Generic algorithms for scheduling applications on hybrid multi-core machines

Marcos Amaris, Giorgio Lucarelli, Clément Mommessin, Denis Trystram

Generic algorithms for scheduling applications on hybrid multi-core machines

Marcos Amaris*†, Giorgio Lucarelli*‡, Clément Mommessin*, and Denis Trystram*
* *LIG, UMR 5217, Grenoble Institute of Technology, Grenoble, France*
{*clement.mommessin, denis.trystram*}*@imag.fr*
† *Institute of Mathematics and Statistics, University of São Paulo, São Paulo, Brazil*
amaris@ime.usp.br
‡ *INRIA Grenoble Rhône-Alpes, Grenoble, France*
giorgio.lucarelli@inria.fr

***Abstract*—We study the problem of executing an application represented by a precedence task graph on a multi-core machine composed of standard computing cores and accelerators. Contrary to most existing approaches, we distinguish the allocation and the scheduling phases and we mainly focus on the allocation part of the problem: choose the more appropriate type of computing unit for each task. We address both off-line and on-line settings. In the first case, we establish strong lower bounds on the worst-case performance of a known approach based on Linear Programming for solving the allocation problem. Then, we refine the scheduling phase and we replace the greedy list scheduling policy used in this approach by a better ordering of the tasks. Although this modification leads to the same approximability guarantees, it performs much better in practice. We also extend this algorithm to more types of heterogeneous cores, achieving an approximation ratio which depends on the number of different types. In the on-line case, we assume that the tasks arrive in any, not known in advance, order which respects the precedence relations and the scheduler has to take irrevocable decisions about their allocation and execution. In this setting, we propose the first scheduling algorithm with precedences based on adequate rules for selecting the type of processor where to allocate the tasks. This algorithm achieves a constant factor approximation guarantee if the ratio of the number of CPUs over the number of GPUs is bounded. Finally, all the previous algorithms have been experimented on a large number of simulations built upon actual libraries. These simulations assess the good practical behavior of the algorithms with respect to the state-of-the-art solutions whenever these exist or baseline algorithms.**

I. Introduction

The parallel and distributed platforms available today become more and more *heterogeneous*. Such heterogeneous architectures, composed of several kind of computing units, have a growing impact on performances in High Performance Computing. Hardware accelerators, such as General Purpose Graphical Processing Units (denoted in short by GPU) [1], are often used in conjunction with multiple Central Processing Units (CPUs) on the same chip sharing the same common memory. As an instance of this, the number of platforms of the TOP500 equipped with accelerators is significantly increased during the last years [2]. In the future it is expected that the nodes of such platforms will be even more diverse than today: they will be composed of fast computing nodes, hybrid computing nodes mixing general purpose units with accelerators, I/O nodes, nodes specialized in data analytics, etc. The interconnect of a huge number of such nodes will also lead to more heterogeneity. Using heterogeneous platforms would lead to better performances through the use of more appropriate resources depending on the computations to perform, but it has a cost in terms of code development and more complex resource management.

In this work, we present efficient algorithms for scheduling an application represented by a precedence task graph on hybrid computing resources. We are interested in designing generic approaches for efficiently implementing parallel applications where the scheduling is not explicitly part of the application. This way, the code is portable and could be easily adapted to the next generation of machines.

A. Underlying architecture

We consider a hybrid multi-core parallel node composed of identical CPUs and identical GPUs. An application consists of tasks that are linked by precedence relations. Each task is characterized by two processing times depending on which type of processors it is assigned to. We assume that an exact estimation of both processing times of a task is available at the time when the task becomes known to the system. This assumption can be justified by several existing models to estimate the execution times of the tasks in multi-core nodes where applications are executed [3], [4]. In several applications we observe always an acceleration of the tasks if they are executed on a GPU compared to their execution on a CPU. However, we consider here the more general case where the relation between the two processing times can differ for different tasks. This work focuses on the analysis of the qualitative behavior induced by heterogeneity since it may be assumed that the computations dominate local shared memory costs. Thus, no memory assignment or overhead for data management are considered, nor communication times between the shared memory and the CPUs or between CPUs and GPUs.

As the application developers are mainly looking for performance, the objective of a scheduler is usually to minimize the completion time of the last finishing task (known as

makespan), which is one of the most commonly studied objectives [5]. In an heterogeneous context, minimizing the makespan of an application corresponds to minimize the maximum between the makespan of the tasks assigned on the CPUs and the makespan of the tasks assigned on the GPUs.

B. Definition and notations

Formally, we consider a parallel application which should be scheduled on a set of m identical CPUs and a set of k identical GPUs. Henceforth, we assume that $m \geq k$. The application is represented by a Directed Acyclic Graph $G = (V, E)$ whose nodes correspond to sequential tasks and arcs correspond to precedence relations among the tasks. Let $\mathscr{T}$ be the set of all tasks. The execution of a task needs a different amount of time if it is performed by a CPU or by a GPU. Specifically, let $\overline{p_j}$ (resp. $\underline{p_j}$) be the processing time of a task $T_j \in \mathscr{T}$ if it is executed on any CPU (resp. GPU). Given a schedule S, we denote by C_j the completion time of a task $T_j \in \mathscr{T}$ in S. In any feasible schedule, for each arc $(i, j) \in E$, the task T_j cannot be executed before the completion of T_i. We say that T_i is a *predecessor* of T_j and we denote by $\Gamma^-(T_j)$ the set of all predecessors of the task T_j. Similarly, we say that T_j is a *successor* of T_i and we denote by $\Gamma^+(T_i)$ the set of all successors of T_i. Moreover, we call *descendant* of a task T_j each task T_i for which there is a path from j to i in G.

The objective is to create a feasible non-preemptive schedule of minimum makespan. In other words, we seek a schedule that respects the precedence constraints among the tasks, does not interrupt their execution and minimizes the completion time of the last task, i.e., $C_{\max} = \max_{T_j \in \mathscr{T}}\{C_j\}$. Using a natural extension of the standard three-field notation of scheduling problems introduced by Graham, this problem can be denoted as $(CPU, GPU) \mid prec \mid C_{\max}$.

C. Contributions and outline

Our purpose in this paper is to study the problem with precedences on both off-line and on-line settings. The goal is to design algorithms through a solid theoretical analysis that can be practically implemented in actual systems. Contrarily to most existing approaches (see for example [6]), we propose to address the problem of executing an application on an hybrid machine by separately focusing on the following two phases:

- *allocation phase*: each task is assigned to a type of resources, either CPU or GPU.
- *scheduling phase*: each task is assigned to a specific pair of resource and time interval, taking into account the allocation decided in the previous phase as well as the precedence constraints.

The motivation for considering the two phases separately is due to the fact that there are strong lower bounds on the approximability of known single-phase algorithms. For example, the approximation ratio of the well-known Heterogeneous Earliest Finish Time (HEFT) algorithm [6] cannot be better than $O(\frac{m}{k^2})$ (see Section III), while it can be easily shown that greedy List Scheduling policies have arbitrarily large approximation ratio, even if we consider some specific order of tasks, like prioritizing the task of the largest acceleration, among other.

The two-phases approach has been used by Kedad-Sidhoum *et al.* [7] where a linear program (which we call Heterogeneous Linear Program or simply HLP) in conjunction with a rounding have been proposed for the allocation phase, while the greedy Earliest Starting Time (EST) policy has been applied to schedule the tasks. This algorithm, which we henceforth call HLP-EST, achieves an approximation ratio of 6. Surprisingly, in Section III, we show that the ratio of this algorithm is tight. In fact, our worst-case example does not depend on the scheduling policy applied in the second phase.

Based on this negative result, we propose to revisit both phases. In Section IV-A, we initially present a series of greedy rules which can be used to decide the allocation. Although these rules are of low complexity, a desired property in practice, they are only based on the relation between the processing times of a task and they neither consider the schedule created up to now nor look to the future precedence relations that define the critical path. For these reasons, they cannot guarantee any approximation ratio. However, a more enhanced combination of rules that takes into account the actual schedule can lead to an algorithm of worst case ratio $O(\sqrt{\frac{m}{k}})$, even in an on-line context where the tasks arrive in any, not known in advance, order that respects the precedence constraints, and the scheduler has to take irrevocable decisions for their execution at the time of their arrival. This is the first on-line upper-bound when precedence constraints are considered in the hybrid context. In Section IV-B, we propose to replace the EST policy of the HLP-EST algorithm by a specific order of the tasks which is based on both the allocation decisions taken in the first phase (linear program) and the critical path. This refined algorithm, denoted by HLP-OLS, preserves the approximation ratio of 6 (the proof is very similar and it is omitted) but it also has a very good practical performance. In Section IV-C we propose an extension of the HLP-EST algorithm and its analysis for the case where more than two types of identical processors are available. We show that this algorithm has a tight approximation ratio of $Q(Q+1)$, where $Q \geq 2$ is the number of resource types.

In Section V, we first describe the generation of the benchmark that we use in the experiments, consisting in five applications of dense linear algebra from Chameleon software [8] and a *fork-join* application generated using GGen [9]. The benchmark used is freely available in SWF format. Using this benchmark, an experimental evaluation of the proposed algorithms has been performed and they have

been compared with the HEFT and HLP-EST algorithms for the off-line case. In the on-line case, we used two baseline algorithms as a reference for the comparison. The experiments show that the new scheduling method based on HLP outperforms both HEFT and HLP-EST in most of the applications, while the proposed on-line algorithm has significantly better makespan than the baseline greedy algorithms used.

Before continuing with the main part of the paper, we present in Section II the works related to our setting and, finally, we conclude in Section VI.

II. Related works

Most papers of the huge existing literature about GPUs concern specific applications. There are only few papers dealing with generic scheduling in mixed CPU/GPU architectures, and very few of them consider precedence constraints.

From a theoretical perspective, the problem of scheduling on two types of resources is more complex than the problem of scheduling tasks on parallel identical machines, $P \mid prec \mid C_{\max}$, but it is easier than the problem on unrelated machines, $R \mid prec \mid C_{\max}$. Moreover, if all tasks are accelerated by the same factor in the GPU side, then $(CPU, GPU) \mid prec \mid C_{\max}$ coincides with the problem of scheduling on uniformly-related parallel machines, $Q \mid prec \mid C_{\max}$. In this sense, we can say that the former is more general than the latter one; however in our problem all tasks have only two different processing times, that makes it simpler. For $P \mid prec \mid C_{\max}$, Graham's List Scheduling algorithm [10] is a 2-approximation, while no algorithm can have a better approximation ratio [11]. Chudak and Shmoys [12] developed a polynomial-time $O(\log m)$-approximation algorithm for $Q \mid prec \mid C_{\max}$, while Chekuri and Bender [13] proposed a faster polynomial-time approximation algorithm with the same order of worst-case performance. For hybrid architectures, a 6-approximation algorithm has been proposed by Kedad-Sidhoum *et al.* [7]. In the case of independent tasks there is a $(\frac{4}{3} + \frac{1}{3k})$-approximation algorithm [14]. If the tasks arrive in an on-line order, a 4-competitive algorithm has been presented by Chen *et al.* [15] for hybrid architectures without precedence relations. A closely related problem, in which the architecture consists of $Q \geq 2$ different types of resources and each task can be executed only on some of them, has been also studied in the literature. This problem generalizes the dedicated processors case if each processor consists of several identical cores, while a $(Q+1)$-approximation algorithm has been proposed for it [16]. Note that given an allocation, the problem of scheduling in hybrid machines reduces to the above generalized dedicated processors problem.

On a more practical side, there exist some work about off-line scheduling, such as the well-known algorithm HEFT introduced by Topcuoglu *et al.* [6], which has been implemented on the run-time system starPU [17]. Another work studied the systematic comparison of various heuristics [18]. Specifically, the authors examined 11 different heuristics. This study provided a good basis for comparison and insights on circumstances why a technique outperforms another. Finally, Bleuse *et al.* [14] compared their proposed $(\frac{4}{3} + \frac{1}{3k})$-approximation algorithm with HEFT. Note that the later two approaches considered only independent tasks.

III. Preliminaries and Lower Bounds

In this section we briefly present the two basic existing approaches for scheduling on heterogeneous/hybrid platforms and we discuss their theoretical efficiency by presenting lower bounds on their performance.

The first approach is the scheduling oriented algorithm HEFT [6]. According to this algorithm, the tasks are initially prioritized with respect to their precedence relations and their average processing times, and then, following this priority, they are scheduled on the available pair of processor and time interval in which they feasibly complete as early as possible. Note that HEFT is a heuristic that works for platforms with several heterogeneous resources and it takes also into account possible communication costs. However, even for the simpler setting which we study in this paper without communication costs and only two type of resources, HEFT cannot have a worst-case approximation guarantee better than $\frac{m}{2}$ [14]. This result depends only on the number of CPUs, since the example provided uses just one GPU. The following theorem, whose proof is given in the Appendix, slightly improves the above result for the case of a single GPU, but, more interestingly, it describes in a better way the relation between the lower bound to the approximation ratio of HEFT and the equilibrium or not of the available types of machines.

Theorem 3.1: The worst-case approximation ratio for HEFT is at least $\frac{m+k}{k^2}\left(1 - \frac{1}{e^k}\right)$ even in the hybrid CPU/GPU model with independent tasks.

The second approach is proposed by Kedad-Sidhoum et al. [7] and it distinguishes the allocation and the scheduling decisions. For the allocation phase, an integer linear program is proposed which decides the allocation of tasks to the CPU or GPU side by optimizing the standard lower bounds for the makespan of a schedule which are proposed by Graham [10], namely the critical path and the load. In order to present this integer linear program, let x_j be a binary variable which is equal to 1 if a task $T_j \in \mathcal{T}$ is assigned to the CPU side, and zero otherwise. Let also C_j be a variable that indicates the completion time of T_j and λ the variable that corresponds to the maximum over all lower bounds used, i.e., to a lower bound of the makespan. Then, the Heterogeneous Linear

Program (HLP) is as follows:

$$\text{minimize } \lambda$$

$$C_i + \overline{p_j} x_j + \underline{p_j}(1-x_j) \le C_j \quad \forall T_j \in \mathscr{T}, T_i \in \Gamma^-(T_j) \quad (1)$$

$$\overline{p_j} x_j + \underline{p_j}(1-x_j) \le C_j \quad \forall T_j \in \mathscr{T} : \Gamma^-(T_j) = \emptyset \quad (2)$$

$$C_j \le \lambda \quad \forall T_j \in \mathscr{T} \quad (3)$$

$$\left(\sum_{T_j \in \mathscr{T}} \overline{p_j} x_j \right) / m \le \lambda \quad (4)$$

$$\left(\sum_{T_j \in \mathscr{T}} \underline{p_j}(1-x_j) \right) / k \le \lambda \quad (5)$$

$$x_j \in \{0,1\} \quad \forall T_j \in \mathscr{T} \quad (6)$$

$$C_j \ge 0 \quad \forall T_j \in \mathscr{T}$$

$$\lambda \ge 0$$

Constraints (1), (2) and (3) describe the critical path, while Constraints (4) and (5) impose that the makespan cannot be smaller than the load on CPU and GPU side, respectively. Note that the particular problem of deciding the allocation in order to minimize the maximum over the three lower bounds is NP-hard, since it is a generalization of the PARTITION problem to which reduces if all tasks are independent, $m = k$, and $\overline{p_j} = \underline{p_j}$ for each $T_j \in \mathscr{T}$.

After relaxing the integrity Constraint (6), a fractional allocation can be found in polynomial time. In order to get an integral solution, the variables x_j are rounded as follows: If $x_j \ge \frac{1}{2}$ then T_j is assigned to the CPU side; otherwise it is assigned to the GPU side. Finally, the Earliest Starting Time (EST) policy is applied for scheduling the tasks: At each step, the ready task with the earliest possible starting time is scheduled with respect to the precedence relations and the already decided allocation. We call the above algorithm HLP-EST.

HLP-EST achieves an approximation ratio of 6 [7]. Surprisingly, the following theorem shows that this ratio is tight.

Theorem 3.2: There is an instance for which HLP-EST achieves an approximation ratio of $6 - O(\frac{1}{m})$. Hence, the ratio for HLP-EST is tight.

Proof: Consider an hybrid system with an equal number of CPUs and GPUs, i.e, $m = k$. The instance consists of $2m+3$ tasks that are partitioned into 3 sets as shown in the following table.

Sets of tasks	# tasks per set	$\overline{p_j}$	$\underline{p_j}$
A	1	$\frac{m(2m+1)}{m-1}$	∞
B_1	$2m+1$	$2m-1$	1
B_2	$2m+1$	1	$2m-1$

The only precedence relations exist between tasks of B_1 and B_2. Specifically, for each task $T_j \in B_2$ we have that $\Gamma^-(T_j) = B_1$, that is no task in B_2 can be executed before the completion of all tasks in B_1. Note that there are no precedences between tasks of the same set.

Any optimal solution of the relaxed HLP for the above instance will assign the task T_A on a CPU, i.e., $x_A = 1$. Hence, the objective value of any optimal solution will be at least $\frac{m(2m+1)}{m-1}$ due to Constraints (2) and (3). The following technical proposition, whose proof is given in the Appendix, shows that an optimal solution for the relaxed HLP has exactly this objective value, by describing a feasible fractional assignment for the remaining tasks.

Proposition 3.3: There is a small constant $\varepsilon > 0$ for which the assignment $x_A = 1$, $x_j = \frac{1}{2}$ for each $T_j \in B_1$, $x_j = \frac{1}{2} - \varepsilon$ for each $T_j \in B_i$, and $\lambda = \frac{m(2m+1)}{m-1}$ corresponds to a feasible solution for the relaxed HLP.

Given the optimal fractional assignment proposed by the above proposition, HLP-EST will round the fractional variables and allocate the tasks as follows: the task T_A is assigned to the CPU side, each task $T_j \in B_1$ is assigned to the CPU side, and each task $T_j \in B_2$ is assigned to the GPU side. Then, HLP-EST schedules the tasks according to the EST policy. However, we will argue here for any possible schedule.

Assuming that an algorithm has scheduled the task T_A on any CPU during any interval $[t, t+\overline{p_A})$ and $m \ge 3$, there is only one meaningful family of schedules for the tasks in $B_1 \cup B_2$. Specifically, the $2m+1$ tasks of B_1 will be scheduled during the interval $[0, 3(2m-1))$ on the m CPUs, while at least one of them completes at time $3(2m-1)$. Then, the $2m+1$ tasks of B_2 will be scheduled during the interval $[3(2m-1), 6(2m-1))$ on the $k = m$ GPUs, while at least one of them completes at time $6(2m-1)$. Clearly, we should define t such that $t + \overline{p_A} \le 6(2m-1)$. An illustration of the above schedule is given in Figure 1.

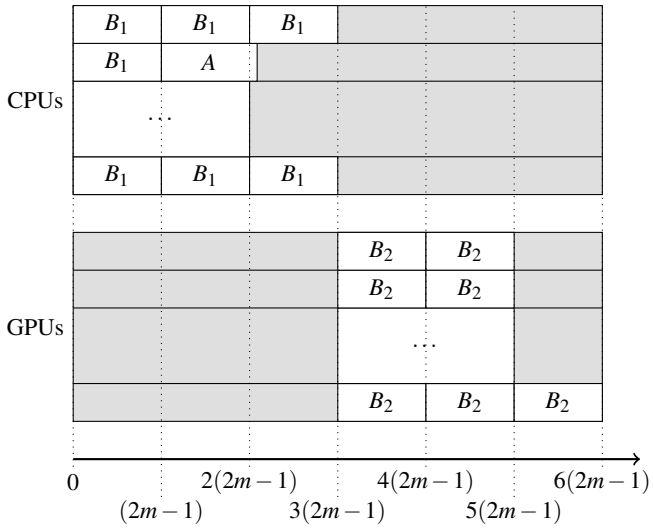

Figure 1. Resulting schedule of HLP-EST for the proposed instance. Notice that the gray areas represent idle times.

The makespan of the created schedule is equal to $6(2m-$

1), while Proposition 3.3 implies a feasible solution for the relaxed HLP of objective value $\frac{m(2m+1)}{m-1}$. Hence, the approximation ratio achieved for this instance is

$$\frac{6(2m-1)}{\frac{m(2m+1)}{m-1}} = 6 - O\left(\frac{1}{m}\right)$$

and the theorem follows. ■

Note that the proof of the previous theorem implies a stronger result since the worst case example does not depend on which scheduling policy will be applied after the allocation step, and hence the following corollary holds.

Corollary 3.4: Any scheduling policy which is applied after the allocation decisions taken by the rounding of an optimal solution of the relaxed HLP leads to an approximation algorithm of ratio at least $6 - O(\frac{1}{m})$.

IV. Algorithms

In this section we focus separately on each of the two phases, allocation and scheduling, and we propose algorithms for them. We then extend the HLP-EST algorithm to deal with $Q \geq 2$ resource types.

A. Allocation phase

In the HLP-EST algorithm, an integer linear program was used to find an efficient allocation of each task to the CPU or GPU side. Although this program optimizes the classical lower bounds for the makespan, and hence informally optimizes the allocation, the resolution of its relaxation has a high complexity in practice. For this reason, we would like to explore some greedy, low complexity, policies that could replace it.

In this direction, we initially propose the following three simple greedy rules:

- R1 If $\frac{\overline{p_j}}{m} \leq \frac{\underline{p_j}}{k}$ then assign T_j to the CPU side; otherwise assign it to the GPU side.
- R2 If $\frac{\overline{p_j}}{\sqrt{m}} \leq \frac{\underline{p_j}}{\sqrt{k}}$ then assign T_j to the CPU side; otherwise assign it to the GPU side.
- R3 If $\overline{p_j} \leq \underline{p_j}$ then assign T_j to the CPU side; otherwise assign it to the GPU side.

However these rules do not take into account neither the critical path nor the actual schedule and they cannot guarantee a bounded approximation ratio.

In what follows, we propose to use a more enhanced set of rules which combines R2 with another rule based on the structure of the actual schedule, in a similar way as in the 4-competitive algorithm proposed by Chen *et al.* [15] for the on-line problem with independent tasks. Our algorithm works also in the on-line setting.

In order to describe our algorithm, we define τ^C (resp. τ^G) to be the earliest time when at least one CPU (resp. GPU) is idle. We also define $R_j^C = \max\{\tau^C, \max_{i\in\Gamma^-(j)}\{C_i\}\}$ to be the *ready time* of task T_j, i.e., the earliest time at which T_j can be executed on a CPU. In a similar way, we define R_j^C for the GPU side. Then, the new enhanced set of rules is defined as follows:

- Step 1: If $\overline{p_j} \geq R_j^G + \underline{p_j}$ then assign T_j to the GPU side.
- Step 2: Apply R2.

After the allocation of each task T_j and before the arrival of the next task, we schedule T_j as early as possible on the CPU or GPU side already decided. We call the above algorithm ER-LS (Enhanced Rules - List Scheduling).

Theorem 4.1: ER-LS is a $(4\sqrt{\frac{m}{k}})$-competitive algorithm.

Proof: Let W_C, W_G and CP be the total load on all CPUs, the total load on all GPUs and the length of the critical path of a schedule produced by the algorithm, respectively. We will prove that $C_{max} \leq \frac{W_C}{m} + \frac{W_G}{k} + CP$. Then, we will bound the average load of both sides ($\frac{W_C}{m} + \frac{W_G}{k}$) by $3\sqrt{\frac{m}{k}}OPT$ and the length of the critical path by $\sqrt{\frac{m}{k}}OPT$. Recall that OPT denotes the makespan of the optimal off-line solution of the instance.

We denote by A_c (resp. A_g) the set containing the tasks placed on the CPU (resp. GPU) side in both a solution of the algorithm and the optimal solution, by B_g the set containing tasks placed by Step 1 on the GPU side in a solution of the algorithm but on the CPU side in the optimal solution, and by C_c (resp. C_g) the set containing tasks placed by Step 2 on the CPU (resp. GPU) side in a solution of the algorithm but on the GPU (resp. CPU) side in the optimal solution. The same notation in lower case is used to denote the sum of the processing times of all tasks in the set.

Bounding the loads. Consider T_{j_0} to be the last finishing task in B_g. Since the task is scheduled according to Step 1, we know that $\overline{p_{j_0}} \geq R_{j_0}^G + \underline{p_{j_0}} \geq \frac{b_g}{k}$. We also know that T_{j_0} is scheduled on a CPU in the optimal solution so we have $\overline{p_{j_0}} \leq OPT$ and then $\frac{b_g}{k} \leq OPT$.

Each task in C_g is scheduled on the CPU side in the optimal solution. According to Step 2, the total processing times of tasks in C_g in the optimal solution is at least $\sqrt{\frac{m}{k}}c_g$, so we have for the CPU side $\frac{a_c + \sqrt{\frac{m}{k}}c_g}{m} \leq OPT$. The same reasoning for the GPU side gives $\frac{a_g + \sqrt{\frac{k}{m}}c_c}{k} \leq OPT$.

By adding the three inequalities we have the following:

$$\frac{b_g}{k} + \frac{a_c + \sqrt{\frac{m}{k}}c_g}{m} + \frac{a_g + \sqrt{\frac{k}{m}}c_c}{k} \leq 3OPT$$

By separating the loads on CPU and on GPU on the left-hand side of the above inequality and taking into account that $m \geq k$ we have:

$$\frac{a_c}{m} + \frac{c_c}{\sqrt{mk}} \geq \frac{a_c + c_c}{m} \geq \sqrt{\frac{k}{m}}\frac{a_c + c_c}{m}$$

and

$$\frac{a_g + b_g}{k} + \frac{c_g}{\sqrt{mk}} \geq \frac{a_g + b_g}{k} + \frac{c_g}{k}\sqrt{\frac{k}{m}} \geq \sqrt{\frac{k}{m}}\frac{a_g + b_g + c_g}{k}$$

Summing these two bounds we finally have

$$\sqrt{\frac{k}{m}}\left(\frac{a_c+c_c}{m}+\frac{a_g+b_g+c_g}{k}\right)\leq 3OPT$$

and thus

$$\frac{W_C}{m}+\frac{W_G}{k}\leq 3\sqrt{\frac{m}{k}}OPT$$

Bounding the critical path. Consider the sets $A_c^{CP}\subseteq A_c$, $A_g^{CP}\subseteq A_g$, $B_g^{CP}\subseteq B_g$, $C_c^{CP}\subseteq C_c$ and $C_g^{CP}\subseteq C_g$ to be the sets containing only the tasks belonging to the critical path obtained by the algorithm, with the same notation in lower case for the sum of processing times of all tasks in each set and the same notation with a star * for the sum of processing times of all tasks in the optimal solution.

For the sets A_c^{CP} and A_g^{CP}, by definition, we have

$$a_c^{CP}+a_g^{CP}=a_c^{CP^*}+a_g^{CP^*}$$

According to Step 1, every task in B_g^{CP} has a processing time smaller than that in the optimal solution, so $b_g^{CP}\leq b_g^{CP^*}$. According to Step 2, every task T_j in C_c^{CP} (resp. C_g^{CP}) verifies $\overline{p_j}\leq\sqrt{\frac{m}{k}}\underline{p_j}$ (resp. $\underline{p_j}\leq\sqrt{\frac{k}{m}}\overline{p_j}$), so we have $c_c^{CP}\leq\sqrt{\frac{m}{k}}c_c^{CP^*}$ and $c_g^{CP}\leq\sqrt{\frac{m}{k}}c_g^{CP^*}$.

By summing the five previous inequalities for the critical path we get

$$\begin{aligned}CP&=a_c^{CP}+a_g^{CP}+b_g^{CP}+c_c^{CP}+c_g^{CP}\\&\leq\sqrt{\frac{m}{k}}(a_c^{CP^*}+a_g^{CP^*}+b_g^{CP^*}+c_c^{CP^*}+c_g^{CP^*})\leq\sqrt{\frac{m}{k}}CP^*\end{aligned}$$

Since $CP^*\leq OPT$, we have $CP\leq\sqrt{\frac{m}{k}}OPT$, and the theorem follows. ■

B. Scheduling phase

We propose here a new scheduling policy which prioritizes the tasks based on the solution obtained for HLP. The motivation of assigning priorities to the tasks is for taking into account the precedence relations between them. More specifically, we want to prioritize the scheduling of *critical tasks*, i.e., the tasks on the critical path, before the remaining (less critical) tasks.

In order to do this, for each task T_j we define a rank $Rank(T_j)$ in the same sense as in the HEFT algorithm. However, in our case, the rank of each task depends on HLP, while in HEFT it depends on the average processing time of the task. Specifically, the rank of each task T_j is computed after the rounding operation of the assignment variable x_j and corresponds to the length, in the sense of processing time, of the longest path between this task and its last descendant in the precedence graph. Thus, each task will have a larger rank than all its descendants. The rank of the task T_j is recursively defined as follows:

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

After ordering the tasks in non-increasing order with respect to their ranks, we apply the standard List Scheduling algorithm adapted to two types of resources and taking into account the rounding of the assignment variables x_j. We call the above described policy Ordered List Scheduling (OLS), while the newly defined algorithm (including the allocation) is denoted by HLP-OST.

Although this policy performs well in practice, as we will see in the experiments in the following section, its approximation ratio cannot be better than 6 due to the lower bound presented in Corollary 3.4. On the other hand, it is quite easy to see that HLP-EST and HLP-OST have the same approximation ratio, and hence this proof is omitted.

C. Generalization on Q Resource Types

We now generalize the HLP-EST algorithm to extend the addressed problem to $Q\geq 2$ different types of identical processors. Before continuing we need some additional notation. Let M_q be the set of processors of type q, $1\leq q\leq Q$, and $m_q=|M_q|$ its size. The execution of a task $T_j\in\mathscr{T}$ on a processor of type q, $1\leq q\leq Q$, takes $p_{j,q}$ time.

In what follows we adapt the HLP to take into account more resource types. In order to do this, we introduce a binary variable $x_{j,q}$ which indicates if the task $T_j\in\mathscr{T}$ is assigned to the resource type q. As before, let C_j be a variable corresponding to the completion time of T_j and λ be the variable that represents a lower bound to the makespan. Then, we consider the following modification of HLP, which we call QHLP:

$$\text{minimize } \lambda$$

$$C_i+\sum_{q=1}^{Q}p_{j,q}x_{j,q}\leq C_j \qquad \forall T_j\in\mathscr{T}, T_i\in\Gamma^-(T_j) \tag{7}$$

$$\sum_{q=1}^{Q}p_{j,q}x_{j,q}\leq C_j \qquad \forall T_j\in\mathscr{T}:\Gamma^-(T_j)=\emptyset \tag{8}$$

$$C_j\leq\lambda \qquad \forall T_j\in\mathscr{T} \tag{9}$$

$$\left(\sum_{T_j\in\mathscr{T}}p_{j,q}x_{j,q}\right)/m_q\leq\lambda \qquad 1\leq q\leq Q \tag{10}$$

$$\sum_{q=1}^{Q}x_{j,q}=1 \qquad \forall T_j\in\mathscr{T} \tag{11}$$

$$x_{j,q}\in\{0,1\} \qquad \forall T_j\in\mathscr{T},\ 1\leq q\leq Q \tag{12}$$

$$C_j\geq 0 \qquad \forall T_j\in\mathscr{T}$$

$$\lambda\geq 0$$

The main difference here concerns Constraint (11) which assures that each task is integrally assigned into exactly one type of resources.

After relaxing the integrity Constraint (12) of QHLP, we can solve in polynomial time the obtained relaxation. In order to get an integral allocation, we assign each task T_j to the resource type q' for which the assignment variable

$x_{j,q}$ have the greatest value, i.e., $q' = \text{argmax}_{1 \leq q \leq Q}\{x_{j,q}\}$. In other words, for such q' we set $x_{j,q'} = 1$ and $x_{j,q} = 0$ for any $q \neq q'$. In case of ties, we give priority to the resource type in which T_j has the smallest processing time. Once the assignment step is done, we use the Earliest Starting Time policy taking into account the precedence constraints as well as the allocation provided by the rounding of $x_{j,q}$ variables. We call this algorithm QHLP-EST. Then, the following theorem, whose proof is given in the Appendix, holds.

Theorem 4.2: QHLP-EST achieves an approximation ratio of $Q(Q+1)$. This ratio is tight.

V. Experiments

In this section, we compare the performance of different scheduling algorithms by an extensive campaihn of simulations using a benchmark composed of 6 parallel applications. In what follows, we describe the generation of the benchmark, as well as the experimental environment, and we analyse the results.

A. Benchmark

The benchmark is composed of six parallel applications. Five of them have been generated from Chameleon, a dense linear algebra software which is part of the MORSE project [8], while the sixth has been generated with GGen, a library for generating directed acyclic graphs [9], and it corresponds to a more irregular application.

The five applications of Chameleon, named *getrf_nopiv*, *posv*, *potrs*, *potri* and *potrs*, are composed of multiple sequential basic tasks of linear algebra such as *SYRK* (symetric rank update), *GEMM* (general matrix-matrix multiply) and *TRSM* (triangular matrix equation solver), as shown in Table I. To generate the applications, different tilings of

Table I
Basic kernel of linear algebra of each application

Kernels \ Apps	getrf_nopiv	posv	potrf	potri	potrs
syrk		x	x	x	
gemm	x	x	x	x	x
trsm	x	x	x	x	x

the matrices have been used, varying the number of submatrices denoted by *nb_blocks* and the size of the submatrices denoted by *block_size*. The different values of *nb_blocks* were 5, 10 and 20 and the different values of *block_size* were 64, 128, 320, 512, 768 and 960, for a total of 18 configurations per application. Table II shows the total number of tasks for each application and each value of *nb_blocks*. Notice that the value of *block_size* does not impact the number of tasks.

The Chameleon applications were executed with the runtime StarPU [17] and the traces of executions were collected. At first, all the applications were executed on CPUs and then

Table II
Total number of tasks in function of the number of blocks

Nb_blocks \ Apps	getrf_nopiv	posv	potrf	potri	potrs
5	55	65	35	105	30
10	385	330	220	660	110
20	2870	1960	1540	4620	420

were forced to execute on GPUs to have the processing times of each task of the application for both computing units.

The machine where this data was collected had the following hardware characteristics: Dual core Xeon E7 v2, with a total of 20 physical cores with hyper-threading of 3 GHz of processor base Frequency, 256 GB of RAM, operative system Linux Ubuntu 14.04. This machine had 4 GPUs NVIDIA Tesla K20 with each 4 GB of global memory, 200 GB/s of bandwidth and 2496 cores divided in 13 multiprocessors.

The execution time of each task and their respective dependencies were collected and formatted in different files, in SWF (Standard Workload Format). The data set and other information are available[1] under Creative Commons Public License for the sake of reproducibility.

The application generated with GGen is *fork-join*. It represents a real application that starts by executing sequentially and then forks to be executed in parallel with a specific diameter (number of parallel tasks), when the parallel execution has completed, results are aggregated by performing a join operation. This procedure can be repeated several times depending on the number of phases. For our experiments, we used 2, 5 and 10 phases with a diameter of 100, 200, 300, 400 and 500 for a total of 15 different configurations. Table III shows the total number of tasks for each configuration of the fork-join application.

Table III
Total number of tasks in function of the number of phases and the width of the phase

Nb_phases \ Diameter	100	200	300	400	500
2	203	403	603	803	1003
5	506	1006	1506	2006	2506
10	1011	2011	3011	4011	5011

The running time of each task was computed using a Gaussian distribution with center p and standard deviation $\frac{p}{4}$, where p is the number of phases. We have decided to establish various acceleration factors in each diameter of fork. In this way, for all the configurations there are five parallel tasks with an acceleration factor between 0.1 and 0.5 while the others have an acceleration factor between 0.5 and 50.

[1]Hosted at: https://github.com/marcosamaris/heterogeneous-SWF [Accessed on 19 October 2016]

B. Environment and algorithms

We compared the performance, in terms of makespan for each instance of the applications, of the original HLP-EST algorithm as well as HLP-OLS, presented in Section IV-B, with the HEFT algorithm.

We also compared in on-line mode, where tasks arrive over a list, the algorithm ER-LS, presented in Section IV-A, with two pure greedy algorithms. The first greedy algorithm, denoted by GreedyOn, allocates a task on the processor type which has the smallest processing time for that task. The second one, denoted by RandomOn, randomly assigns a task to the CPU or GPU side.

Each algorithm takes as input an application, composed of a DAG of precedence between the tasks and the processing times on CPU and on GPU for each task, as well as the number of CPUs and GPUs on which the application is to be scheduled on. The three algorithms are implemented in Python, version 2.7.6, and the linear program solver used is the *glpsol* command-line solver, version 4.52, of the GLPK package (GNU Linear Programming Kit).

The number of tasks of the six applications ranged from 30 to 5011. Moreover, we determined different sets of pairs (Nb_CPUs, Nb_GPUs) for the experiments. Specifically, we used 16, 32, 64 and 128 CPUs with 2, 4, 8 and 16 GPUs for a total of 16 machine configurations.

Each combination of application and machine configuration has been executed only once since all algorithms, except for the random greedy algorithm and *glpsol*, are deterministic. For each run, we stored the computed value of the optimal objective solution of the linear program, denoted by LP^*, and the makespans of HLP-EST, HLP-OLS, HEFT as well as the on-line algorithms ER-LS, GreedyOn and RandomOn.

C. Analysis of results

Off-line algorithms. To study the performance of the 3 off-line algorithms we computed the ratio between each makespan and the optimal solution of the linear program LP^*, which corresponds to a good lower bound of the optimal makespan. Figure 2 shows the ratio of each instance of an application, with each configuration, grouped by application. Notice that the red / bigger dot represents the mean value of the ratio for each application. We can see that HLP-EST is outperformed, on average, by the two other algorithms. The performances of HLP-OLS and HEFT are quite similar, on average, but we observe that HEFT does create more outlier makespans.

Figures 3 and 4 compare more specifically the two Linear Program-based algorithms and the algorithms HLP-OLS and HEFT, respectively, by showing the ratio between the makespans of the two algorithms. We can see that the algorithm HLP-OLS clearly outperforms HLP-EST, except for a few instances with the application *potri*, with an improvement of nearly 10% on average. On Figure 4 we

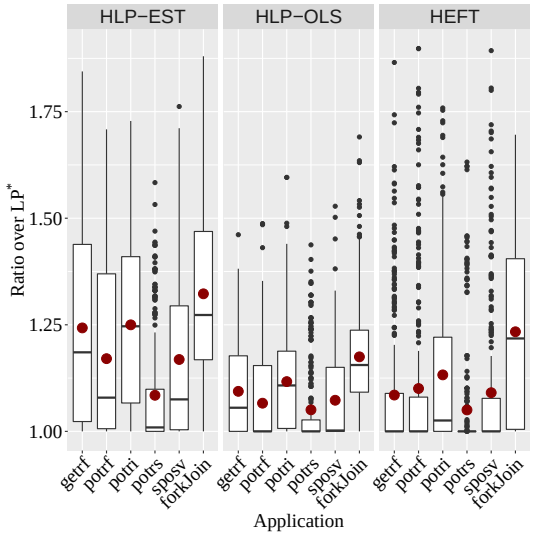

Figure 2. Ratios over LP^* of the 3 off-line algorithms for each instance, grouped by application.

notice that, even if the two algorithms have similar performances, HEFT is on average outperformed by HLP-OLS by 5%. Moreover, HEFT has a significantly worse performance than HLP-OLS in strongly heterogeneous applications where there is a bigger perturbation in the (dis-)acceleration of the tasks on the GPU side, like *forkJoin*, since in these irregular cases the allocation problem becomes more critical.

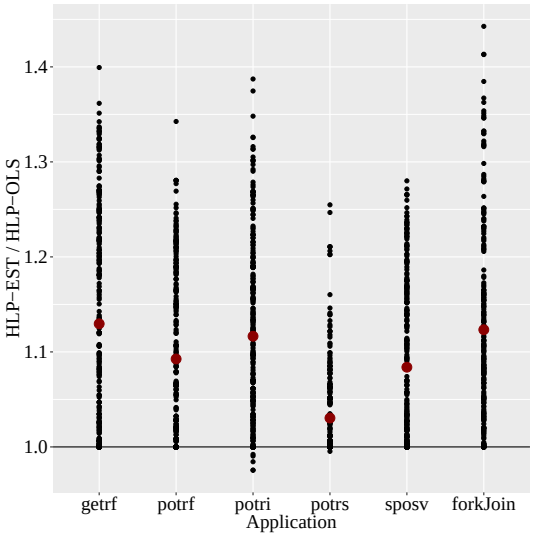

Figure 3. Ratio between the makespans of HLP-EST and HLP-OLS for each instance, grouped by application.

On-line algorithms. The ratios between the makespans and LP^* are also computed to compare the 3 on-line algorithms. As Figures 5 shows, the RandomOn algorithm performs badly compared to the two other algorithms. Except for

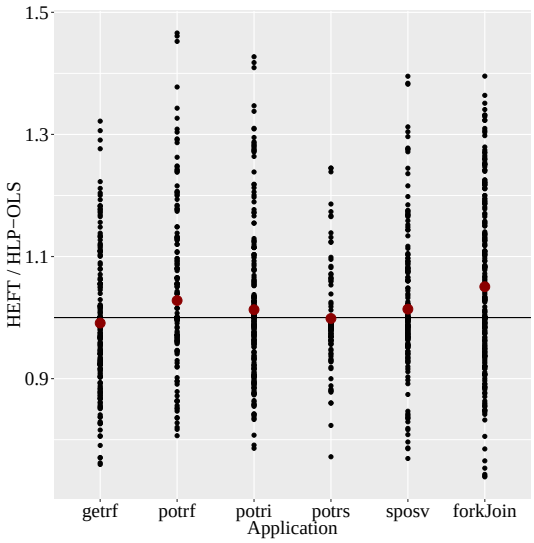

Figure 4. Ratio between the makespans of HEFT and HLP-OLS for each instance, grouped by application.

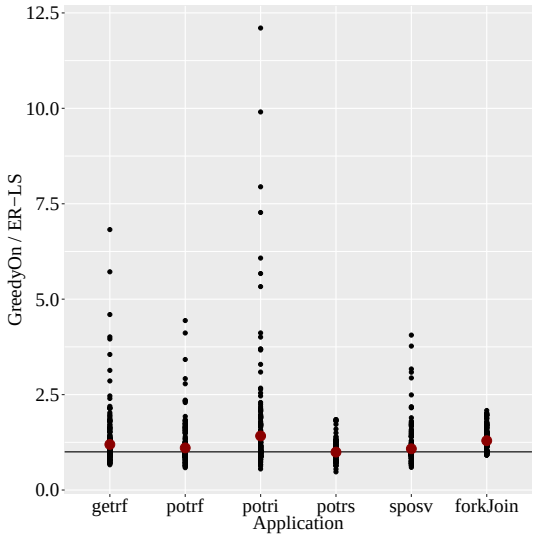

Figure 6. Ratio between the makespans of GreedyOn and ER-LS for each instance, grouped by application.

a few number of instances, the algorithm is significantly outperformed by ER-LS and GreedyOn.

Figure 6 presents more in detail the comparison between ER-LS and GreedyOn. As we can see, the ratio of the makespans is on average greater than 1, meaning that GreedyOn is outperformed by ER-LS. The mean value of the ratio per application is between 1 and 1.5 while, for some instances, GreedyOn can even perform up to 12.5 times worse than ER-LS.

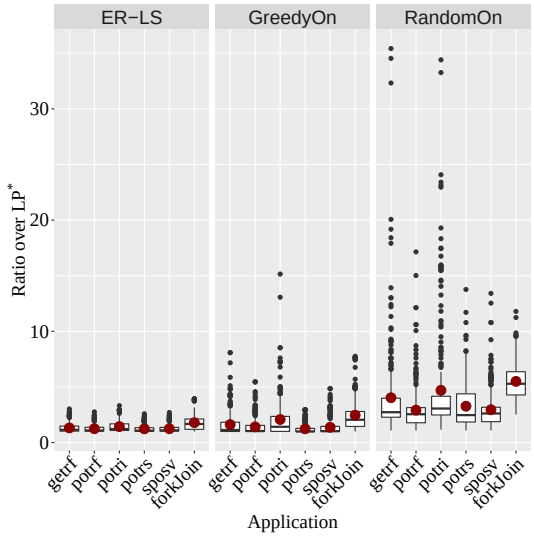

Figure 5. Ratios over LP^* of the 3 on-line algorithms for each instance, grouped by application.

We also studied the competitive ratio of the algorithms ER-LS and GreedyOn to compare with the theoretical upper bound of the ratio discussed in Section IV-A. Figure 7 shows the mean competitive ratio of ER-LS (plain line) and GreedyOn (dashed line) along with the standard error in function of the ratio $\sqrt{\frac{m}{k}}$ associated to each instance. To simplify the lecture of the figure, we only selected the applications potri and fork-join. The competitive ratio is smaller than $\sqrt{\frac{m}{k}}$ and, thus, far from the theoretical upper bound of $4\sqrt{\frac{m}{k}}$.

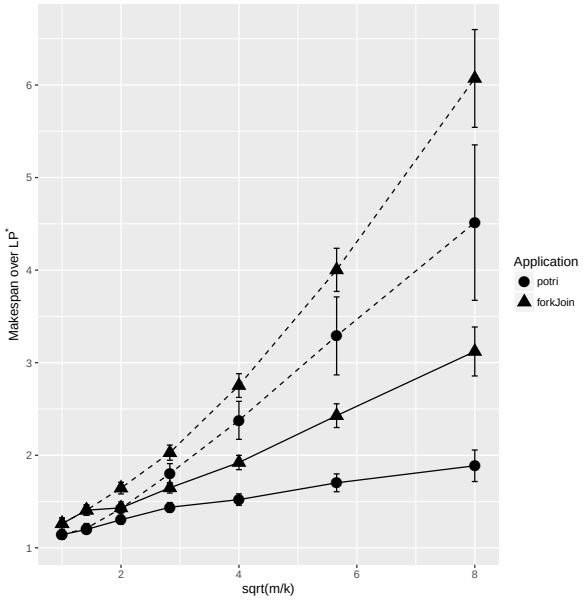

Figure 7. Competitive ratio of the algorithms ER-LS, in plain line, and GreedyOn, in dashed line, in function of the ratio $\sqrt{\frac{m}{k}}$.

VI. Conclusions

We studied the problem of scheduling parallel applications, represented by a precedence task graph, on hybrid multi-core machines. We focused on generic approaches, non depending on the particular application, by distinguishing the allocation and the scheduling phases and we proposed several efficient algorithms with worst-case performance guarantees for both off-line and on-line settings. In the off-line case, motivated by new lower bounds on the performance of existing algorithms, we refined the scheduling phase and presented an algorithm that preserves the approximation guarantee of the best known existing algorithm but also performs very well in our experimental campaign. We also extended this methodology for the more general case where the architecture is composed of Q types of resources. In the on-line case, we presented an algorithm of competitive ratio equal to $O(\sqrt{\frac{m}{k}})$ based on an adequate set of rules, which could be considered as constant since, practically, the ratio $\frac{m}{k}$ is bounded.

From the practical point of view, the experiments based on an extensive simulation campaign on representative benchmarks constructed by real applications, showed that it is possible to outperform the classical HEFT algorithm keeping reasonable running times. Moreover, the on-line algorithm based on rules is a very good trade-off since it delivers a solution close to the optimal in reasonable time. We aim at implementing it on a real run-time system (such as StarPU [17]) which currently uses HEFT on successive sets of independent tasks.

This work was done under the assumption that the communications between CPUs and GPUS and the shared memory are neglected. Our next objective is to introduce communication costs in the algorithms, which should not be too hard in both integer program and interactive rules.

Acknowledgment

We would like to thank the team of the project MORSE, especially Manuel Agullo and Mathieu Faverge, for the collaboration in this work. This work was partially supported by FAPESP (São Paulo Research Foundation, grant number #2012/23300-7) and ANR Moebus Project.

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

References

[1] V. W. Lee *et al.*, "Debunking the 100x gpu vs. cpu myth: An evaluation of throughput computing on cpu and gpu," *SIGARCH Comput. Archit. News*, vol. 38, no. 3, pp. 451–460, 2010.

[2] TOP-500-Supercomputer, " [Web site http://www.top500.org] Visited on Oct 2016."

[3] L. G. Valiant, "A bridging model for multi-core computing," in *ESA*, ser. LNCS, vol. 5193. Springer, 2008, pp. 13–28.

[4] M. Amaris, D. Cordeiro, A. Goldman, and R. Y. Camargo, "A simple bsp-based model to predict execution time in gpu applications," in *HiPC*, 2015, pp. 285–294.

[5] M. Drozdowski, *Scheduling for Parallel Processing*, 1st ed. Springer Publishing Company, Incorporated, 2009.

[6] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Task scheduling algorithms for heterogeneous processors," in *Heterogeneous Computing Workshop (HCW)*, 1999, pp. 3–14.

[7] S. Kedad-Sidhoum, F. Monna, and D. Trystram, "Scheduling Tasks with Precedence Constraints on Hybrid Multi-core Machines," in *HCW - IPDPS Workshops*, 2015, pp. 27–33.

[8] E. Agullo *et al.*, "Poster: Matrices over runtime systems at exascale," in *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, 2012, pp. 1332–1332.

[9] D. Cordeiro *et al.*, "Random graph generation for scheduling simulations," in *ICST (SIMUTools)*, 2010.

[10] R. L. Graham, "Bounds on multiprocessing timing anomalies," *SIAM Journal On Applied Mathematics*, vol. 17, no. 2, pp. 416–429, 1969.

[11] O. Svensson, "Hardness of precedence constrained scheduling on identical machines," *SIAM J. Comput.*, vol. 40, no. 5, pp. 1258–1274, 2011.

[12] F. A. Chudak and D. B. Shmoys, "Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds," *Journal of Algorithms*, vol. 30, pp. 323–343, 1999.

[13] C. Chekuri and M. Bender, "An efficient approximation algorithm for minimizing makespan on uniformly related machines," *J. Algorithms*, vol. 41, no. 2, pp. 212–224, 2001.

[14] R. Bleuse, S. Kedad-Sidhoum, F. Monna, G. Mounié, and D. Trystram, "Scheduling independent tasks on multi-cores with GPU accelerators," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 6, pp. 1625–1638, 2015.

[15] L. Chen, D. Ye, and G. Zhang, "Online scheduling of mixed cpu-gpu jobs," *International Journal of Foundations of Computer Science*, vol. 25, no. 06, pp. 745–761, 2014.

[16] J. W. Liu and C. L. Liu, "Performance analysis of multiprocessor systems containing functionally dedicated processors," *Acta Inf.*, vol. 10, no. 1, pp. 95–104, 1978.

[17] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier, "Starpu: A unified platform for task scheduling on heterogeneous multicore architectures," *Concurr. Comput. : Pract. Exper.*, vol. 23, no. 2, pp. 187–198, 2011.

[18] T. D. Braun *et al.*, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *J. Parallel Distrib. Comput.*, vol. 61, no. 6, pp. 810–837, 2001.

APPENDIX

Theorem 3.1: The worst-case approximation ratio for HEFT is at least $\frac{m+k}{k^2}\left(1-\frac{1}{e^k}\right)$ even in the hybrid CPU/GPU model with independent tasks.

Proof: We describe an instance that consists of independent tasks, and hence no communication costs are defined. We also consider the hybrid platform model where we only have a set of m identical CPUs and a set of k identical GPUs. Then, the rank of each task $T_j \in \mathscr{T}$ computed by HEFT is simplified as follows:

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

HEFT considers the tasks in non-increasing order with respect to their rank and assigns each task to the CPU or GPU where its completion time is minimized. In case of ties, we assume, without loss of generality, that HEFT prefers to assign the task to a GPU, while it chooses arbitrarily between CPUs or GPUs. Notice that, since all tasks are independent, no idle times are introduced in the schedule.

Our instance consists of the following $2m$ sets of $km+m^2$ tasks in total.

Sets of tasks	# tasks per set	$\overline{p_j}$	$\underline{p_j}$
A_i, $1 \le i \le m$	k	$\left(\frac{m}{m+k}\right)^i$	$\left(\frac{m}{m+k}\right)^i$
B_i, $1 \le i \le m$	m	$\left(\frac{m}{m+k}\right)^i$	$\frac{k}{m^2}\left(\frac{m}{m+k}\right)^m$

The rank of each task $T_j \in A_i$, $1 \le i \le m$, is:

$$rank(T_j) = \frac{(m+k)\left(\frac{m}{m+k}\right)^i}{m+k}$$

while the rank of each task $T_j \in B_i$, $1 \le i \le m$, is:

$$rank(T_j) = \frac{m\left(\frac{m}{m+k}\right)^i + \frac{k^2}{m^2}\left(\frac{m}{m+k}\right)^m}{m+k}$$

According to the above ranks, HEFT will schedule all tasks in A_{i+1} (resp. B_{i+1}) after all tasks in A_i (resp. B_i), $1 \le i \le m-1$. Moreover, for any $T_j \in A_i$ and $T_{j'} \in B_i$, $1 \le i \le m$, we have

$$\begin{aligned}
&(m+k)\left(rank(T_j) - rank(T_{j'})\right) \\
&= (m+k)\left(\frac{m}{m+k}\right)^i - m\left(\frac{m}{m+k}\right)^i - \frac{k^2}{m^2}\left(\frac{m}{m+k}\right)^m \\
&= k\left(\left(\frac{m}{m+k}\right)^i - \frac{k}{m^2}\left(\frac{m}{m+k}\right)^m\right) \\
&\ge k\left(\left(\frac{m}{m+k}\right)^m - \frac{k}{m^2}\left(\frac{m}{m+k}\right)^m\right) > 0
\end{aligned}$$

where the last inequality holds since $k \le m$. For any $T_j \in B_i$ and $T_{j'} \in A_{i+1}$, $1 \le i \le m-1$, we have

$$\begin{aligned}
&(m+k)\left(rank(T_j) - rank(T_{j'})\right) \\
&= m\left(\frac{m}{m+k}\right)^i + \frac{k^2}{m^2}\left(\frac{m}{m+k}\right)^m - (m+k)\left(\frac{m}{m+k}\right)^{i+1} \\
&> \left(\frac{m}{m+k}\right)^i \left(m - (m+k)\frac{m}{m+k}\right) = 0
\end{aligned}$$

Based on the above, HEFT will consider the sets of tasks according to the following order:

$$A_1 \prec B_1 \prec A_2 \prec B_2 \prec \cdots \prec A_i \prec B_i \prec A_{i+1} \prec \cdots \prec A_m \prec B_m$$

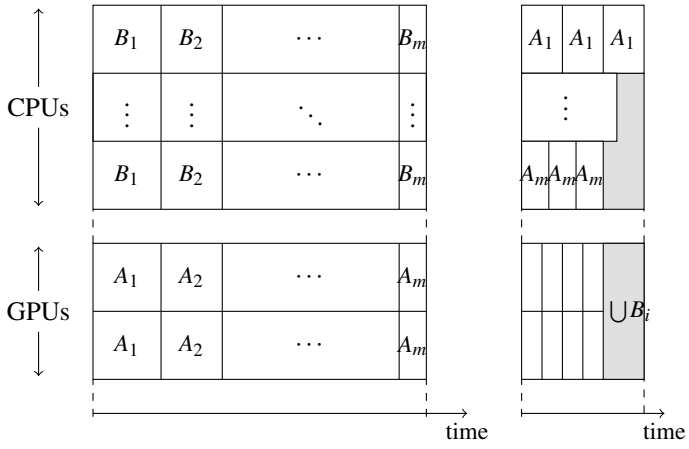

Figure 8. Possible schedule of HEFT (left) and optimal schedule (right). Notice that the gray area represents idle time.

Initially, HEFT will schedule the k tasks in A_1 in a different GPU. Hence, in order to minimize the completion times of the m tasks in B_1, each one should be scheduled on a different CPU. Note that, all tasks in $A_1 \cup B_1$ finish at the same time, i.e., at time $\frac{m}{m+k}$. Similarly, the tasks in A_2 will be scheduled on a different GPU, the tasks in B_2 on a different CPU, and all of them will finish at the same time, i.e., at time $\frac{m}{m+k} + \left(\frac{m}{m+k}\right)^2$. The scheduling procedure continues in the same way for the tasks in the remaining sets. The left-hand side of Figure 8 shows a schedule produced by HEFT. In this schedule, all machines finish their execution at time:

$$\begin{aligned}
\sum_{i=1}^{m}\left(\frac{m}{m+k}\right)^i &= \frac{1-\left(\frac{m}{m+k}\right)^{m+1}}{1-\frac{m}{m+k}} - 1 \simeq \frac{1-\frac{m}{m+k}\frac{1}{e^k}}{1-\frac{m}{m+k}} - 1 \\
&= \frac{(m+k)e^k - m}{ke^k} - 1 = \frac{me^k - m}{ke^k} = \frac{m}{k}\left(1-\frac{1}{e^k}\right)
\end{aligned}$$

On the other hand, we can create a schedule of makespan at most $\frac{km}{m+k}$. In order to see this, we assign all tasks of A_i, $1 \le i \le m$, on CPU i, while to each of the k GPUs we assign $\frac{m^2}{k}$ different tasks of $\bigcup_{i=1}^{m} B_i$. The right-hand side of Figure 8 visualizes such a schedule, whose makespan is dominated either by the load of CPU 1 or by the load of any of the

GPUs. Specifically, the makespan will be equal to

$$\max\left\{k\left(\frac{m}{m+k}\right), \frac{m^2}{k}\frac{k}{m^2}\left(\frac{m}{m+k}\right)^m\right\} \leq \frac{km}{m+k}$$

Since an optimal schedule could have an even smaller makespan, the theorem follows. ■

Proposition 3.3: There is a small constant $\varepsilon > 0$ for which the assignment $x_A = 1$, $x_j = \frac{1}{2}$ for each $T_j \in B_1$, $x_j = \frac{1}{2} - \varepsilon$ for each $T_j \in B_i$, and $\lambda = \frac{m(2m+1)}{m-1}$ corresponds to a feasible solution for the relaxed HLP.

Proof: We will show that every constraint of the relaxed HLP is satisfied by the assignment of the binary variables x_j proposed in the statement and by setting $LP^* = \frac{m(2m+1)}{m-1}$. But before this, we need to feasibly define C_j, for each $T_j \in \mathscr{T}$, based on Constraints (1) and (2).

For the task T_A, we set

$$C_A = \frac{m(2m+1)}{m-1}$$

for each task $T_j \in B_1$, we set

$$C_j = \frac{1}{2}(2m-1) + \frac{1}{2} = m$$

while for each task $T_j \in B_2$, we set

$$C_j = m + (\frac{1}{2} - \varepsilon) + (\frac{1}{2} + \varepsilon)(2m-1) = 2m + 2\varepsilon(m-1)$$

satisfying by definition Constraints (1) and (2).

In order to show the feasibility of Constraint (3), it suffices to prove it for T_A as well as for a task $T_j \in B_2$. For these cases, we have

$$C_A = \frac{m(2m+1)}{m-1} = LP^*$$
$$C_j = 2m + 2\varepsilon(m-1) \leq LP^*$$

where the last inequality holds for arbitrarily small ε, and hence Constraint (3) is satisfied.

For Constraint (4), we have

$$\begin{aligned}\sum_{T_j \in \mathscr{T}} \overline{p_j} x_j &= \overline{p_A} x_A + \sum_{T_j \in B_1 \cup B_2} \overline{p_j} x_j \\ &= \frac{m(2m+1)}{m-1} + (2m+1)\frac{2m-1}{2} + (2m+1)(\frac{1}{2} - \varepsilon) \\ &< \frac{m(2m+1)}{m-1} + m(2m+1) = m\frac{m(2m+1)}{m-1} = m\lambda\end{aligned}$$

and hence it is satisfied.

For Constraint (5), we have

$$\begin{aligned}\sum_{T_j \in \mathscr{T}} \underline{p_j}(1-x_j) &= \underline{p_A}(1-x_A) + \sum_{T_j \in B_1 \cup B_2} \underline{p_j}(1-x_j) \\ &= 0 + (2m+1)\frac{1}{2} + (2m+1)(2m-1)(\frac{1}{2} + \varepsilon) \\ &< m(2m+1) + \varepsilon(4m^2-1) \leq m\lambda = k\lambda\end{aligned}$$

where the last inequality is true for an arbitrarily small ε, and hence the constraint is satisfied.

Concluding, all constraints are satisfied with $\lambda = \frac{m(2m+1)}{m-1}$, and thus the proposition holds. ■

Theorem 4.2: QHLP-EST achieves an approximation ratio of $Q(Q+1)$. This ratio is tight.

Proof: We analyse the structure of a schedule produced by the algorithm to give an upper bound on the approximation ratio. The analysis of the algorithm and the structure of the schedule are similar to the ones of Kedad-Sidhoum *et al.* for the HLP-EST algorithm [7].

We denote by W_q, $1 \leq q \leq Q$, the total load on all processors of type q in the schedule. We also denote by C^R_{max}, W^R_q and L^R the objective value, the total load on all processors of type q and the length of the longest path in the fractional optimal solution of the relaxed QHLP, respectively. Finally, we define by $C^*_{\max}$ the optimal makespan over all feasible schedules for our problem. Then, the following inequalities hold.

$$L^R \leq C^R_{\max} \leq C^*_{\max} \tag{13}$$

$$\frac{W^R_q}{m_q} \leq C^R_{\max} \leq C^*_{\max},\ 1 \leq q \leq Q \tag{14}$$

To analyze the structure of the schedule, we partition the time interval of the schedule $I = [0, C_{max})$ into two disjoint subsets of intervals I_{CP} and I_W. The set I_{CP} contains every time slot where at least one processor of each type is idle, while the set I_W consists of the remaining time slots in I, i.e., $I_W = I \backslash I_{CP}$. We then can divide the set I_W into Q, possibly non-disjoint, subsets I_q, $1 \leq q \leq Q$, which contain every time slot where all processors of type q are busy. Henceforth, we denote by $|I|$ the number of unitary time slots in I. Then, we have that

$$C_{\max} = |I| \leq |I_{CP}| + \sum_{q=1}^{Q} |I_q|$$

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

$$x_{j,q} \leq Q \cdot x^R_{j,q} \quad \forall T_j \in \mathscr{T}, 1 \leq q \leq Q \tag{15}$$

Consider first the subset of intervals I_{CP}. There is a directed path $\mathscr{P}$ of tasks being executed during any time slot in I_{CP}. The construction of $\mathscr{P}$ is the same as described by Kedad-Sidhoum *et al.* [7]. Since the directed path $\mathscr{P}$ covers every time slot in I_{CP}, the length of I_{CP} is smaller than the length of $\mathscr{P}$ and the length of $\mathscr{P}$ in the optimal solution of (LP_q), noted $\mathscr{P}^R$, is smaller than L^R. Thus, using the inequalities (13) and (15), we have the following bound:

$$\begin{aligned}|I_{CP}| &\leq |\mathscr{P}| \leq \sum_{j \in \mathscr{P}} \sum_{q=1}^{Q} p_{j,q} x_{j,q} \leq Q \sum_{j \in \mathscr{P}} \sum_{q=1}^{Q} p_{j,q} x^R_{j,q} \\ &= Q|\mathscr{P}^R| \leq Q \cdot L^R \leq Q \cdot C^*_{\max}\end{aligned}$$

Consider now each subset I_q, $1 \leq q \leq Q$. For each time slot in I_q all processors of type q are busy, so $|I_q|$ is smaller than the average load on all the processors of type q. Using the inequalities (14) and (15), we have the following bounds:

$$\begin{aligned} |I_q| &\leq \frac{W_q}{m_q} \leq \frac{1}{m_q} \sum_{x_{j,q}=1} p_{j,q} \leq \frac{Q}{m_q} \sum_{j \in V} p_{j,q} x^R_{j,q} \\ &\leq \frac{Q \cdot W^R_q}{m_q} \leq Q \cdot C^*_{\max} \end{aligned}$$

Thus, by combining the calculated bounds we get

$$C_{\max} = |I| \leq |I_{CP}| + \sum_{q=1}^{Q} |I_q| \leq Q(Q+1)C^*_{max}$$

The tightness come directly from Theorem 3.2, and hence the theorem follows. ■