Low-Cost Approximation Algorithms for Scheduling Independent Tasks on Hybrid Platforms

Louis-Claude Canon[1,2(✉)], Loris Marchal[2], and Frédéric Vivien[2]

[1] FEMTO-ST Institute, Université de Bourgogne Franche-Comté, 16 route de Gray, 25 030 Besançon, France
louis-claude.canon@univ-fcomte.fr

[2] CNRS, Inria, ENS Lyon and University of Lyon, LIP Laboratory 46 allée d'Italie, 69 007 Lyon, France
loris.marchal@ens-lyon.fr, frederic.vivien@inria.fr

Abstract. Hybrid platforms embedding accelerators such as GPUs or Xeon Phis are increasingly used in computing. When scheduling tasks on such platforms, one has to take into account that a task execution time depends on the type of core used to execute it. We focus on the problem of minimizing the total completion time (or makespan) when scheduling independent tasks on two processor types, also known as the $(Pm, Pk)||C_{\max}$ problem. We propose BalancedEstimate and BalancedMakespan, two novel 2-approximation algorithms with low complexity. Their approximation ratio is both on par with the best approximation algorithms using dual approximation techniques (which are, thus, of high complexity) and significantly smaller than the approximation ratio of existing low-cost approximation algorithms. We compared both algorithms by simulations to existing strategies in different scenarios. These simulations showed that their performance is among the best ones in all cases.

1 Introduction

Modern computing platforms increasingly use specialized computation accelerators, such as GPUs or Xeon Phis: 86 of the supercomputers in the TOP500 list include such accelerators, while 3 of them include several accelerator types [17]. One of the most basic but also most fundamental scheduling step to efficiently use these hybrid platforms is to decide how to schedule independent tasks. The problem of minimizing the total completion time (or makespan) is well-studied in the case of homogeneous cores (problem $P||C_{\max}$ in Graham's notation [13]). Approximation algorithms have been proposed for completely unrelated processors ($R||C_{\max}$), such as the 2-approximation algorithms by Lenstra et al. [14] based on linear programming. Some specialized algorithms have been derived for the problem of scheduling two machine types ($(Pm, Pk)||C_{\max}$, where m and k are the number of machines of each type), which precisely corresponds to hybrid machines including only two types of cores, such as CPUs and GPUs

F.F. Rivera et al. (Eds.): Euro-Par 2017, LNCS 10417, pp. 232–244, 2017.
DOI: 10.1007/978-3-319-64203-1_17

(which corresponds to most hybrid platforms in the TOP500 list). Among the more recent results, we may cite the DADA [5] and DualHP [3] algorithms which both use dual approximation to obtain 2-approximations. Bleuse et al. [6] also propose a more expensive $(\frac{4}{3} + \frac{1}{3k} + \epsilon)$-approximation relying on dynamic programming and dual approximation with a time complexity $O(n^2m^2k^3)$ (with n being the number of tasks). PTAS have even been proposed for this problem [7,12]. However, the complexity of all these algorithms is large, which makes them unsuitable for efficiently scheduling tasks on high-throughput computing systems.

Our objective is to design an efficient scheduling algorithm for $(Pm, Pk)||C_{\max}$ whose complexity is as low as possible, so as to be included in modern runtime schedulers. Indeed with the widespread heterogeneity of computing platforms, many scientific applications now rely on runtime schedulers such OmpSs [16], XKaapi [5], or StarPU [2]. In this context, low complexity schedulers have recently been proposed. The closest approaches to our work in terms of cost, behavior, and guarantee are HeteroPrio [4], a $(2 + \sqrt{2})$-approximation algorithm when spoliation is permitted, and CLB2C [10], a 2-approximation algorithm in the case where every task processing time, on any resource, is smaller than the optimal makespan. A more detailed and complete analysis of the related work can be found in the companion research report [9].

In this paper, we propose a 2-approximation algorithm, named BalancedEstimate, which makes no assumption on the task processing times. Moreover, we propose BalancedMakespan which extends this algorithm with a more costly mechanism to select the final schedule, while keeping the same approximation ratio. We also present the simulations carried out to estimate in realistic scenarios the relative performance of the algorithms. Table 1 summarizes the comparison between our algorithms and existing solutions. Among many available high complexity solutions, we selected the ones whose running times were not prohibitive. The time complexity, when not available in the original articles, corresponds to our best guess, while performance are the range of the most frequent relative overheads of the obtained makespan with respect to a proposed lower bound that precisely estimates the minimum load on both processor types. In this table, BalancedEstimate and BalancedMakespan achieve both the best approximation ratio and the best performance in simulation.

Therefore, the main contributions of this paper are:

1. Two new approximation algorithms, BalancedEstimate and BalancedMakespan, which both achieve very good tradeoffs between runtime complexity, approximation ratios, and practical performance. The former has the smallest known complexity, improves the best known approximation ratio for low-complexity algorithms without constraints, and is on par with all competitors for practical performance, while the latter outperforms other strategies in most cases, at the cost of a small increase in the time complexity.
2. A new lower bound on the optimal makespan, a useful tool for assessing the actual performance of algorithms.

Table 1. Complexity and performance of the reference and new algorithms. The "performance" corresponds to the 2.5%–97.5% quantiles. The time complexity of HeteroPrio assumes an offline variant that needs to compute the earliest processor at each step. $A = \sum_i \max(c_i^1, c_i^2) - \max_i \min(c_i^1, c_i^2)$ is the range of possible horizon guesses for the dual approximations. (*: 3.42-approximation ratio for HeteroPrio when spoliation is permitted; **: 2-approximation ratio for CLB2C restricted to the cases when $\max(c_i^1, c_i^2) \leq \text{OPT}$)

Name	Time complexity	Approx. ratio	Performance
BalancedEstimate	$n \log(nmk)$	2	0.2–15%
BalancedMakespan	$n^2 \log(nmk)$	2	0.2–8%
HeteroPrio [4]	$n \log(n) + (n+m+k) \log(m+k)$	3.42^{**}	3.3–17%
CLB2C [10]	$n \log(nmk)$	2^{*}	3.6–37%
DualHP [4]	$n \log(nmkA)$	2	0.2–14%
DADA [5]	$n \log(mk) \log(A) + n \log(n)$	2	0.9–15%

3. A set of simulations including the state-of-the-art algorithms. They show that BalancedMakespan achieves the best makespan in more than 96% of the cases. Moreover, its makespan is always within 0.6% of the best makespan achieved by any of the tested algorithms.

The rest of the paper is organized as follows. The problem is formalized in Sect. 2 and the proposed algorithms are described in Sect. 3. Section 4 is devoted to a sketch of the proof of the approximation ratio. Section 5 presents a new lower bound for the makespan. Finally, we report the simulation results in Sect. 6 and conclude in Sect. 7.

2 Problem Formulation

A set of n tasks must be scheduled on a set of processors of two types containing m processors of type 1 and k processors of type 2. Let c_i^1 (resp. c_i^2) be the integer time needed to process task i on processors of type 1 (resp. of type 2). We indifferently refer to the c_i's as *processing times* or *costs*. The completion time of a processor of type u to which a set S of tasks is allocated is simply given by $\sum_{i \in S} c_i^u$. The objective is to allocate tasks to processors such that the maximum completion time, or makespan, is minimized.

3 Algorithm Description

We now move to the description of the first proposed approximation algorithm: BalancedEstimate. We start by introducing some notations/definitions that are used in the algorithm and in its proof. In the following μ represents an allocation of the tasks to the two processor types: $\mu(i) = 1$ (resp. $\mu(i) = 2$) means that task i is allocated to some processor of type 1 (resp. 2) in the allocation μ.

The precise allocation of tasks to processors will be detailed later. Note that in the algorithms, allocation μ is stored as an array and thus referred to as $\mu[i]$, which corresponds to $\mu(i)$ in the text. For a given allocation μ, we define $W^1(\mu)$ (resp. $W^2(\mu)$) as the average work of processors of type 1 (resp. 2):

$$W^1(\mu) = \frac{1}{m} \sum_{i:\mu(i)=1} c_i^1 \quad \text{and} \quad W^2(\mu) = \frac{1}{k} \sum_{i:\mu(i)=2} c_i^2.$$

We also define the maximum processing time $M^1(\mu)$ (resp. $M^2(\mu)$) of tasks allocated to processors of type 1 (resp. 2):

$$M^1(\mu) = \max_{i:\mu(i)=1} c_i^1 \quad \text{and} \quad M^2(\mu) = \max_{i:\mu(i)=2} c_i^2.$$

The proposed algorithm relies on the maximum of these four quantities to estimate the makespan of an allocation, as defined by the following *allocation cost estimate*:

$$\lambda(\mu) = \max(W^1(\mu), W^2(\mu), M^1(\mu), M^2(\mu)).$$

Finally, we use $\text{imax}(\mu)$, which is the index of the largest task allocated to a processor of type 1 but that would be more efficient on a processor of type 2:

$$\text{imax}(\mu) = \operatorname*{argmax}_{i:\mu(i)=1 \text{ and } c_i^1 > c_i^2} c_i^1.$$

We can now define a *dominating* task j as a task such that $j = \text{imax}(\mu)$ and $\lambda(\mu) = c^1_{\text{imax}(\mu)}$.

The algorithm works in two passes: it first computes two allocations with good allocation cost estimates (Algorithm 1) and then builds a complete schedule using the Largest Processing Time first (LPT) rule from these allocations (Algorithm 2).

The allocation phase (Algorithm 1) starts by putting each task on their most favorable processor type to obtain an initial allocation μ. Without loss of generality, we assume that processors of type 2 have the largest average work, otherwise we simply switch processor types. Then, tasks are moved from processors of type 2 to processors of type 1 to get a better load balancing. During this process, we carefully avoid task processing times from becoming arbitrarily long: whenever some dominating task appears, it is moved back to processors of type 2. The allocation phase produces two schedules: the one with the smallest cost estimate (μ_{best}) and the one corresponding to the iteration when the relative order of the average works is inversed (μ_{inv}). We define μ_i (resp. μ'_i) as the allocation before (resp. after) task i is allocated to processors of type 1 at iteration i on Line 10 ($\mu_{i_{\text{start}}} = \mu'_{i_{\text{start}}-1}$ is the initial allocation).

The scheduling phase (Algorithm 2) simply computes an LPT schedule for each processor type for the two previous allocations. The schedule with minimum makespan is selected as final result.

The time complexity of Algorithm 1 is $O(n \log(n))$ (computing the allocation cost estimate on Line 11 is the most costly operation). The time complexity of the subsequent scheduling phase (Algorithm 2) is $O(n \log(n) + n \log(m) + n \log(k))$.

Algorithm 1. Allocation Algorithm

Input : number m of processors of type 1; number k of processors of type 2
Input : number n of tasks; task durations c_i^l for $1 \leq i \leq n, 1 \leq l \leq 2$
Output: a set of allocations

for $i = 1 \dots n$ **do**
  if $c_i^1 < c_i^2$ **then** $\mu[i] \leftarrow 1$ **else** $\mu[i] \leftarrow 2$
if $W^1(\mu) > W^2(\mu)$ **then** switch processor types
$\mu_{\text{best}} \leftarrow \mu$
Sort tasks by non-decreasing c_i^1/c_i^2
$i_{\text{start}} = \min\{i : \mu[i] = 2\}$ `/* first task on a processor of type 2 */`
for $i = i_{start} \dots n$ **do**
  if $W^1(\mu) \leq W^2(\mu)$ **and** $W^1(\mu) + c_i^1/m > W^2(\mu) - c_i^2/k$ **then**
    $\mu_{\text{inv}} \leftarrow \mu$ `/* remember μ */`
  $\mu[i] \leftarrow 1$ `/* move a task (μ_i → μ'_i) */`
  if $\lambda(\mu) < \lambda(\mu_{best})$ **then**
    $\mu_{\text{best}} \leftarrow \mu$ `/* update best allocation so far */`
  if $\lambda(\mu) = c^1_{imax(\mu)}$ **then**
    $\mu[\text{imax}(\mu)] \leftarrow 2$ `/* move back a task (μ'_i → μ_{i+1}) */`
if μ_{inv} *is not defined* **then** $\mu_{\text{inv}} \leftarrow \mu$
return $(\mu_{\text{best}}, \mu_{\text{inv}})$

Theorem 1. BalancedEstimate *(Algorithm 2) is a 2-approximation for the makespan.*

We prove this result in the next section. Figure 1 provides an example showing that this 2-approximation ratio is tight. Both BalancedEstimate and BalancedMakespan build the schedule on the left, which has a makespan of $2k-2$ (initially they assign all the tasks on processors of type 2 and then move all the small tasks on processors of type 1). The makespan of the optimal schedule (on the right) is equal to k. The ratio is thus $2 - \frac{2}{k}$.

BalancedEstimate balances the average works on both processor types during the allocation while ensuring that no single task will degrade the

Algorithm 2. BalancedEstimate

Input : number m of processors of type 1; number k of processors of type 2
Input : number n of tasks; task durations c_i^l for $1 \leq i \leq n, 1 \leq l \leq 2$
Output: schedule of the tasks on the processors

Compute $(\mu_{\text{best}}, \mu_{\text{inv}})$ using Algorithm 1
foreach *Allocation* μ *in* (μ_{best}, μ_{inv}) **do**
  Schedule tasks $\{i : \mu[i] = 1\}$ on processors of type 1 using LPT
  Schedule tasks $\{i : \mu[i] = 2\}$ on processors of type 2 using LPT
return the schedule that minimizes the global makespan

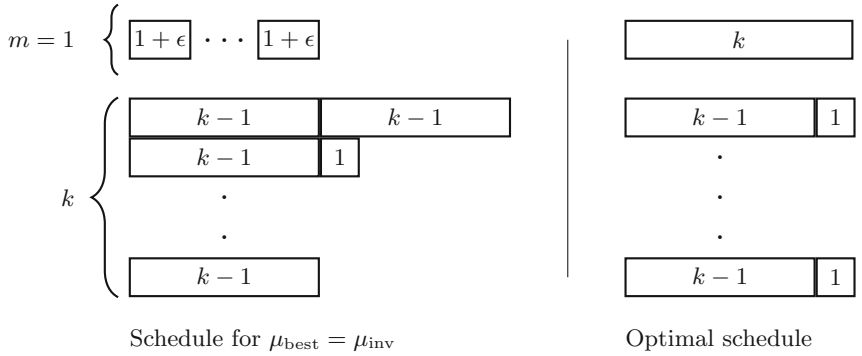

Fig. 1. Example with $m = 1$ processor of type 1, an arbitrary number $k > 1$ processors of type 2 and two types of tasks: k tasks with costs $c_i^1 = 1 + \epsilon$ (with $\epsilon < \frac{1}{k-1}$) and $c_i^2 = 1$, and $k + 1$ tasks with costs $c_i^1 = k$ and $c_i^2 = k - 1$.

makespan when scheduled. BalancedMakespan (Algorithm 3) extends this approach by computing the LPT schedule of each allocation (μ_i and μ'_i) considered by BalancedEstimate (including μ_{best} and μ_{inv}), and thus has the same approximation ratio. It uses the makespan instead of the allocation cost estimate to update μ_{best} and returns the schedule with the lowest makespan. Its time complexity is $O(n^2 \log(nmk))$ as it runs LPT $2n$ times. In Algorithm 3, $L(\mu)$ denotes the makespan of the schedule obtained using LPT on both processor types.

4 Approximation Ratio Proof

The proof that the previous scheduling algorithm produces a makespan at most twice the optimal one is quite long and technical (it includes seven lemmas, one corollary and the main proof requires the study of six different cases). For lack of space, we only present some of the key points of the proof in the present paper. The interested reader may find the whole detailed proof in the companion research report [9].

The proof starts by adding dummy tasks (with 0 cost on processors of type 2), to prove that μ_{inv} is always defined by Line 9: it corresponds to the last iteration where the relative order of the average works is inversed. We also prove that when Algorithm 1 completes, μ_{best} is the allocation with smallest cost estimate among all μ'_i's and μ_i's.

Then, our proof strongly relies on a new lower bound on the optimal makespan. Note that in the following property, μ is any allocation of the tasks to the processor types, not necessarily an allocation encountered by the algorithm.

Proposition 1. *Let μ be an allocation and $i_1 = \max\{i : \mu(i) = 1\}$ be the largest index of tasks that are on processors of type 1 (or 0 if there is none). Then,*

$$\min(W^1(\mu), W^2(\mu), \min_{\substack{1 \le i < i_1, \\ \mu(i)=2}} c_i^1) \le \text{OPT}, \tag{1}$$

Algorithm 3. BalancedMakespan

Input : number m of processors of type 1, number k of processors of type 2,
Input : number n of tasks, task durations c_i^l for $1 \leq i \leq n, 1 \leq l \leq 2$
Output: schedule of the tasks on the processors

for $i = 1 \dots n$ **do**
  if $c_i^1 < c_i^2$ **then** $\mu[i] \leftarrow 1$ **else** $\mu[i] \leftarrow 2$
if $W^1(\mu) > W^2(\mu)$ **then** switch processor types
$\mu_{\text{best}} \leftarrow \mu$
Sort tasks by non-decreasing c_i^1/c_i^2
$i_{\text{start}} = \min\{i : \mu[i] = 2\}$ `/* first task on processors of type 2 */`
for $i = i_{\text{start}} \dots n$ **do**
  $\mu[i] \leftarrow 1$ `/* move a task */`
  if $L(\mu) < L(\mu_{best})$ **then**
    $\mu_{\text{best}} \leftarrow \mu$ `/* update best allocation so far */`
  if $\lambda(\mu) = c^1_{imax(\mu)}$ **then**
    $\mu[\text{imax}(\mu)] \leftarrow 2$ `/* move back a task (` $\mu'_i \rightarrow \mu_{i+1}$ `) */`
  if $L(\mu) < L(\mu_{best})$ **then**
    $\mu_{\text{best}} \leftarrow \mu$ `/* update best allocation so far */`
return the schedule of tasks using LPT on both types of processors from μ_{best}

where OPT *is the makespan of an optimal schedule.*

The proof of this property proceeds as follows: we look at where the set of tasks $S = \{1 \leq i < i_1 : \mu(i) = 2\}$ are processed in an optimal allocation.

(i) Either one of those tasks is allocated to a processor of type 1, and then $\min_{i \in S} c_i^1$ is a lower bound on OPT;
(ii) Or all tasks of S are on processors of type 2. We then transform μ into the optimal allocation by exchanging tasks and, thanks to the fact that tasks are sorted by non-decreasing c_i^1/c_i^2, we can prove that not both W^1 and W^2 can increase simultaneously. As $\max(W^1(\text{OPT}), W^2(\text{OPT})) \leq \text{OPT}$, then $\min(W^1(\mu), W^2(\mu)) \leq \text{OPT}$.

We also need a classical result for list scheduling algorithms, summarized in the following lemma.

Lemma 1. *For a given set of tasks, any list scheduling algorithm (such as LPT) builds a schedule on p identical processors with a makespan lower than or equal to $W + (1 - \frac{1}{p})M$ where W is the average work and M is the maximum cost of any task.*

Algorithm 1 produces two allocations: μ_{best} and μ_{inv}, and the final schedule comes from one of them. The extensive proof considers a large number of special cases, but here we restrict to two cases, which we find the most significant: one case considers μ_{best} while the other one considers μ_{inv}.

Case 1. Assume that the cost estimate of μ_{best} is achieved on M^1 or M^2 ($\lambda(\mu_{\text{best}}) = \max(M^1(\mu_{\text{best}}), M^2(\mu_{\text{best}}))$) and that there is no dominating task in μ_{best} ($\lambda(\mu_{\text{best}}) > c^1_{\text{imax}(\mu_{\text{best}})}$). Then, we prove that $\lambda(\mu_{\text{best}}) \leq \text{OPT}$ by considering the two possible cases:

- The maximum defining $\lambda(\mu_{\text{best}})$ is achieved by $M^1(\mu_{\text{best}}) = \max_{j:\mu_{\text{best}}(j)=1} c^1_j$. Let j be a task achieving this maximum. Note that $c^1_j \leq c^2_j$ because otherwise we would have $M^1(\mu_{\text{best}}) = c^1_{\text{imax}(\mu_{\text{best}})}$, which is not possible because $\lambda(\mu_{\text{best}}) > c^1_{\text{imax}(\mu_{\text{best}})}$. Consider an optimal schedule: $\text{OPT} \geq \min(\text{c}^1_\text{j}, \text{c}^2_\text{j}) = \text{c}^1_\text{j} = \text{M}^1(\mu_{\text{best}})$ and thus $\lambda(\mu_{\text{best}}) \leq \text{OPT}$.
- The maximum defining $\lambda(\mu_{\text{best}})$ is achieved by $M^2(\mu_{\text{best}}) = \max_{j:\mu_{\text{best}}(j)=2} c^2_j$. Let j be a task achieving this maximum. This case is analogous to the previous one by remarking that j was already allocated to processors of type 2 in the initial allocation, and thus $c^1_j \geq c^2_j$.

As $\lambda(\mu_{\text{best}}) \leq \text{OPT}$, we know by Lemma 1 that LPT on μ_{best} gives a schedule with makespan at most 2OPT.

Case 2. This case reasons on μ_{inv}. By an abuse of notation we call *inv* the iteration at which μ_{inv} was defined at Line 9. We recall that after adding the task with index inv on processors of type 1, μ'_{inv} has an average work larger on processors of type 1 while μ_{inv} had an average work larger on processors of type 2. We apply Proposition 1 on μ_{inv} and μ'_{inv} and forget the cases where the minimum is achieved on a c^1_i in Eq. (1). This gives $W^1(\mu_{\text{inv}}) \leq \text{OPT}$ and $W^2(\mu'_{\text{inv}}) \leq \text{OPT}$. We also forget the case where the cost estimate of either μ_{inv} or μ'_{inv} is given by M^1 or M^2 (which can be treated as in Case 1).

We have

$$W^1(\mu'_{\text{inv}}) = W^1(\mu_{\text{inv}}) + \frac{c^1_{\text{inv}}}{m}.$$

and, since $W^1(\mu'_{\text{inv}}) \geq M^1(\mu'_{\text{inv}})$, $c^1_{\text{inv}} \leq W^1(\mu'_{\text{inv}})$. Those two relations bring

$$c^1_{\text{inv}} \leq \frac{W^1(\mu_{\text{inv}})}{1 - 1/m}.$$

Let M be the task with largest cost allocated on processors of type 1 in μ_{inv} ($c^1_M = M^1(\mu_{\text{inv}})$). We have

$$c^1_M \leq W^1(\mu'_{\text{inv}}) \leq W^1(\mu_{\text{inv}}) + \frac{c^1_{\text{inv}}}{m} \leq W^1(\mu_{\text{inv}}) + \frac{W^1(\mu_{\text{inv}})}{m-1} = \frac{m}{m-1} W^1(\mu_{\text{inv}}).$$

Consider the schedule built by Algorithm 2 on allocation μ_{inv}. On processors of type 1, we have $M^1(\mu_{\text{inv}}) = c^1_M$ bounded as above and the average work is $W^1(\mu_{\text{inv}}) \leq \text{OPT}$ (by assumption). Thanks to Lemma 1, we know that the makespan produced by LPT on this instance has a makespan bounded by:

$$\begin{aligned} C^1_{\max} &\leq W^1(\mu_{\text{inv}}) + \left(1 - \frac{1}{m}\right) M^1(\mu_{\text{inv}}) \leq W^1(\mu_{\text{inv}}) + \left(1 - \frac{1}{m}\right) c^1_M \\ &\leq W^1(\mu_{\text{inv}}) + \left(1 - \frac{1}{m}\right) \frac{m}{m-1} W^1(\mu_{\text{inv}}) \\ &\leq 2W^1(\mu_{\text{inv}}) \leq 2\text{OPT}. \end{aligned}$$

We now concentrate on processors of type 2. We know that

$$W^2(\mu_{\text{inv}}) = W^2(\mu'_{\text{inv}}) + \frac{c^2_{\text{inv}}}{k} \leq W^2(\mu'_{\text{inv}}) + \frac{\text{OPT}}{k},$$

The above inequality comes from the fact that $\text{OPT} \geq \min(\text{c}^1_{\text{inv}}, \text{c}^2_{\text{inv}}) = \text{c}^2_{\text{inv}}$ as task inv was on processors of type 2 in the initial allocation. For the same reason, $M^2(\mu_{\text{inv}}) \leq \text{OPT}$. Together with $W^2(\mu'_{\text{inv}}) \leq \text{OPT}$, we finally get

$$W^2(\mu_{\text{inv}}) \leq \left(1 + \frac{1}{k}\right) \text{OPT}.$$

Thanks to Lemma 1, we know that the makespan of Algorithm 2 on processors of type 2 of allocation μ_{inv} is bounded by

$$\begin{aligned} C^2_{\max} &\leq W^2(\mu_{\text{inv}}) + \left(1 - \frac{1}{k}\right) M^2(\mu_{\text{inv}}) \\ &\leq \left(1 + \frac{1}{k}\right) \text{OPT} + \left(1 - \frac{1}{\text{k}}\right) \text{OPT} \ \leq \ 2\text{OPT}. \end{aligned}$$

Thus, $\max(C^1_{\max}, C^2_{\max}) \leq 2\text{OPT}$ which yields the result for this case.

The whole proof with many other cases can be found in [9].

5 Lower Bound

We now present a new lower bound on the optimal makespan, which is then used as a reference in our simulations. Note that we could have used Proposition 1 to derive lower bounds, but this would require to first compute interesting allocations. On the contrary, we present here an analytical lower bound, which can be expressed using a simple formula, and which is finer than the previous one in the way it considers how the workload should be distributed.

The bound is obtained by considering the average work on all processors, as in the W/p bound for scheduling on identical machines. To obtain this bound, we consider the divisible load relaxation of the problem: we assume that all tasks can be split in an arbitrary number of subtasks which can be processed on different processors (possibly simultaneously). We are then able to show that the optimal load distribution is obtained when tasks with smaller c^1_i/c^2_i ratio are placed on processors of type 1, while the others are on processors of type 2, so that the load is well balanced. This may require to split one task, denoted by i in the theorem, among the two processor types.

Theorem 2. *Assume tasks are sorted so that* $c_i^1/c_i^2 \leq c_j^1/c_j^2$ *for* $i < j$*, and let* i *be the task such that*

$$\frac{1}{m}\sum_{j\leq i} c_j^1 \geq \frac{1}{k}\sum_{j>i} c_j^2 \quad \text{and} \quad \frac{1}{m}\sum_{j<i} c_j^1 \leq \frac{1}{k}\sum_{j\geq i} c_j^2.$$

Then, the following quantity is a lower bound on the optimal makespan:

$$\mathrm{LB} = \frac{c_i^2 \sum_{j<i} c_j^1 + c_i^1 \sum_{j>i} c_j^2 + c_i^1 c_i^2}{kc_i^1 + mc_i^2}.$$

As this bound only considers average load, it may be improved by also considering the maximum processing time over all tasks: $\max_i \min(c_i^1, c_i^2)$ is the equivalent of the $\max c_i$ lower bound for scheduling independent tasks on identical machines.

6 Simulations

In the context of linear algebra computations, hardware is typically composed of several CPU cores and a few GPU units to compute hundreds of tasks. The following simulations consider 300 tasks, 20 CPU cores, and 4 GPU units. Task processing times are randomly generated and follow a gamma distribution with expected value 15 for the CPUs and 1 for the GPUs. These values are inspired from the measures in [1,3]. Moreover, the gamma distribution has been advocated for modeling job runtimes [11,15]. This distribution is positive and it is possible to specify its expected value and standard deviation by adjusting its parameters. The Coefficient of Variation (CV[1]) of both types of processing times is either 0.2 (low) or 1 (high). Each combination of CV for the CPUs and the GPUs leads to 100 instances. For each instance, the set of processing times is given as input to all six algorithms and the obtained makespans are then divided by the lower bound given by Theorem 2. The algorithms are implemented in R and the related code, data and analysis are available in [8].

The studied algorithms are the reference algorithms DualHP, DADA, HeteroPrio and CLB2C, and our two new algorithms, BalancedEstimate and BalancedMakespan. HeteroPrio and CLB2C both start by sorting the tasks by their acceleration ratios. In HeteroPrio, each ready processor will then start the execution of the next best task. When all tasks are running, ready processors will steal a running task if this reduces its completion time. In CLB2C, at each iteration, the two tasks that are the best for each type of processors are considered and the one that can finish the soonest is scheduled.

Figure 2 depicts the ratios of the achieved makespans by the lower bound using boxplots in which the bold line is the median, the box shows the quartiles, the bars show the whiskers (1.5 times the interquartile range from the box) and additional points are outliers.

[1] The Coefficient of Variation is the ratio of thestandard deviation to the mean.

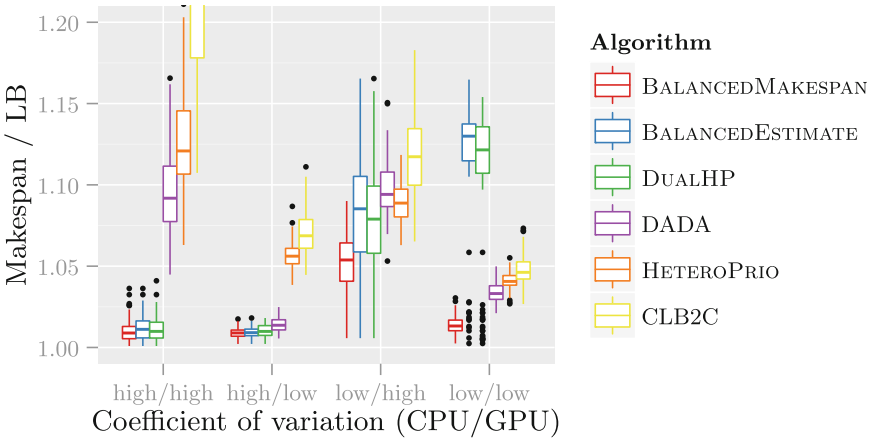

Fig. 2. Ratios of the makespan over a lower bound for 6 algorithms over 400 hundreds instances. For each instance, there are $n = 300$ tasks, $m = 20$ CPUs and $k = 4$ GPUs. The costs follow a gamma distribution with expected value 15 for the CPUs and 1 for the GPUs, while the coefficient of variation is either 0.2 (low) or 1 (high).

BalancedMakespan has the best median in all cases and is often below 2% from the lower bound except when the CPU CV is low and the GPU CV is high, for which the lower bound seems to be the furthest. This case is also the most realistic [1,3]. BalancedEstimate and DualHP have similar performance. It may be due to their similar mechanism: allocating the jobs to balance the average CPU and GPU works, and then scheduling the jobs in a second step. DADA, HeteroPrio and CLB2C, which all schedule the jobs incrementally, perform similarly for most of the cases. There are classes of problems for which CLB2C has median performance that is more than 20% away from the lower bound. No other algorithms achieve so low performance.

When the CPU CV is high, BalancedEstimate is close to the lower bound (the median is around 1%). In the opposite case, however, CPU costs are more homogeneous and the performance degrades. The LPT scheduling step of BalancedEstimate may schedule a last large task on a single CPU whereas it would have been better to allocate it to the GPUs. In comparison, BalancedMakespan, HeteroPrio, and CLB2C are not affected by this limitation because they build the schedule step by step and adjust the allocation depending on the actual finishing times.

Finally, we measured that BalancedMakespan provides the best makespan among the six tested algorithms in more than 96% of the cases. Moreover, the makespan is always within 0.6% of the best makespan achieved by the different algorithms. By contrast, the next two best algorithms in this regard, BalancedEstimate and DualHP, both provide the best makespan in more than 36% of the cases and their makespan is always within 16% of the best makespan.

7 Conclusion

With the recent rise in the popularity of hybrid platforms, efficiently scheduling tasks on multiple types of processors such as CPUs and GPUs has become critical. This paper presents BalancedEstimate, a new algorithm for the $(Pm, Pk)||C_{\max}$ problem. It balances the tasks from the most loaded processor type to the other type of processors. This algorithm is the first to achieve an approximation ratio of 2 in all cases with a low time complexity. We also propose BalancedMakespan, a more costly variant with the same guarantee. Among these two algorithms, simulations showed the latter outperforms competing algorithms in more than 96% of the cases, while the former is on par with a more costly dual approximation. The performance of the algorithms was assessed using a new lower bound on the optimal makespan.

Future developments will consist in evaluating the robustness of the algorithm against incertainties in the processing time estimates and implementing this approach in a real runtime system to see its benefits in practical situations. Furthermore, the model could be extended to fit more closely to realistic environments by considering precedence constraints, more than 2 types of processors and taking into account startup times for launching tasks on GPUs.

Acknowledgments. This work was supported by the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR). This material is also based upon research supported by the SOLHAR project operated by the French National Research Agency (ANR).

References

1. Agullo, E., Beaumont, O., Eyraud-Dubois, L., Kumar, S.: Are static schedules so bad? A case study on Cholesky factorization. In: 2016 IEEE International Parallel and Distributed Processing Symposium, pp. 1021–1030. IEEE (2016)
2. Augonnet, C., Thibault, S., Namyst, R., Wacrenier, P.A.: StarPU: a unified platform for task scheduling on heterogeneous multicore architectures. Concurr. Comput.: Pract. Exp. **23**(2), 187–198 (2011)
3. Beaumont, O., Cojean, T., Eyraud-Dubois, L., Guermouche, A., Kumar, S.: Scheduling of linear algebra kernels on multiple heterogeneous resources. In: International Conference on High Performance Computing, Data, and Analytics (HiPC) (2016)
4. Beaumont, O., Eyraud-Dubois, L., Kumar, S.: Approximation proofs of a fast and efficient list scheduling algorithm for task-based runtime systems on multicores and GPUs (2016, to appear in IEEEIPDPS 2017). https://hal.inria.fr/hal-01386174
5. Bleuse, R., Gautier, T., Lima, J.V.F., Mounié, G., Trystram, D.: Scheduling data flow program in XKaapi: a new affinity based algorithm for heterogeneous architectures. In: Silva, F., Dutra, I., Santos Costa, V. (eds.) Euro-Par 2014. LNCS, vol. 8632, pp. 560–571. Springer, Cham (2014). doi:10.1007/978-3-319-09873-9_47
6. Bleuse, R., Kedad-Sidhoum, S., Monna, F., Mounié, G., Trystram, D.: Scheduling independent tasks on multi-cores with GPU accelerators. Concurr. Comput.: Pract. Exp. **27**(6), 1625–1638 (2015)

7. Bonifaci, V., Wiese, A.: Scheduling unrelated machines of few different types. arXiv preprint arXiv:1205.0974 (2012)
8. Canon, L.C.: Code for low-cost approximation algorithms for scheduling independent tasks on hybrid platforms. https://doi.org/10.6084/m9.figshare.4674841.v1
9. Canon, L.C., Marchal, L., Vivien, F.: Low-cost approximation algorithm for scheduling independent tasks on hybrid platforms. Research report 9029, INRIA, February 2017. https://hal.inria.fr/INRIA/hal-01475884v1
10. Cheriere, N., Saule, E.: Considerations on distributed load balancing for fully heterogeneous machines: two particular cases. In: Proceedings of IEEE International Parallel and Distributed Processing Symposium Workshop (IPDPSW), pp. 6–16. IEEE (2015)
11. Feitelson, D.G.: Workload Modeling for Computer Systems Performance Evaluation, 1st edn. Cambridge University Press, New York (2015)
12. Gehrke, J.C., Jansen, K., Kraft, S.E.J., Schikowski, J.: A PTAS for scheduling unrelated machines of few different types. In: Freivalds, R.M., Engels, G., Catania, B. (eds.) SOFSEM 2016. LNCS, vol. 9587, pp. 290–301. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49192-8_24
13. Graham, R.L., Lawler, E.L., Lenstra, J.K., Kan, A.H.G.R.: Optimization and approximation in deterministic sequencing and scheduling: a survey. Ann. Discret. Math. **5**, 287–326 (1979)
14. Lenstra, J.K., Shmoys, D.B., Tardos, É.: Approximation algorithms for scheduling unrelated parallel machines. Math. Program. **46**(1–3), 259–271 (1990)
15. Lublin, U., Feitelson, D.G.: The workload on parallel supercomputers: modeling the characteristics of rigid jobs. J. Parallel Distrib. Comput. **63**(11), 1105–1122 (2003)
16. Sainz, F., Mateo, S., Beltran, V., Bosque, J.L., Martorell, X., Ayguadé, E.: Leveraging OmpSs to exploit hardware accelerators. In: IEEE International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), pp. 112–119 (2014)
17. TOP500 supercomputer site, list of November 2016. http://www.top500.org