# Generic algorithms for scheduling applications on hybrid multi-core machines

Marcos Amaris[1 2], Giorgio Lucarelli[1], Clément Mommessin[1], and Denis Trystram[1]

[1] Univ. Grenoble Alpes, CNRS, Inria, LIG, F-38000 Grenoble France
{giorgio.lucarelli,clement.mommessin,denis.trystram}@imag.fr
[2] Institute of Mathematics and Statistics, University of São Paulo, São Paulo, Brazil
amaris@ime.usp.br

**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 most 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. 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 online scheduling algorithm which takes into account precedences. Our algorithm is based on adequate rules for selecting the type of processor where to allocate the tasks and it 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 on 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.

## 1 Introduction

The parallel and distributed platforms available today become more and more *heterogeneous*. Such heterogeneous architectures have a growing impact on performance in high-performance computing. Hardware accelerators, such as General Purpose Graphical Processing Units (in short GPUs) [12], 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 has significantly increased during the last years [14]. 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. In this way, the code is portable and can be adapted to the next generation of machines.

*Underlying architecture.* We consider an hybrid multi-core node composed of identical CPUs and 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 these processing times is available to the scheduler. This assumption can be justified by several existing models to estimate the execution times of tasks [2]. In several applications we always observe an acceleration of the tasks if they are executed on a GPU. However, we consider 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.

*Definition and notations.* We consider a parallel application which should be scheduled on $m$ identical CPUs and $k$ 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. We denote by $\mathcal{T}$ the set of all tasks. Let $\overline{p_j}$ (resp. $\underline{p_j}$) be the processing time of a task $T_j$ 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 $S$. 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 $T_j$. 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 $T_i$. We call *descendant* of $T_j$ 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 tasks, does not interrupt their execution and minimizes the completion time of the last task, i.e., $C_{\max} = \max_{T_j}\{C_j\}$. Extending the three-fields

notation for scheduling problems introduced by Graham, this problem can be denoted as $(CPU, GPU) \mid prec \mid C_{\max}$.

*Contributions and outline.* In this paper we study the above problem 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 [15]), we propose to address the problem by separately focusing on the following two phases:

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

We aim to study the two phases separately motivated by 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 [15] cannot be better than $\Omega(\frac{m}{k^2})$ (Section 3), while it can be easily shown that List Scheduling policies have arbitrarily large approximation ratio, even if we consider some enhanced order of tasks, like prioritizing the task of the largest acceleration. The two-phases approach has been used by Kedad-Sidhoum et al. [11] 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, henceforth called HLP-EST, achieves an approximation ratio of 6. Surprisingly, in Section 3, 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 4.1, we initially present three greedy rules which can be used to decide the allocation. Although these rules are of low complexity, a desired property in practice, they cannot guarantee any approximation ratio. However, a more enhanced set 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 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 4.2, we propose to replace the EST policy in HLP-EST by a specific order of tasks which is based on both the allocation decisions taken in the first phase and the critical path. This refined algorithm preserves the approximation ratio of 6 and it also has a very good practical performance.

In Section 5, we describe the generation of the benchmark used in our experiments, which is freely available in Standard Workload Format (SWF). The experiments show that the new scheduling method based on HLP outperforms both HEFT and HLP-EST in most of the applications, while our proposed on-line algorithm has significantly better makespan than the baseline greedy algorithms.

Before continuing, we present in Section 2 the works related to our setting and, finally, we conclude in Section 6. Omitted proofs can be found in [3].

## 2 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 tasks on two types of resources is more complex than the problem 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 assuming a particular variant of the Unique Games Conjecture [13]. Chudak and Shmoys [8] developed a polynomial-time $O(\log m)$-approximation algorithm for $Q \mid prec \mid C_{\max}$. For hybrid architectures, a 6-approximation algorithm has been proposed by Kedad-Sidhoum et al. [11]. In the case of independent tasks there is a $(\frac{4}{3} + \frac{1}{3k})$-approximation algorithm [5]. If the tasks arrive in an on-line order, a 4-competitive algorithm has been presented by Chen et al. [7] for hybrid architectures without precedence relations.

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. [15], which has been implemented on the run-time system starPU [4]. Another work studied the systematic comparison of various heuristics [6]. 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. [5] compared their proposed $(\frac{4}{3} + \frac{1}{3k})$-approximation algorithm with HEFT. Note that the later two approaches considered only independent tasks.

## 3 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 [15]. According to HEFT, the tasks are initially prioritized with respect to their precedence relations and their average processing times. Then, following this priority, tasks are scheduled with possible backfilling 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 also takes into account possible communication costs. However, even for the simpler setting which we study in this paper without communication costs, with only two types of resources and $k = 1$, HEFT cannot have a worst-case approximation

guarantee better than $\frac{m}{2}$ [5]. This result depends only on the number of CPUs, since the example provided uses just one GPU. The following theorem, whose proof is omitted, slightly improves the above result for the case of a single GPU. More interestingly, it expresses the lower bound to the approximation ratio of HEFT using both the number of CPUs and of GPUs.

**Theorem 1.** *For any $k \leq \sqrt{m}$, 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 model with independent tasks.*

The second approach is proposed by Kedad-Sidhoum et al. [11] 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. To present this integer linear program, let $x_j$ be a binary variable which is equal to 1 if a task $T_j$ 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 $\lambda$ the variable that corresponds to the maximum over all lower bounds used. Then, the Heterogeneous Linear Program (HLP) is as follows:

$$\text{minimize } \lambda$$

$$C_i + \overline{p_j}x_j + \underline{p_j}(1 - x_j) \leq C_j \qquad \forall T_j \in \mathcal{T}, T_i \in \Gamma^-(T_j) \qquad (1)$$

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

$$C_j \leq \lambda \qquad \forall T_j \in \mathcal{T} \qquad (3)$$

$$\max\{\frac{1}{m}\sum_{T_j \in \mathcal{T}} \overline{p_j}x_j, \frac{1}{k}\sum_{T_j \in \mathcal{T}} \underline{p_j}(1 - x_j)\} \leq \lambda \qquad (4)$$

$$x_j \in \{0, 1\} \qquad \forall T_j \in \mathcal{T} \qquad (5)$$

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

Constraints (1), (2) and (3) describe the critical path, while Constraint (4) imposes that the makespan cannot be smaller than the load on CPU and GPU sides. Note that the particular problem of deciding the allocation 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 $T_j$.

After relaxing the integrity Constraint (5), a fractional allocation can be found in polynomial time. To get an integral solution, the variables $x_j$ are rounded as follows: If $x_j \geq \frac{1}{2}$ then $T_j$ is assigned to the CPU side, otherwise 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 respecting the precedence relations and the decided allocation. We call this algorithm HLP-EST.

HLP-EST achieves an approximation ratio of 6 [11]. Surprisingly, the following theorem shows that this ratio is tight. In fact, the theorem implies an even

stronger result since the worst case example does not depend on the scheduling policy which will be applied after the allocation step.

**Theorem 2.** *Any scheduling policy which is applied after the allocation decisions taken by the rounding of an optimal fractional solution of the relaxed HLP leads to an approximation algorithm of ratio at least* $6 - O(\frac{1}{m})$.

*Proof (sketch).* 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 Table 1. The only precedence relations exist between tasks of $B_1$ and $B_2$: 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 $B_1$. There are no precedences between tasks of the same set.

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

**Table 1:** Tasks and their processing times for the input instance.

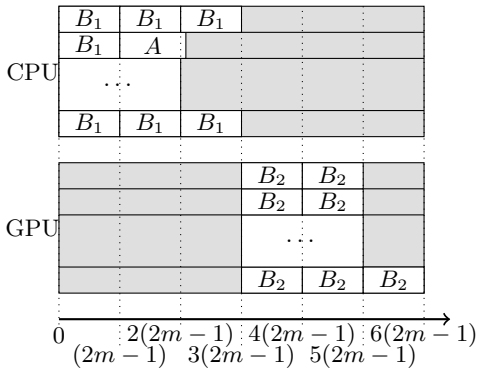



**Fig. 1:** The schedule created by the algorithm (the gray areas correspond to idle times).

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).

On the other hand, we can show that the following assignment is optimal for the relaxed HLP: given a small constant $\epsilon > 0$, set $x_A = 1$, $x_j = \frac{1}{2}$ for each $T_j \in B_1$, $x_j = \frac{1}{2} - \epsilon$ for each $T_j \in B_i$, and $\lambda = \frac{m(2m+1)}{m-1}$. Given this optimal fractional assignment, the algorithm 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, assuming that $m \geq 3$, there is only one meaningful family of schedules for the tasks in $B_1 \cup B_2$. An illustration of such a schedule is given in Fig. 1.

The makespan of the created schedule is equal to $6(2m-1)$, while the optimal fractional solution for the relaxed HLP has objective value $\frac{m(2m+1)}{m-1}$. Hence, the approximation ratio achieved for this instance is $6 - O\left(\frac{1}{m}\right)$ and the theorem follows. □

## 4 Algorithms

In this section we focus separately on each of the two phases, allocation and scheduling, and we propose algorithms for them.

### 4.1 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. 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, else 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, else assign it to the GPU side.
**R3** If $\overline{p_j} \leq \underline{p_j}$ then assign $T_j$ to the CPU side, else 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 a rule based on the structure of the actual schedule, in a similar way as in the 4-competitive algorithm proposed by Chen et al. [7] for the on-line problem with independent tasks. Our algorithm works also in the on-line setting.

To describe the new rule, we define $\tau^G$ to be the earliest time when at least one GPU is idle. Let also $R_j^G = \max\{\tau^G, \max_{i \in \Gamma^-(j)}\{C_i\}\}$ be the *ready time* of task $T_j$, i.e., the earliest time at which $T_j$ can be executed on a GPU. 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:** Otherwise apply R2.

This set of rules can be combined with a greedy List Scheduling policy that schedules each task as early as possible on the CPU or GPU side already decided by the rules. We call the algorithm obtained by this combination as ER-LS (Enhanced Rules - List Scheduling). Note that both the allocation policy based on rules and the List Scheduling policy can be applied in an on-line context, by considering the tasks one by one and taking irrevocable decisions for them.

**Theorem 3.** *ER-LS is a* $(4\sqrt{\frac{m}{k}})$*-competitive algorithm.*

### 4.2 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.

To do this, we define for each task $T_j$ 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-OLS.

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 Theorem 2. On the other hand, it is quite easy to see that HLP-EST and HLP-OLS have the same approximation ratio.

## 5 Experiments

In this section, we compare the performance of various scheduling algorithms by a simulation campaign using a benchmark composed of 6 parallel applications.

### 5.1 Benchmark

The benchmark is composed of five applications generated by *Chameleon*, a dense linear algebra software [1], and a more irregular application (*fork-join*) generated using *GGen*, a library for generating directed acyclic graphs [9].

The applications of *Chameleon*, named *getrf_nopiv*, *posv*, *potrs*, *potri* and *potrs*, are composed of multiple sequential basic tasks of linear algebra. Different number, denoted by *nb_blocks*, and sizes, denoted by *block_size*, of sub-matrices have been used for the applications; specifically, $nb\_blocks \in \{5, 10, 20\}$ and $block\_size \in \{64, 128, 320,\ 512, 768, 960\}$. The applications were executed with the runtime StarPU [4] on a Dual core Xeon E7 v2 machine with a total of 20 physical cores with hyper-threading of 3 GHz and 256 GB of RAM. This machine had 4 GPUs NVIDIA Tesla K20 with 4 GB of global memory, 200 GB/s of bandwidth and 2,496 cores divided in 13 multiprocessors.

The *fork-join* application corresponds to a real situation where the execution starts sequentially and then forks to *width* parallel tasks. The results are aggregated by performing a join operation, completing a phase. For our experiments,

we used $p \in \{2, 5, 10\}$ phases and $width \in \{100, 200, 300, 400, 500\}$. The running time of each task on CPU was computed using a Gaussian distribution with center $p$ and standard deviation $\frac{p}{4}$. We established various acceleration factors for the running time on GPU. In all configurations, there are five parallel tasks in each phase with an acceleration factor in $[0.1, 0.5]$ while the remaining tasks have an acceleration factor in $[0.5, 50]$. The data set and other information are available under Creative Commons Public License[3].

### 5.2 Environment and algorithms

We compare the performance, in terms of makespan, of HLP-EST and HLP-OLS with HEFT. We also compare in on-line mode, where tasks arrive over a list, the algorithm ER-LS with two greedy algorithms: GreedyOn which allocates a task on the processor type which has the smallest processing time for that task, and RandomOn which randomly assigns a task to the CPU or GPU side.

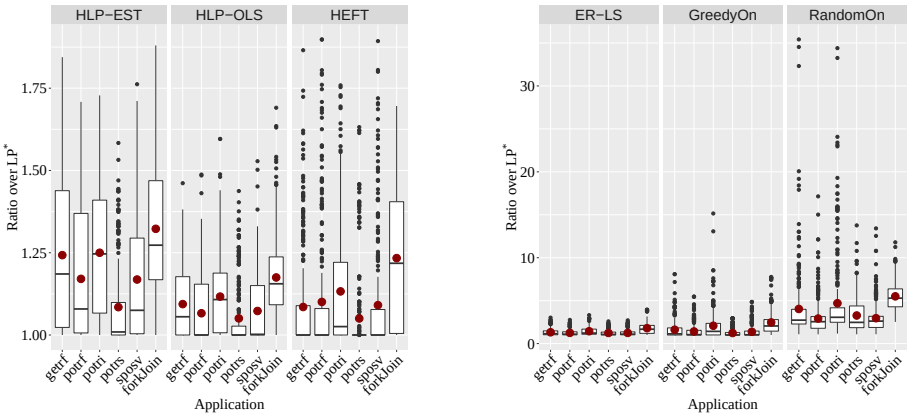
The algorithms are implemented in Python (v. 2.7.6). The command-line *glpsol* (v. 4.52) solver of the GLPK package is used for the linear program. The number of tasks of the six applications range from 30 to 5011. Moreover, we test different machine configurations, combining 16, 32, 64 or 128 CPUs with 2, 4, 8 or 16 GPUs. Each combination of application and machine configuration is executed only once since all algorithms, except for the random greedy algorithm, are deterministic. For each run, we store the optimal objective solution of the linear program, denoted by $LP^*$, and the makespan of the six algorithms.

### 5.3 Analysis of results

*Off-line algorithms.* To study the performance of the 3 off-line algorithms we computed the ratio between each makespan and $LP^*$, which corresponds to a good lower bound of the optimal makespan. Fig. 2a shows the ratio of each instance of application and configuration. 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.

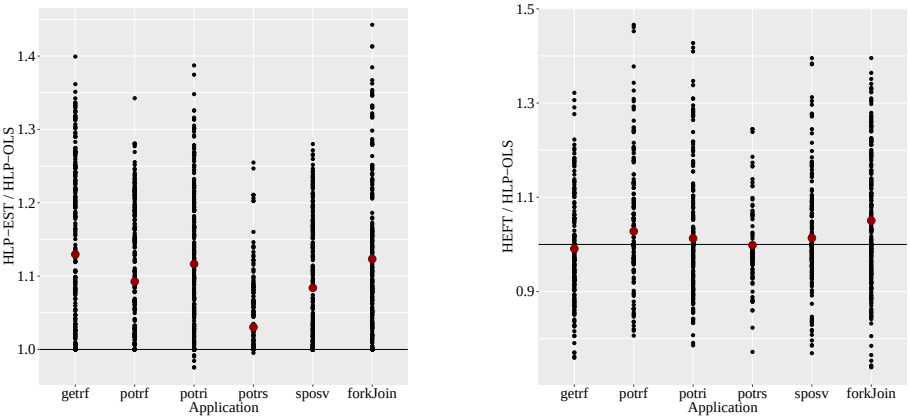
Fig. 3 compares more specifically the two HLP-based algorithms and the algorithms HLP-OLS and HEFT, respectively, by showing the ratio between the makespans of the two algorithms. We can see that HLP-OLS clearly outperforms HLP-EST, except for a few instances with the application *potri*, with an improvement close to 10% on average. We also 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.

[3] Hosted at: `https://github.com/marcosamaris/heterogeneous-SWF`




**(a)** Off-line algorithms. **(b)** On-line algorithms.

**Fig. 2:** Ratios over $LP^*$ for each instance, grouped by application.

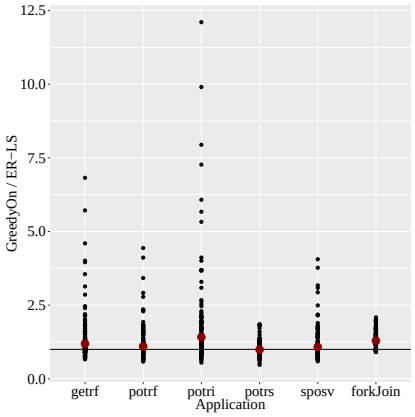



**(a)** **(b)**

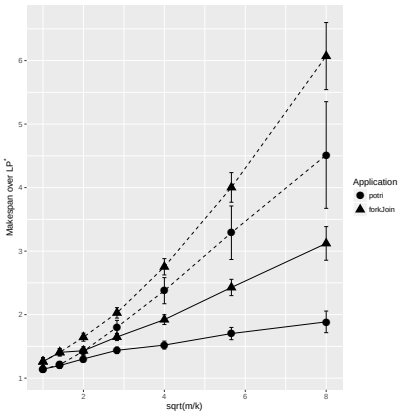
**Fig. 3:** Ratio between the makespans of HLP-EST and HLP-OLS (left) and HEFT and HLP-OLS (right) for each instance, grouped by application.

*On-line algorithms.* The ratios between the makespan of each of the 3 on-line algorithms and $LP^*$ are compared in Fig. 2b which shows that, except for a few number of instances, RandomOn is significantly outperformed by ER-LS and GreedyOn. Fig. 4a presents a more detailed comparison between GreedyOn and ER-LS, and shows that the ratio of their performance 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. For some instances, ER-LS can even perform up to 12.5 times better than GreedyOn. We also study the performance

of ER-LS and GreedyOn with respect to the theoretical upper bound given in Section 4.1. Fig. 4b shows the mean competitive ratio of ER-LS and GreedyOn along with the standard error in function of $\sqrt{\frac{m}{k}}$ associated to each instance. To simplify the lecture, we only present the applications *potri* and *fork-join*. The competitive ratio is smaller than $\sqrt{\frac{m}{k}}$ and far from the theoretical upper bound of $4\sqrt{\frac{m}{k}}$.




**(a)** Ratio between the makespans of GreedyOn and ER-LS.




**(b)** Competitive ratio of ER-LS (plain) and GreedyOn (dashed) in function of $\sqrt{\frac{m}{k}}$.

**Fig. 4:** Comparison of on-line algorithms.

## 6 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 efficient algorithms with worst-case performance guarantees. In the off-line case, motivated by new lower bounds on the performance of existing algorithms, we refined the scheduling phase of the best known approximation algorithm and we presented a new algorithm that preserves the approximation ratio and performs better in our experiments. In the on-line case, we presented a $O(\sqrt{\frac{m}{k}})$-competitive algorithm based on adequate rules, which can be considered as constant-factor since, practically, the ratio $\frac{m}{k}$ is bounded.

From the practical point of view, 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 good trade-off since it delivers a solution close to the optimal. We aim to implement it on a real run-time system (such as StarPU [4]) which currently uses HEFT on successive sets of independent tasks.

In this work we assumed that the communications between CPUs, GPUs and the shared memory are neglected. Our next step is to introduce communication costs in the algorithms, which should not be too hard in both integer program and greedy rules.

**Acknowledgment** This work was partially supported by FAPESP (São Paulo Research Foundation, grant #2012/23300-7) and ANR Moebus Project.

## References

1. Agullo et al. Poster: Matrices over runtime systems at exascale. In *SC Companion*, pages 1332–1332, 2012.
2. M. Amaris, D. Cordeiro, A. Goldman, and R. Y. de Camargo. A simple BSP-based model to predict execution time in GPU applications. In *HiPC*, 2015.
3. M. Amaris, G. Lucarelli, C. Mommessin, and D. Trystram. Generic algorithms for scheduling applications on hybrid multi-core machines. preprint, December 2016.
4. 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.*, 23(2):187–198, 2011.
5. R. Bleuse, S. Kedad-Sidhoum, F. Monna, G. Mounié, and D. Trystram. Scheduling independent tasks on multi-cores with GPU accelerators. *Concurr. Comput.: Pract. Exper.*, 27(6):1625–1638, 2015.
6. 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.*, 61(6):810–837, 2001.
7. L. Chen, D. Ye, and G. Zhang. Online scheduling of mixed CPU-GPU jobs. *International Journal of Foundations of Computer Science*, 25(06):745–761, 2014.
8. F. A. Chudak and D. B. Shmoys. Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds. *J. Algorithms*, 30:323–343, 1999.
9. D. Cordeiro, G. Mounié, S. Perarnau, D. Trystram, J.-M. Vincent, and F. Wagner. Random graph generation for scheduling simulations. In *ICST (SIMUTools)*, 2010.
10. R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal On Applied Mathematics*, 17(2):416–429, 1969.
11. S. Kedad-Sidhoum, F. Monna, and D. Trystram. Scheduling Tasks with Precedence Constraints on Hybrid Multi-core Machines. In *HCW*, pages 27–33, 2015.
12. Lee et al. Debunking the 100x GPU vs. CPU myth: An evaluation of throughput computing on CPU and GPU. *SIGARCH Comput. Archit. News*, 38:451–460, 2010.
13. O. Svensson. Hardness of precedence constrained scheduling on identical machines. *SIAM J. Comput.*, 40(5):1258–1274, 2011.
14. TOP-500-Supercomputer. [Web site `http://www.top500.org`].
15. H. Topcuoglu, S. Hariri, and Min-You Wu. Task scheduling algorithms for heterogeneous processors. In *HCW*, pages 3–14, 1999.