evaluate $ct_{i,j}$ which takes into consideration latencies and available bandwidth between processors is provided in [28]. We denote by $\Gamma^-(i)$ (resp. $\Gamma^+(i)$) the sets of the predecessors (resp. successors) of task t_i. Figure 1 presents an example of our application.

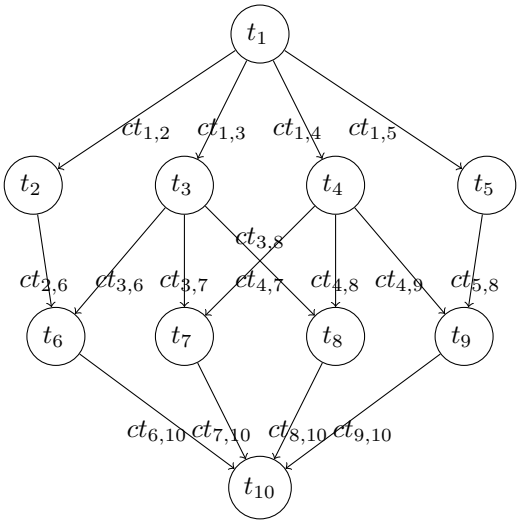

Figure 1: DAG application.

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 A (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 $start_i$ for $i = \overline{1..n}$ and $j = \overline{1..m}$, where $x_{i,j} = 1$ if t_i is executed on the CPU j for $j \in [1, \ell]$ (resp. GPU j for $j \in [\ell+1, m]$), 0 otherwise. $start_i$ 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_{i1} that precedes task t_{i2}, and the communication cost $ct_{i1,i2}$ 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}.

$$(P)\begin{cases} \sum_{j=1}^{m} x_{i,j} = 1, \forall i = \overline{1..n} & (1)\\ start_{i_1} + x_{i_1,j_1} w_{i_1,\lfloor \frac{j_1}{\ell+1} \rfloor} + x_{i_1,j_1} x_{i_2,j_2} ct_{i_1,i_2} \leqslant start_{i_2} & (2)\\ \forall\ \ i_1 \longrightarrow i_2,\ \ \forall j_1 = \overline{1..m},\ \ \forall j_2 = \overline{1..m},\ \ j_1 \neq j_2 & \\ start_{i_1} + x_{i_1,j} w_{i_1,\lfloor \frac{j}{\ell+1} \rfloor} \leqslant start_{i_2} + B \times (1 - x_{i_1 j} \times x_{i_2 j}) & \\ \vee & (3)\\ start_{i_2} + x_{i_2,j} w_{i_2,\lfloor \frac{j}{\ell+1} \rfloor} \leqslant start_{i_1} + B \times (1 - x_{i_1 j} \times x_{i_2 j}) & \\ \forall i_1, i_2 = \overline{1..n} \quad i_1 \neq i_2,\ \forall j = \overline{1..m}, B = Cte & \\ start_i + \sum_{j=1}^{m} x_{i,j} w_{i,\lfloor \frac{j}{\ell+1} \rfloor} \leqslant C_{max}, \forall i, \Gamma^+(i) = \emptyset & (4)\\ Z(min) = C_{max} & \end{cases}$$

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 which is equal to 1 if the task t_i is assigned to a CPU and 0 otherwise. Let the 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.

$$(P1)\begin{cases} C_i + x_j w_{j,0} + (1 - x_j) w_{j,1} + \zeta_{i,j} \leqslant C_j & (1)\\ \zeta_{i,j} = (1 - |y_{i,j} - z_{i,j}|) ct_{i,j}, \forall (t_i, t_j) \in E & \\ z_{i,j} \leqslant x_i, \forall (t_i, t_j) \in E & (2)\\ z_{i,j} \leqslant x_j, \forall (t_i, t_j) \in E & (3)\\ y_{i,j} \leqslant 1 - x_i, \forall (t_i, t_j) \in E & (4)\\ y_{i,j} \leqslant 1 - x_j, \forall (t_i, t_j) \in E & (5)\\ x_i w_{i,0} + (1 - x_i) w_{i,1} \leqslant C_i, \forall i = \overline{1..n}, \Gamma^-(i) = \emptyset & (6)\\ 0 \leqslant C_i \leqslant C_{max}, \forall i = \overline{1..n}, \Gamma^+(i) = \emptyset & (7)\\ \sum_{i=1}^{n} x_i w_{i,0} \leqslant \ell C_{max} & (8)\\ \sum_{i=1}^{n} (1 - x_i) w_{i,1} \leqslant k C_{max} & (9)\\ x_i, y_{i,j}, z_{i,j} \in \{0, 1\},\ \forall i = \overline{1..n}, j = \overline{1..n} & (10)\\ Z(min) = C_{max} & \end{cases}$$

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 two different processors, we obtain two cases: either $x_i = 1$ and $x_j = 0$ or $x_i = 0$ and $x_j = 1$. In the two cases, we obtain $y_{i,j} = 0$ and $z_{i,j} = 0$, implies that $1 - |y_{i,j} - z_{i,j}| = 1$ because of 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: $x_i = 0$ and $x_j = 0$: 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: $x_i = 1$ and $x_j = 1$: 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 $(P2)$ without 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:

1) $|y_{i,j} - z_{i,j}| = 0$: $2a_{i,j} \geqslant b_{i,j}$ and $2(1-a_{i,j}) \geqslant b_{i,j}$, then $b_{i,j} = 0$ for $a_{i,j} \in \{0,1\}$.
2) $|y_{i,j} - z_{i,j}| = 1$: if $(y_{i,j} - z_{i,j}) = 1$, $1+2a_{i,j} \geqslant b_{i,j}$ and $-1+2(1-a_{i,j}) \geqslant b_{i,j}$, and since $b_{i,j} \in \{0.1\}$, $b_{i,j} = 1$ for $a_{i,j} = 0$. If $(z_{i,j} - y_{i,j}) = 1$, $-1 + 2a_{i,j} \geqslant b_{i,j}$ and $1 + 2(1 - a_{i,j}) \geqslant b_{i,j}$, and since $b_{i,j} \in \{0.1\}$, $b_{i,j} = 1$ for $a_{i,j} = 1$.

$$
(P2)\begin{cases}
C_i + x_j w_{j,0} + (1-x_j)w_{j,1} + \zeta_{i,j} \leqslant C_j & (1)\\
\zeta_{i,j} = (1-b_{i,j})ct_{i,j}, \forall (t_i,t_j) \in E & \\
z_{i,j} \leqslant x_i, \forall (t_i,t_j) \in E & (2)\\
z_{i,j} \leqslant x_j, \forall (t_i,t_j) \in E & (3)\\
y_{i,j} \leqslant 1 - x_i, \forall (t_i,t_j) \in E & (4)\\
y_{i,j} \leqslant 1 - x_j, \forall (t_i,t_j) \in E & (5)\\
(z_{i,j} - y_{i,j}) + 2(1-a_{i,j}) \geqslant b_{i,j}, \forall (t_i,t_j) \in E & (5.1)\\
(y_{i,j} - z_{i,j}) + 2a_{i,j} \geqslant b_{i,j}, \forall (t_i,t_j) \in E & (5.2)\\
x_i w_{i,0} + (1-x_i)w_{i,1} \leqslant C_i, \forall i = \overline{1..n}, \Gamma^-(i) = \emptyset & (6)\\
0 \leqslant C_i \leqslant C_{max}, \forall i = \overline{1..n}, \Gamma^+(i) = \emptyset & (7)\\
\sum_{i=1}^{n} x_i w_{i,0} \leqslant \ell C_{max} & (8)\\
\sum_{i=1}^{n} (1-x_i) w_{i,1} \leqslant k C_{max} & (9)\\
x_i, y_{i,j}, z_{i,j}, a_{i,j}, b_{i,j} \in \{0,1\},\ \forall i = \overline{1..n}, j = \overline{1..n} & (10)\\
Z(min) = C_{max} &
\end{cases}
$$

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 integrity variables x_i, $y_{i,j}$ and $z_{i,j}$. We also obtain the model $(P2^{'})$ by relaxing the integrity 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 $j = \overline{1..n}$. 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 $i \in \overline{1..n}$, the solution obtained is feasible and optimal for $(P1)$ and $(P2)$, otherwise the fractional values are rounded. We denote by x^r_i the rounded value of the fractional value of the assignment variable of task t_i in the optimal solution of $(P1^{'})$ or $(P2^{'})$. We set $x^r_i = 0$ if $x^{'}_i < \frac{1}{2}$, $x^r_i = 1$ otherwise.
Let θ_1 be the mapping obtained by this rounding. Each task t_i is mapped in either CPU or GPU. Thus, $\theta_1(t_i) \longrightarrow \{CPU, GPU\}$.

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

$$x^r_i \leqslant 2x^{'}_i$$
$$(1 - x^r_i) \leqslant 2(1 - x^{'}_i).$$

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

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

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

Proof: By constraints (2) and (3) from $(P1^{'})$ or $(P2^{'})$, $z^{'}_{i,j} \leqslant \min\{x^{'}_i, x^{'}_j\}$. By constraints (4) and (5) from $(P1^{'})$ or $(P2^{'})$, $y^{'}_{i,j} \leqslant \min\{1 - x^{'}_i, 1 - x^{'}_j\}$. 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 β. $\beta = |y^{'}_{i,j} - z^{'}_{i,j}| \leqslant max\{y^{'}_{i,j}, z^{'}_{i,j}\} \leqslant max\{\min\{1 - x^{'}_i, 1 - x^{'}_j\}, \min\{x^{'}_i, x^{'}_j\}\}$. Then, if $\min\{1 - x^{'}_i, 1 - x^{'}_j\} \geqslant \min\{x^{'}_i, x^{'}_j\}$, then we can put $y^{'}_{i,j} = \min\{1 - x^{'}_i, 1 - x^{'}_j\}$ and $z^{'}_{i,j} = 0$ to maximize β. Otherwise, we can put $z^{'}_{i,j} = \min\{x^{'}_i, x^{'}_j\}$ and $y^{'}_{i,j} = 0$. □

In the following, we suppose that the solution obtained by solving the 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. $\alpha_{i,j} \geqslant 0$. Indeed, if $y^{'}_{i,j} = \min\{1 - x^{'}_i, 1 - x^{'}_j\} \leqslant 1$ and $z^{'}_{i,j} = 0$, then $\alpha_{i,j} = 1 - y^{'}_{i,j} \geqslant 0$. Furthermore, if $z^{'}_{i,j} = \min\{x^{'}_i, x^{'}_j\} \leqslant 1$ and $y^{'}_{i,j} = 0$, then $\alpha_{i,j} = 1 - z^{'}_{i,j} \geqslant 0$.

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

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

□

Let two tasks t_i and t_j, such that t_i precedes t_j. We denote by $Cost^r_{i,j}$ the value given by $Cost^r_{i,j} = 0$ if $x^r_i = x^r_j$, $Cost^r_{i,j} = ct_{i,j}$ otherwise.

Proposition 2. $Cost^r_{i,j} \leqslant 2\alpha_{i,j}ct_{i,j}$.

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

4.2. Phase 2: scheduling algorithm

We note by EST_i the Earliest Start Time of the task t_i according to the mapping θ_1. The following algorithm builds a feasible schedule. The mapping θ_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 θ_1, list L.
Result: feasible scheduling.
begin
  $S \longleftarrow \emptyset$
  while $S \neq T$ **do**
    $RD = \{t_i \in T, \Gamma^-(t_i) \subseteq S\}$;
    $EST = min\{EST_j, t_j \in RD\}$;
    Let $t_k \in RD$ the first task following the order of list L such that $EST_k = EST$;
    Execute t_k at EST according to θ_1;
    $S \longleftarrow S \cup \{t_k\}$;

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, LST (List by Start Time) and LFT (List by Finish Time). The LST (resp. LFT) list can be obtained by sorting the tasks in ascending order of their processing start time (resp. processing finish time) obtained by solving the model $(P1^{'})$ or $(P2^{'})$. Let $Start^{'}_i$ (resp. $C^{'}_i$) be the processing start time (resp. processing finish time) of the task t_i obtained by solving the model $(P1^{'})$ or $(P2^{'})$, $i = \overline{1..n}$. Thus, $LST = \{t_1, t_2, ..., t_n\}$, with $Start^{'}_1 \leqslant Start^{'}_2 \leqslant ... \leqslant Start^{'}_n$. $LFT = \{t_1, t_2, ..., t_n\}$, with $C^{'}_1 \leqslant C^{'}_2 \leqslant ... \leqslant C^{'}_n$.

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_i precedes t_j and $x_i \neq x_j$, 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 $PL_1 \geqslant PL_2 \geqslant ... \geqslant PL_n$.

4.3. Algorithm analysis

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

1) $L(P_f)$: length of the fractional critical path P_f in the optimal solution of the program $(P1^{'})$ or $(P2^{'})$.
2) $\frac{W^f_{CPU}}{\ell}$: The fractional weight of the tasks allocated to the CPU in the optimal solution of the program $(P1^{'})$ or $(P2^{'})$ divided by ℓ, with $W^f_{CPU} = \sum_{i=1}^{n} x^{'}_i w_{i,0}$.
3) $\frac{W^f_{GPU}}{k}$: The fractional weight of the tasks allocated to the GPU in the solution of the optimal program $(P1^{'})$ or $(P2^{'})$ divided by k, with $W^f_{GPU} = \sum_{i=1}^{n}(1 - x^{'}_i)w_{i,1}$.

Furthermore, the solution $\widehat{C}_{max}$ obtained by Algorithm 1 is bounded by $L(P_r)$, $\frac{W^r_{CPU}}{\ell}$, $\frac{W^r_{GPU}}{k}$, length of the critical path P_r and the works on CPUs divided by ℓ 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 $A_1 = \sum_{i=1}^{n} x^r_i w_{i,0}$ (resp. $A_2 = \sum_{i=1}^{n}(1 - x^r_i)w_{i,1}$) 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 $\frac{A_1}{\ell}$ (resp. $\frac{A_2}{k}$), then $I_1 \leqslant k\frac{A_1}{\ell}$ (resp. $I_2 \leqslant \ell\frac{A_2}{k}$). Let I_3 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.

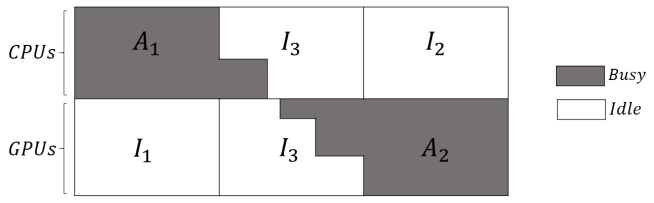

Figure 2: Occupation of processing elements.

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

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

Proposition 3.

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

Proof: By definition, $A_1 = \sum_{i=1}^{n} x_i^r w_{i,0}$. From Proposition 1, $x_i^r \leqslant 2x_i^{'}$. Then, $A_1 = \sum_{i=1}^{n} x_i^r w_{i,0} \leqslant \sum_{i=1}^{n} 2x_i^{'} w_{i,0} = 2W_{CPU}^{f} \leqslant 2\ell C_{max}^{'} \leqslant 2\ell C^{\star}_{max}$. Furthermore, by definition, $A_2 = \sum_{i=1}^{n}(1 - x_i^r)w_{i,1}$. From Proposition 1, $(1 - x_i^r) \leqslant 2(1 - x_i^{'})$. Then, $A_2 = \sum_{i=1}^{n}(1 - x_i^r)w_{i,1} \leqslant \sum_{i=1}^{n} 2(1 - x_i^{'})w_{i,1} = 2W_{GPU}^{f} \leqslant 2kC_{max}^{'} \leqslant 2kC^{\star}_{max}$. □

Corollary 1. If for each task t_i, x_i^r is integer, such that $x_i^r = x_i^{'}$ and $(1 - x_i^r) = (1 - x_i^{'})$ for $i = \overline{1..n}$, then the mapping θ_1 is optimal. Follow, $A_1 = \sum_{i=1}^{n} x_i^r w_{i,0} = \sum_{i=1}^{n} x_i^{'} w_{i,0} \leqslant \ell C^{\star}_{max}$ and $A_2 = \sum_{i=1}^{n}(1 - x_i^r)w_{i,1} = \sum_{i=1}^{n}(1 - x_i^{'})w_{i,1} \leqslant kC^{\star}_{max}$.

Corollary 2.

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

Proof:

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

Proposition 4. $I_3 \leqslant 2(\ell + k)C^{\star}_{max}$.

Proof: There exists a critical path γ in the final scheduling such that the sum of the instants where at least one CPU and one GPU are idle is less than $2L(P_f)$. Indeed, we assume that the tasks are stalled on the left. Let t_0 be the last task, such as during the execution of t_0, there is an idle CPU and an idle GPU. Let $Start_0$ be the processing start time of the task t_0. If there is an idle CPU and idle GPU before $Start_0$, then t_0 has a predecessor t_1 that ends before $Start_0$, the idle slots between $Start_1$ and $Start_0$ are covered either by the execution time of the task t_1 and eventually the communication cost between t_1 and his successor $t_0^{'}$ which can be t_0 or a task on the path from t_1 to t_0. If there is an idle CPU and idle GPU before $Start_1$, then t_1 has a predecessor t_2 that ends before $Start_1$ which can be obtained in the same precedent way. Let t_0, $t_0^{'}$, t_1, $t_1^{'}$,..., t_ℓ be the maximum sequence of tasks obtained. There is no more slots before $Start_\ell$ where at least one CPU and one GPU are idle. Let γ 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_r, the processing time of t_i in the final scheduling will be at most twice the fractional solution obtained by the program $(P1^{'})$ or $(P2^{'})$. For any two tasks t_i, t_j in P_r, from the Proposition 2, the communication cost $Cost_{i,j}^{r}$ between t_i and t_j in the final scheduling will increase by at most twice the fractional communication cost $\alpha_{i,j}ct_{i,j}$ obtained by the program $(P1^{'})$ or $(P2^{'})$. Then, $L(\gamma) \leqslant L(P_r) = \sum_{t_i \in P_r}(x_i^r w_{i,0} + (1 - x_i^r)w_{i,1}) + \sum_{(t_i,t_j)\in P_r} Cost_{i,j}^{r} \leqslant \sum_{t_i \in P_r}(2x_i^{'} w_{i,0} + 2(1 - x_i^{'})w_{i,1}) + \sum_{(t_i,t_j)\in P_r} 2\alpha_{i,j}ct_{i,j} \leqslant 2L(P_f)$. Finally, $I_3 \leqslant (\ell + k)L(P_r) \leqslant 2(\ell + k)L(P_f) \leqslant 2(\ell + k)C^{\star}_{max}$. □

Theorem 1. The ratio between the solution $\widehat{C}_{max}$ obtained by Algorithm 1 (LS) and the optimal scheduling solution $C^{\star}_{max}$ is given by $\frac{\widehat{C}_{max}}{C^{\star}_{max}} \leqslant 6$.

Proof: $\widehat{C}_{max} = \frac{A+I}{\ell+k} = \frac{A_1+A_2+I_1+I_2+I_3}{\ell+k}$. Then, from Proposition 3, 4 and Corollary 2, $\widehat{C}_{max} \leqslant \frac{(2\ell+2k+2k+2\ell+2(\ell+k))C^{\star}_{max}}{\ell+k} = 6C^{\star}_{max}$. Finally, $\frac{\widehat{C}_{max}}{C^{\star}_{max}} \leqslant 6$. □

Corollary 3. If the mapping θ_1 is optimal, then $\frac{\widehat{C}_{max}}{C^{\star}_{max}} \leqslant 5$.

Proof:

From Corollary 1, we know that $A_1 \leqslant \ell C^{\star}_{max}$ and $A_2 \leqslant kC^{\star}_{max}$. Then, $\widehat{C}_{max} = \frac{A_1+A_2+I_1+I_2+I_3}{\ell+k} \leqslant \frac{(\ell+k+2k+2\ell+2(\ell+k))C^{\star}_{max}}{\ell+k}$. Finally, $\frac{\widehat{C}_{max}}{C^{\star}_{max}} \leqslant 5$. □

4.4. Iterative mapping

In order to find a more efficient rounding than θ_1 described previously, we try to assign the tasks progressively. Let θ_2 be the rounding obtained by the following algorithm 2.

Algorithm 2: Rounding algorithm.

Data: model (P) $((P1^{'})$ or $(P2^{'}))$, $\upsilon \geqslant 2$.
Result: mapping θ_2.
begin
  for $i = 1$ *to* $\lfloor \frac{\upsilon}{2} \rfloor$ **do**
    Solve P;
    for $j = 1$ *to* n **do**
      if $x_j^{'} < (\frac{i}{\upsilon})$ **then**
        Set in $P : x_j^{'} = 0$
      if $x_j^{'} \geqslant 1 - (\frac{i}{\upsilon})$ **then**
        Set in $P : x_j^{'} = 1$
  for $l = 1$ *to* n **do**
    if $x_l^{'} < \frac{1}{2}$ **then**
      $x_l^r = 0$
    else
      $x_l^r = 1$

For a given integer $\upsilon \geqslant 2$, we solve the model $(P1^{'})$ or $(P2^{'})$ $\lfloor \frac{\upsilon}{2} \rfloor$ times adding new assignment constraint at each resolution. Contrary to θ_1, we try to assign the tasks progressively, starting by setting $x_j^{'}$ to 0 if $x_j^{'} < (\frac{1}{\upsilon})$ and $x_j^{'}$ to 1 if $x_j^{'} \geqslant 1 - (\frac{1}{\upsilon})$ for the first resolution of $(P1^{'})$ or $(P2^{'})$, and finishing by setting $x_j^{'}$ to 0 if $x_j^{'} < (\frac{\lfloor \frac{\upsilon}{2} \rfloor}{\upsilon})$ and $x_j^{'}$ to 1 if $x_j^{'} \geqslant 1 - (\frac{\lfloor \frac{\upsilon}{2} \rfloor}{\upsilon})$, where $\frac{\lfloor \frac{\upsilon}{2} \rfloor}{\upsilon} \leqslant \frac{1}{2}$ according to the Lemma 3.

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

Lemma 3. For an integer $v \geqslant 2$, $\frac{\lfloor \frac{v}{2} \rfloor}{v} \leqslant \frac{1}{2}$.

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, $v = 2\lambda + 1 : \frac{\lfloor \frac{v}{2} \rfloor}{v} = \frac{\lambda}{2\lambda+1} < \frac{1}{2}$. □

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 $Cte = \sum_{t_i \in T \setminus T'} max\{w_{i,0}, w_{i,1}\} + \sum_{t_i \in T'} w_i + \sum_{(t_j,t_k) \in E} ct_{j,k}$). With this transformation, the rounding θ_1 or θ_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 θ_1 and θ_2 using the different lists given in section 4.

5.1. Benchmark

TABLE 1: Description of applications and platforms.

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

The benchmark 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 $i = \overline{1..10}$. The execution times of the tasks are generated randomly over an interval $[w_{min}, w_{max}]$, w_{min} has been fixed at 5 and w_{max} at 30. The 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 $[ct_{min}, ct_{max}]$, we set ct_{min} to 35 and ct_{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 Platform1, we add more CPU and GPU by increasing the size of applications. For Platform2, 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 each makespan value obtained by LS algorithm with HEFT algorithm, the optimal solution obtained by *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 θ_1 (resp. θ_2) in platform 1. For the rounding θ_2, we set $\upsilon = 10$. 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 $C^{'}_{max}$, $GAP = \frac{\text{LS makespan} - C^{'}_{max}}{C^{'}_{max}} \times 100$. 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 *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 θ_1 (resp. Best LS with θ_2) presents the number of instances where LS algorithm provides better or the same solution obtained by using the rounding θ_1 (resp. θ_2) and the best list of three lists (LST,LFT,LLP) compared to the solution obtained by HEFT and LS algorithm using θ_2 (resp. θ_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 *CPLEX* in the same platform. Column Time *CPLEX* presents average time that was needed to *CPLEX* to provide the optimal solution using (P). We only have the result for the first two instances due to the large running time ($> 14h$). The next four columns concern the HEFT algorithm, where the column GAP gives the average ratio between makespan obtained by HEFT algorithm and $C^{'}_{max}$, $GAP = \frac{\text{HEFT makespan} - C^{'}_{max}}{C^{'}_{max}} \times 100$. 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 *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 θ_1 or θ_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 θ_1 or θ_2 with list LFT or LLP. Furthermore, LS algorithm using rounding θ_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 4: HEFT algorithm results and *CPLEX* running time for platform 1.

Instances	Time *CPLEX*	HEFT Gap	HEFT Opt	Best HEFT	Time HEFT
test_1	5m	11.08%	5	5	0.01s
test_2	8h	13.48%	4	4	0.01s
test_3	/	22.16%	/	2	0.02s
test_4	/	16.33%	/	3	0.02s
test_5	/	19.31%	/	4	0.04s
test_6	/	16.72%	/	0	0.09s
test_7	/	15.05%	/	1	0.15s
test_8	/	15.05%	/	0	0.20s
test_9	/	11.79%	/	0	0.33s
test_10	/	15.26%	/	0	0.64s
Average	>4h	15.62%	/	19%	0.15s

For the running time, HEFT algorithm needs less time than LS algorithm to provide a solution, where the first iteration of rounding algorithm (θ_2) takes the same time than θ_1 (remark 2). Finally, we notice that the model $(P2^{'})$ is more efficient than $(P1^{'})$ in running time using rounding θ_1 or θ_2, where LS algorithm gives a solution in less than 3 seconds for instances of 1000 tasks using model $(P2^{'})$, while it need more than 4 seconds using model $(P1^{'})$. Thus it may provides better results for much bigger instances.

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

TABLE 2: LS algorithm results using three lists (LST, LFP, LLP) with rounding θ_1 in platform 1.

Inst	LST Gap	LST Best	LST Opt	LFT Gap	LFT Best	LFT Opt	LLP Gap	LLP Best	LLP Opt	Best LS with θ_2	Time $P1^{'}$	Time $P2^{'}$
test_1	0%	10	10	0%	10	10	0%	10	10	10	0.07s	0.09s
test_2	2.92%	5	5	1.24%	5	5	0.09%	9	6	9	0.018s	0.16s
test_3	17.77%	0	/	14.16%	0	/	10.31%	8	/	5	0.17s	0.17s
test_4	15.64%	2	/	11.32%	2	/	7.76%	5	/	6	0.31s	0.34s
test_5	36.33%	2	/	25.29%	4	/	21.08%	5	/	4	0.85s	0.83s
test_6	38.07%	0	/	20.59%	0	/	9.28%	10	/	9	5.01s	2.95s
test_7	33.61%	0	/	21.66%	1	/	11.10%	8	/	8	6.71s	4.59s
test_8	36.51%	0	/	18.19%	0	/	6.96%	10	/	10	8.63s	5.28s
test_9	40.07%	0	/	13.71%	0	/	2.01%	10	/	7	11.55s	7.06s
test_10	39.33%	0	/	18.63%	0	/	4.77%	10	/	10	15.10s	8.13s
Average	26.02%	19%	/	14.47%	22%	/	7.33%	85%	/	78%	4.84s	2.96s

TABLE 3: LS algorithm results using three lists (LST, LFP, LLP) with rounding θ_2 in platform 1.

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 θ_1 or θ_2 with list LFT or LLP. Furthermore, LS algorithm using rounding θ_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 θ_1 or θ_2, where the LS algorithm gives a solution in less than 1 second for instances of 1000 tasks for the two models.

TABLE 7: HEFT algorithm results and *CPLEX* running time for platform 2.

Instances	**Time** ***CPLEX***	**HEFT**		**Best HEFT**	**Time HEFT**
		Gap	**Opt**		
test_1	0.39s	20.58%	4	6	0.01s
test_2	1h40	44.31%	3	4	0.01s
test_3	/	29.99%	/	3	0.01s
test_4	/	17.26%	/	1	0.02s
test_5	/	12.17%	/	0	0.03s
test_6	/	11.26%	/	0	0.05s
test_7	/	12.15%	/	0	0.09s
test_8	/	11.86%	/	2	0.11s
test_9	/	11.21%	/	0	0.14s
test_10	/	11.90%	/	0	0.22s
Average	>50m	18.26%	/	16%	0.06s

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

TABLE 5: LS algorithm results using three lists (LST, LFP, LLP) with rounding θ_1 in platform 2.

Inst	***LST***			***LFT***			***LLP***			**Best LS**	**Time**	
	Gap	**Best**	**Opt**	**Gap**	**Best**	**Opt**	**Gap**	**Best**	**Opt**	**with θ_2**	$P1^{'}$	$P2^{'}$
test_1	17.26%	8	6	17.26%	8	6	16.30%	9	6	9	0.07s	0.09s
test_2	55.68%	1	1	49.07%	1	1	43.03%	6	3	3	0.33s	0.35s
test_3	60.16%	0	/	39.38%	0	/	29.21%	6	/	5	0.40s	0.40s
test_4	41.96%	0	/	23.31%	0	/	9.11%	9	/	9	0.11s	0.14s
test_5	20.52%	0	/	6.22%	2	/	0.43%	10	/	7	0.17s	0.23s
test_6	11.41%	0	/	0.15%	8	/	0.15%	10	/	10	0.62s	0.82s
test_7	8.73%	0	/	1.68%	8	/	0.19%	10	/	8	0.52s	0.82s
test_8	8.08%	2	/	0.11%	10	/	0.11%	10	/	9	0.77s	1.13s
test_9	10.90%	0	/	0.44%	9	/	0.03%	10	/	8	1.25s	1.89s
test_10	8.67%	0	/	0.08%	10	/	0.08%	10	/	8	1.32s	1.91s
Average	24.33%	11%	/	9.96%	56%	/	9.86%	90%	/	76%	0.55s	0.77s

TABLE 6: LS algorithm results using three lists (LST, LFP, LLP) with rounding θ_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 θ_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 θ_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

[1] https://www.top500.org/lists/2017/11/.

[2] Ishfaq Ahmad and Yu-Kwong Kwok. On exploiting task duplication in parallel program scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 9(9):872–892, 1998.

[3] Massinissa Ait Aba, Lilia Zaourar, and Alix Munier. Approximation algorithm for scheduling a chain of tasks on heterogeneous systems. In *Euro-Par 2017: Parallel Processing Workshops*. Santiago de Compostela, Spain, September 2017.

[4] Shaikhah AlEbrahim and Imtiaz Ahmad. Task scheduling for heterogeneous computing systems. *The Journal of Supercomputing*, 73(6):2313–2338, 2017.

[5] Marcos Amaris, Giorgio Lucarelli, Clément Mommessin, and Denis Trystram. Generic algorithms for scheduling applications on hybrid multi-core machines. In *European Conference on Parallel Processing*, pages 220–231. Springer, 2017.

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

[7] Rashmi Bajaj and Dharma P Agrawal. Improving scheduling of tasks in a heterogeneous environment. *IEEE Transactions on Parallel and Distributed Systems*, 15(2):107–118, 2004.

[8] Olivier Beaumont, Lionel Eyraud-Dubois, and Yihong Gao. Influence of tasks duration variability on task-based runtime schedulers. 2018.

[9] Olivier Beaumont, Lionel Eyraud-Dubois, and Suraj Kumar. Approximation proofs of a fast and efficient list scheduling algorithm for task-based runtime systems on multicores and gpus. In *Parallel and Distributed Processing Symposium (IPDPS), 2017 IEEE International*, pages 768–777. IEEE, 2017.

[10] Luiz F Bittencourt, Rizos Sakellariou, and Edmundo RM Madeira. Dag scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm. In *Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on*, pages 27–34. IEEE, 2010.

[11] Bruno Bodin, Youen Lesparre, Jean-Marc Delosme, and Alix Munier-Kordon. Fast and efficient dataflow graph generation. In *Proceedings of the 17th International Workshop on Software and Compilers for Embedded Systems*, pages 40–49. ACM, 2014.

[12] Cristina Boeres, Vinod EF Rebello, et al. A cluster-based strategy for scheduling task on heterogeneous processors. In *Computer Architecture and High Performance Computing, 2004. SBAC-PAD 2004. 16th Symposium on*, pages 214–221. IEEE, 2004.

[13] Louis-Claude Canon, Loris Marchal, and Frédéric Vivien. Low-cost approximation algorithms for scheduling independent tasks on hybrid platforms. In *European Conference on Parallel Processing*, pages 232–244. Springer, 2017.

[14] Philippe Chretienne. Task scheduling with interprocessor communication delays. *European Journal of Operational Research*, 57(3):348–354, 1992.

[15] Fabián A Chudak and David B Shmoys. Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds. *Journal of Algorithms*, 30(2):323–343, 1999.

[16] Michael R Garey and Ronald L. Graham. Bounds for multiprocessor scheduling with resource constraints. *SIAM Journal on Computing*, 4(2):187–200, 1975.

[17] IBM. Ibm ilog cplex v12.5 user's manual for cplex, http://www.ibm.com. 2013.

[18] E Ilavarasan, P Thambidurai, and R Mahilmannan. High performance task scheduling algorithm for heterogeneous computing system. In *ICA3PP*, volume 2005, pages 193–203. Springer, 2005.

[19] Muhammad Kafil and Ishfaq Ahmad. Optimal task assignment in heterogeneous distributed computing systems. *IEEE concurrency*, 6(3):42–50, 1998.

[20] Safia Kedad-Sidhoum, Florence Monna, and Denis Trystram. Scheduling tasks with precedence constraints on hybrid multi-core machines. In *Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International*, pages 27–33. IEEE, 2015.

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

[22] Minhaj Ahmad Khan. Task scheduling for heterogeneous systems using an incremental approach. *The Journal of Supercomputing*, 73(5):1905–1928, 2017.

[23] Sunita Kushwaha and Sanjay Kumar. An investigation of list heuristic scheduling algorithms for multiprocessor system. *IUP Journal of Computer Sciences*, 11(2):29, 2017.

[24] Samantha Ranaweera and Dharma P Agrawal. A task duplication based scheduling algorithm for heterogeneous systems. In *Parallel and Distributed Processing Symposium, 2000. IPDPS 2000. Proceedings. 14th International*, pages 445–450. IEEE, 2000.

[25] Linshan Shen and Tae-Young Choe. Posterior task scheduling algorithms for heterogeneous computing systems. In *International Conference on High Performance Computing for Computational Science*, pages 172–183. Springer, 2006.

[26] Haluk Topcuoglu, Salim Hariri, and Min-you Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE transactions on parallel and distributed systems*, 13(3):260–274, 2002.

[27] Shuli Wang, Kenli Li, Jing Mei, Guoqing Xiao, and Keqin Li. A reliability-aware task scheduling algorithm based on replication on heterogeneous computing systems. *Journal of Grid Computing*, 15(1):23–39, 2017.

[28] Lilia Zaourar, Massinissa Ait Aba, David Briand, and Jean-Marc Philippe. Modeling of applications and hardware to explore task mapping and scheduling strategies on a heterogeneous micro-server system. In *Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2017 IEEE International*, pages 65–76. IEEE, 2017.