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

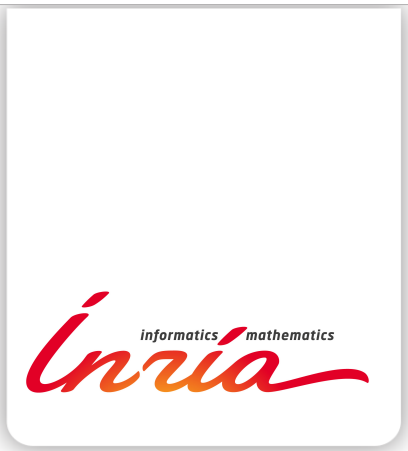
Loris Marchal, Louis-Claude Canon, Frédéric Vivien

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

Loris Marchal, Louis-Claude Canon, Frédéric Vivien

RESEARCH REPORT
N° 9029
February 2017
Project-Team ROMA

ISSN 0249-6399 ISRN INRIA/RR--9029--FR+ENG



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

Loris Marchal, Louis-Claude Canon, Frédéric Vivien

Project-Team ROMA

Research Report n° 9029 — February 2017 — 28 pages

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.

Key-words: Scheduling; independant tasks; hybrid platforms; approximation algorithms.

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Algorithmes d'approximation pour l'ordonnancement de tâches indépendantes sur des plates-formes hybrides

Résumé : Les plates-formes hybrides utilisant sur des accélérateurs tels que des GPUs ou des Xeon Phis sont de plus en plus utilisées pour le calcul. Lors de l'ordonnancement de tâches sur de telles plates-formes, il est nécessaire de prendre en compte les temps d'exécution des tâches en fonction du type de cœurs utilisé pour l'exécuter. Nous traitons le problème consistant à minimiser le temps total d'exécution (ou makespan) lors de l'ordonnancement de tâches indépendantes sur deux types de processeurs, problème noté $(Pm, Pk)||C_{\max}$. Nous proposons BalancedEstimate et BalancedMakespan, deux nouveaux algorithmes d'approximation de facteur 2 avec de faibles complexités. Leur facteur d'approximation est à la fois similaire à celui des meilleurs algorithmes d'approximation reposant sur des techniques d'approximation duale (et qui ont donc de grandes complexités) et significativement meilleur que ceux des algorithmes d'approximation existants de faible coût. Nous avons comparé ces deux algorithmes aux stratégies existantes avec des simulations dans différents scénarios. Ces simulations ont montré que leurs performances sont parmi les meilleures dans tous les cas.

Mots-clés : Ordonnancement; tâches indépendantes; plates-formes hybrides; algorithmes d'approximation.

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 [34]. 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). Approximation algorithms have been proposed for completely unrelated processors ($R||C_{\max}$), such as the 2-approximation algorithms by Lenstra et al. [26] 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 (which corresponds to most hybrid platforms in the TOP500 list). Among the more recent results, we may cite the DADA [10] and DualHP [7] algorithms which both use dual approximation to obtain 2-approximations. Bleuse et al. [12] 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 [13, 21]. 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 StarSS [28], XKaapi [10], or StarPU [5]. 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 [8], a $(2 + \sqrt{2})$-approximation algorithm, and CLB2C [17], a 2-approximation algorithm in the case where every task processing time, on any resource, is smaller than the optimal makespan.

In this report, 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 report 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 perfor-

Name	algo.	time complexity	approx. ratio	performance
BalancedEstimate	2	$n\log(nmk)$	2	0.3-15%
BalancedMakespan	3	$n^2\log(nmk)$	2	0.3-8%
CLB2C [17]	5	$n\log(nmk)$	2*	3.2-33%
HeteroPrio [8]	6	$(n\log(n)+m+k)\times\log(m+k)$	3.42	12-40%
DualHP [8]	7	$n\log(nmkA)$	2**	0.3-15%
DADA [10]	9	$n\log(mk)\log(A)+n\log(n)$	2**	0.8-14%

Table 1: Complexity and performance of the reference and new algorithms. The "performance" corresponds to the 2.5%–97.5% quantiles. $A=\sum_i \max(c_i^1,c_i^2)-\max_i\min(c_i^1,c_i^2)$ is the range of possible horizon guesses for the dual approximations.
*: 2-approximation ratio for CLB2C restricted to the cases when $\max(c_i^1,c_i^2)\leq \mathrm{OPT}$
**: see Theorems 4 and 5

mance of algorithms.

3. A set of simulations including the state-of-the-art algorithms. They show that BalancedMakespan achieves the best makespan in more than 95% of the cases. Moreover, its makespan is always within 0.5% of the best makespan achieved by any of the tested algorithms.

The rest of the report is organized as follows. The problem is formalized in Section 2 and the proposed algorithms are described in Section 3. Section 4 is devoted to the proof of the approximation ratio. Section 5 presents a new lower bound for the makespan. We report the simulation results in Section 6. Finally, we present the related work in Section 7 and conclude in Section 8.

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 processing time of task i on processors of type 1 (resp. of type 2). 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. The most frequent notations are summarized in Appendix A.

3 Algorithm Description

3.1 BalancedEstimate

We now move to the description of the first proposed approximation algorithm. 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 i as a task such that $i = \text{imax}(\mu)$ 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 $O(n \log(n) + n \log(m) + n \log(k))$.

3.2 BalancedMakespan

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

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 $i = i_{\text{start}} \dots n$ **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 (`$\mu_i \rightarrow \mu'_i$`) */`
  if $\lambda(\mu) < \lambda(\mu_{\text{best}})$ **then**
    $\mu_{\text{best}} \leftarrow \mu$ `/* update best allocation so far */`
  if $\lambda(\mu) = c^1_{\text{imax}(\mu)}$ **then**
    $\mu[\text{imax}(\mu)] \leftarrow 2$ `/* move back a task (`$\mu'_i \rightarrow \mu_{i+1}$`) */`
if μ_{inv} *is not defined* **then** $\mu_{\text{inv}} \leftarrow \mu$
return $(\mu_{\text{best}}, \mu_{\text{inv}})$

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 task on the processors

Compute $(\mu_{\text{best}}, \mu_{\text{inv}})$ using Algorithm 1
foreach *Allocation* μ *in* $(\mu_{\text{best}}, \mu_{\text{inv}})$ **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

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.

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 task on the processors

1 **for** $i = 1 \dots n$ **do**
2 **if** $c_i^1 < c_i^2$ **then** $\mu[i] \leftarrow 1$ **else** $\mu[i] \leftarrow 2$
3 **if** $W^1(\mu) > W^2(\mu)$ **then** switch processor types
4 $\mu_{\text{best}} \leftarrow \mu$
5 Sort tasks by non-decreasing c_i^1/c_i^2
6 $i_{\text{start}} = \min\{i : \mu[i] = 2\}$ `/* first task on processors of type 2 */`
7 **for** $i = i_{\text{start}} \dots n$ **do**
8 $\mu[i] \leftarrow 1$ `/* move a task */`
9 **if** $L(\mu) < L(\mu_{\text{best}})$ **then**
10 $\mu_{\text{best}} \leftarrow \mu$ `/* update best allocation so far */`
11 **if** $\lambda(\mu) = c^1_{\text{imax}(\mu)}$ **then**
12 $\mu[\text{imax}(\mu)] \leftarrow 2$ `/* move back a task (` $\mu'_i \rightarrow \mu_{i+1}$ `) */`
13 **if** $L(\mu) < L(\mu_{\text{best}})$ **then**
14 $\mu_{\text{best}} \leftarrow \mu$ `/* update best allocation so far */`
15 **return** the schedule of tasks using LPT on both types of processors from μ_{best}

4 Approximation Ratio Proof

4.1 Restricting Termination Cases

For the sake of the proof, we slightly modify the algorithm by introducing new tasks in the instance. This simplifies the analysis of the algorithm termination and does not change the allocation returned by the algorithm. Assume, without loss of generality, that the test at Line 3 was false. Furthermore, we assume there is no task with a null cost in the instance. We discard such tasks from the analysis because they may be allocated to any processor on which its cost is null without any impact on the algorithm or the makespan. Then, we add $m\left(\sum_{i=1}^{n} c_i^2\right) + 1$ dummy tasks with cost 1 on processors of type 1 and cost 0 on processors of type 2. The proof of the approximation ratio of Algorithm 1 relies on this modified version that returns the same allocation (by Lemma 1).

Lemma 1. *Adding dummy tasks does not change the allocation computed by Algorithm 1.*

Proof. Let n (resp. n^*) be the number of tasks for the original (resp. the extended) instance (let μ be the allocation for the original instance and μ^* the allocation for the extended one). Starting at iteration $n+1$, each dummy task is allocated to processors of type 1 (they are considered last because their ratios c_i^1/c_i^2 are the largest). At each of these allocations, W_μ^{*2} remains constant while W_μ^{*1} increases. Therefore, $\lambda(\mu^*)$ may only increase and the best allocation ${\mu_{\text{best}}}^*$ found

so far is unchanged. Then, let l be the last iteration where the test at Line 9 is true. If $l \leq n$, then $\mu^*_{\text{inv}|1..n} = \mu_{\text{inv}}$. If $l > n$, then $\mu^*_{\text{inv}|1..n} = \mu^*_{n+1|1..n} = \mu_{n+1}$ because the allocation of the tasks that have a non-zero cost on processors of type 2 is the same for each μ^*_i such that $n+1 \leq i < n'+1$. □

The following lemma states that there is no dominating task at the beginning of an iteration, and that μ_{best} is always the best allocation encountered so far, i.e., the one with the smallest λ value.

Lemma 2. *At the end of any iteration i of Algorithm 1, we have $\lambda(\mu_i) > c^1_{\text{imax}(\mu_i)}$ and $\lambda(\mu_{\text{best}}) = \min_{1\leq j\leq i}(\min(\lambda(\mu_j), \lambda(\mu'_j)))$.*

Proof. We prove both properties by induction on i.

Induction basis: at iteration $i = i_{\text{start}}$, each task is allocated to the processor type on which its execution is fastest. Therefore, $\text{imax}(\mu_{i_{\text{start}}})$ is undefined and the first property is true. Moreover, μ_{best} is the initial allocation and it is the only one that was considered by the algorithm until step i_{start}.

Induction step: we assume the lemma holds true for $i \geq i_{\text{start}}$ and thus, $\lambda(\mu_i) > c^1_{\text{imax}(\mu_i)}$ and $\lambda(\mu_{\text{best}}) = \min_{1\leq j\leq i}(\lambda(\mu_j))$. Let us prove it holds true for μ_{i+1}, i.e., that $\lambda(\mu_{i+1}) > c^1_{\text{imax}(\mu_{i+1})}$ and $\lambda(\mu_{\text{best}}) = \min_{1\leq j\leq i+1}(\lambda(\mu_j))$. If the condition on Line 13 is false for iteration i, then $\mu_{i+1} = \mu'_i$ and the results trivially holds true.

Let us consider the other case (the condition on Line 13 is true). Thus, $\lambda(\mu'_i) = c^1_{\text{imax}(\mu'_i)}$ and task $\text{imax}(\mu'_i)$ is moved back from processors of type 1 to processors of type 2. In the remainder of this proof we denote by $\mu_{\text{best},i}$ the value of μ_{best} at the *end* of iteration i.

We distinguish two cases for the definition of $\lambda(\mu_i)$:

Case 1: $\lambda(\mu_i) > W^2(\mu_i)$, and thus $\lambda(\mu_i) = \max(M^1(\mu_i), M^2(\mu_i), W^1(\mu_i))$. On the one hand, note that $\max(M^1(\mu_i), M^2(\mu_i)) = \max_k c_k^{\mu_i(k)} = \max(\max_{k\neq i} c_k^{\mu_i(k)}, c_i^2)$. After moving task i to processors of type 1 (Line 10), this quantity becomes $\max(M^1(\mu'_i), M^2(\mu'_i)) = \max(\max_{k\neq i} c_k^{\mu'_i(k)}, c_i^1) = \max(\max_{k\neq i} c_k^{\mu_i(k)}, c_i^1)$ because μ_i and μ'_i differ only for task i. Moreover, $c_i^1 \geq c_i^2$ because task i is on its best processor in μ_i. Hence, $\max(M^1(\mu'_i), M^2(\mu'_i)) \geq \max(M^1(\mu_i), M^2(\mu_i))$. On the other hand, after the allocation on Line 10, $W^1(\mu'_i) = W^1(\mu_i) + \frac{c_i^1}{m} \geq W^1(\mu_i)$. Therefore, $\lambda(\mu'_i) \geq \lambda(\mu_i) > c^1_{\text{imax}(\mu_i)}$ by induction hypothesis. By assumption, $\lambda(\mu'_i) = c^1_{\text{imax}(\mu'_i)}$ and thus $c^1_{\text{imax}(\mu'_i)} > c^1_{\text{imax}(\mu_i)}$. We deduce that only the newly added task i can dominate on processors of type 1: $\text{imax}(\mu'_i) = i$. Hence, task i is moved back at Line 14 of iteration i. Thus, $\mu_{i+1} = \mu_i$ and there is no dominating task when iteration $i+1$ starts ($\lambda(\mu_{i+1}) > c^1_{\text{imax}(\mu_{i+1})}$) because there is no dominating task in μ_i by induction hypothesis. It also follows that $\lambda(\mu_{\text{best},i+1}) = \min_{1\leq j\leq i+1}(\lambda(\mu_j))$ because $\lambda(\mu_{\text{best},i}) = \min_{1\leq j\leq i}(\lambda(\mu_j))$ by induction hypothesis and because $\mu_{i+1} = \mu_i$.

Case 2: $\lambda(\mu_i) = W^2(\mu_i)$. Note that we have $W^2(\mu_{i+1}) = W^2(\mu'_i) + \frac{c^2_{\text{imax}(\mu'_i)}}{k} = W^2(\mu_i) + \frac{c^2_{\text{imax}(\mu'_i)} - c_i^2}{k}$. Moreover, $\text{imax}(\mu'_i) \leq i$, which implies that $c^1_{\text{imax}(\mu'_i)} \geq c_i^1$ (we select the largest task on processors of type 1) and that $\frac{c_i^1}{c_i^2} \geq \frac{c^1_{\text{imax}(\mu'_i)}}{c^2_{\text{imax}(\mu'_i)}}$ (tasks are sorted). Thus, $c^2_{\text{imax}(\mu'_i)} \geq c_i^2$, which implies that $W^2(\mu_{i+1}) \geq W^2(\mu_i)$. Therefore, $\lambda(\mu_{i+1}) \geq \lambda(\mu_i)$ and $\lambda(\mu_{\text{best},i+1}) = \min_{1\leq j\leq i+1}(\lambda(\mu_j))$. Moreover, $\lambda(\mu_i) > c^1_{\text{imax}(\mu_i)}$ by induction hypothesis. To go from μ_i to μ_{i+1}, we added task i to processors of type 1, and then removed the longest task on these processors ($\text{imax}(\mu'_i)$). Either i is this longest task and the allocation is unchanged, or the longest task is a different one that was already in μ_i. In both

cases, the size of the longest task on processors of type 1 has not increased from μ_i to μ_{i+1}, i.e., $c^1_{\text{imax}(\mu_i)} \geq c^1_{\text{imax}(\mu_{i+1})}$. Thus, $\lambda(\mu_{i+1}) > c^1_{\text{imax}(\mu_{i+1})}$.

We conclude by remarking that $\lambda(\mu_{\text{best}}) = \min_{1\leq j\leq i}(\lambda(\mu'_j))$ is straightforward because of Line 11. □

Thanks to the introduction of new tasks, we are able to prove that at some point, the average work gets larger on processors of type 1, as stated in the following lemma.

Lemma 3. *Consider the allocation μ_{inv} computed by Algorithm 1 when adding dummy tasks. The following properties hold true:*
(i) $W^1(\mu_{\text{inv}}) \leq W^2(\mu_{\text{inv}})$*;*
(ii) $W^1(\mu'_{\text{inv}}) > W^2(\mu'_{\text{inv}})$*;*
(iii) There is no dominating task in either μ_{inv} and μ'_{inv}.

Proof. By construction of μ_{inv} by the algorithm, if μ_{inv} was defined at Line 9, it was defined by the last iteration respecting properties (i) and (ii). We now prove that in the problem instance with dummy tasks, μ_{inv} is indeed defined by Line 9.

At the first iteration, i_{start}, Line 3 ensures that $W^1(\mu_{i_{\text{start}}}) \leq W^2(\mu_{i_{\text{start}}})$. Starting with iteration $n+1$, the algorithm will move the dummy tasks to processors of type 1 one after the other and these tasks are too small to be moved back [1]. We know that $W^1(\mu_{n+1}) \geq 0$. Between iteration $n+1$ and n^*, $m\left(\sum_{i=1}^n c_i^2\right)+1$ tasks with cost 1 are added to processors of type 1. Thus, $W^1(\mu_{n^*+1}) \geq \sum_{i=1}^n c_i^2 + \frac{1}{m}$. Moreover, $W^2(\mu_{n'+1}) \leq \frac{\sum_{i=1}^n c_i^2}{k}$ and thus $W^1(\mu_{n'+1}) > W^2(\mu_{n'+1})$.

To go from $\mu_{i_{\text{start}}}$ to $\mu_{n'+1}$, there is necessarily an inversion in the average works. Therefore, there is one iteration i for which $W^1(\mu_i) \leq W^2(\mu_i)$ and $W^1(\mu_{i+1}) > W^2(\mu_{i+1})$. This implies that $W^1(\mu'_i) > W^2(\mu'_i)$ because Line 14 may only move a task from processors of type 1 to processors of type 2. Therefore, Line 9 defines μ_{inv} at least once. Let l be the last iteration at which Line 9 defines μ_{inv}.

Lemma 2 guarantees that $\lambda(\mu_l) > c^1_{\text{imax}(\mu_l)}$, thus there is no dominating task in μ_l. Let us consider by contradiction that there is a dominating task in μ'_l. In this case, some task $\text{imax}(\mu'_l)$ is moved back to processors of type 2. Let us compute the average work at the beginning of the next iteration:

- $W^1(\mu_{l+1}) = W^1(\mu_l) + c_l^1 - c^1_{\text{imax}(\mu'_l)}$. Thanks to the definition of imax(), we have $c_l^1 \leq c^1_{\text{imax}(\mu'_l)}$ and thus $W^1(\mu_{l+1}) \leq W^1(\mu_l)$.
- $W^2(\mu_{l+1}) = W^2(\mu_l) - c_l^2 + c_j^2$. Let $j = \text{imax}(\mu'_l)$. As $j \leq l$, $\frac{c_j^1}{c_j^2} \leq \frac{c_l^1}{c_l^2}$, and thus $c_j^2 \geq c_l^2$, which leads to $W^2(\mu_{l+1}) \geq W^2(\mu_l)$.

Thus, the average works at step $l+1$ are not inversed anymore, which necessarily leads to another inversion later on. This contradicts the hypothesis that l is the last iteration of Line 9 that defines μ_{inv}. Thus, there is no dominating task in μ'_l and thus in μ'_{inv}. □

4.2 Ratio Proof

Lemma 4. *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,*

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

[1] To be sure that this property is satisfied, we start by multiplying by 2 the cost of any task on any processor type. Obviously, this does not impact the behaviour of Algorithm 1. Then, because we have assumed all execution times to be non-null integers (see Section), we know that for $i > n$, $c_i^1 = 1 < 2 \leq \lambda(\mu)$ whatever the allocation μ. Therefore we cannot have $\lambda(\mu) = c^1_{\text{imax}(\mu)}$ with $c^1_{\text{imax}(\mu)} = 1$.

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

Proof. Let us consider an optimal schedule of makespan OPT. In this schedule, either one of the tasks $\{i : 1 \leq i < i_1, \mu(i) = 2\}$ is on a processor of type 1, in which case $\min_{1\leq i\leq i_1,\ \mu(i)=2} c_i^1$ is a lower bound on OPT, or all of these tasks are on processors of type 2. In the latter case, we show that $\min(W^1(\mu), W^2(\mu))$ is smaller than or equal to OPT by contradiction. Let us assume that S^1 is the set of the indices of the tasks that are on processors of type 1 in μ but on processors of type 2 in the optimal schedule (we define S^2 analogously). In order to transform μ into the optimal allocation, we have to migrate tasks of S_1 on processors of type 2 and tasks of S_2 on processors of type 1. This leads to a decrease of $W^1(\mu)$ by $\sum_{i\in S^1} c_i^1 - \sum_{i\in S^2} c_i^1$ and of $W^2(\mu)$ by $\sum_{i\in S^2} c_i^2 - \sum_{i\in S^1} c_i^2$. If both these amounts were strictly positive, then we would have $\sum_{i\in S^2} c_i^1 < \sum_{i\in S^1} c_i^1$ (1) and $\sum_{i\in S^1} c_i^2 < \sum_{i\in S^2} c_i^2$ (2). Moreover, $\sum_{i\in S^1} c_i^1 = \sum_{i\in S^1} \frac{c_i^1}{c_i^2} c_i^2 \leq \left(\max_{i\in S^1} \frac{c_i^1}{c_i^2}\right) \sum_{i\in S^1} c_i^2$ (3) and, similarly, $\sum_{i\in S^2} c_i^2 \leq \frac{1}{\min_{i\in S^2} c_i^1/c_i^2} \sum_{i\in S^2} c_i^1$ (4). Thus, we would have:

$$\sum_{i\in S^2} c_i^1 \overset{(1)}{<} \sum_{i\in S^1} c_i^1 \overset{(3)}{\leq} \left(\max_{i\in S^1} \frac{c_i^1}{c_i^2}\right) \sum_{i\in S^1} c_i^2 \overset{(2)}{<} \left(\max_{i\in S^1} \frac{c_i^1}{c_i^2}\right) \sum_{i\in S^2} c_i^2 \overset{(4)}{\leq} \frac{\max_{i\in S^1} \frac{c_i^1}{c_i^2}}{\min_{i\in S^2} \frac{c_i^1}{c_i^2}} \sum_{i\in S^2} c_i^1.$$

However, as we will show below, $\max(S^1) < \min(S^2)$. Thus, $\max_{i\in S^1} \frac{c_i^1}{c_i^2} \leq \min_{i\in S^2} \frac{c_i^1}{c_i^2}$ because tasks are sorted by non-decreasing c_i^1/c_i^2. Therefore, the amounts by which $W^1(\mu)$ and $W^2(\mu)$ can decrease cannot be both positive. Hence, there is no optimal schedule with a makespan lower than $\min(W^1(\mu), W^2(\mu))$.

We now prove that $\max(S^1) < \min(S^2)$. By definition, i_1 is the largest index of a task that is allocated to a processor of type 1 in μ. By definition, S^1 is the set of the indices of the tasks that are on processors of type 1 in μ but on processors of type 2 in the optimal schedule. Therefore, $\max(S^1) \leq i_1$. By definition, S^2 is the set of the indices of the tasks that are on processors of type 2 in μ but on processors of type 1 in the optimal schedule. Therefore $i_1 \notin S^2$ because $\mu(i_1) = 1$ by definition. Furthermore, we have assumed that all the tasks i for $1 \leq i < i_1$ are on processors of type 2 in the schedule OPT. Therefore $\min(S^2) > i_1$ ands thus, $\max(S^1) < \min(S^2)$. □

We continue with two simple lemmas that help simplify the proof of Theorem 1 in some special cases.

Lemma 5. *Let μ be an allocation at some iteration of Algorithm 1 (μ_i or μ'_i) such that $\lambda(\mu) > c^1_{\mathrm{imax}(\mu)}$ and $\lambda(\mu) = \max(M^1(\mu), M^2(\mu))$. Then, $\lambda(\mu) \leq$ OPT.*

Proof. Let us consider such an allocation μ with $\lambda(\mu) = \max(M^1(\mu), M^2(\mu))$. We consider the two possible cases:

Case 1: The maximum is achieved by $M^1(\mu) = \max_{j:\mu(j)=1} c_j^1$. Let j be a task achieving this maximum. Note that $c_j^1 \leq c_j^2$ because otherwise we would have $M^1(\mu) = c^1_{\mathrm{imax}(\mu)}$, which is not possible because $\lambda(\mu) > c^1_{\mathrm{imax}(\mu)}$. Consider an optimal schedule: OPT $\geq \min(c_j^1, c_j^2) = c_j^1$ and thus $\lambda(\mu) \leq$ OPT.

Case 2: The maximum is achieved by $M^2(\mu) = \max_{j:\mu(j)=2} c_j^2$. Let j be a task achieving this maximum. j is thus assigned to a processor of type 2. Then, either j was initially assigned on a processor of type 2 (and was never moved to a processor of type 1) and thus $c_j^1 \geq c_j^2$ or it was assigned to a processor of type 2 by Line 13 and, in that case also, $c_j^1 \geq c_j^2$. Therefore, OPT $\geq \max\{c_j^1, c_j^2\} \geq c_j^2 = M^2(\mu) = \lambda(\mu)$.

□

Lemma 6. *Consider an iteration i of the algorithm and a task j, $i_{\text{start}} \leq j < i$, such that $\mu_i(j) = 2$ or $\mu'_i(j) = 2$. Then, we have $\lambda(\mu_{\text{best}}) \leq c_j^1$.*

Proof. As task j has an index larger than or equal to i_{start} but smaller than i, it means that it was originally on processors of type 2 and moved to processors of type 1 (in allocation μ'_j). As it is now allocated to processors of type 2 in μ_i or μ'_i, it also means that it was moved back to processors of type 1 by Line 14 at some iteration k. We consider the allocation μ'_k, where j was still on processors of type 1 and right before it was moved back ($\text{imax}(\mu'_k) = j$). For this allocation, we have $\lambda(\mu'_k) = c_j^1$ and thus $\lambda(\mu_{\text{best}}) \leq \lambda(\mu'_k) \leq c_j^1$ by Lemma 2. □

The following lemma is a classical result in scheduling, used in Graham's proof of the $(2-1/m)$ approximation ratio for list scheduling algorithms. We include it for the sake of completion.

Lemma 7. *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.*

Proof. Let us build a schedule from the set of tasks with a list heuristic, which never keeps a processor idle if a task is available. Let t be the earliest date at which not all processors are used. No task will start its execution after t because we use a list heuristic. Let c be the cost of a task that finishes its execution the latest. Then, $mks \leq t + c$ where mks is the makespan. Moreover, the amount of work that is performed (i.e., pW) is greater than or equal to $tp + c$ (all processors are used until t and at least one is used until $t + c$): $pW \geq tp + c$. Thus, $t \leq W - \frac{c}{p}$. Therefore, $mks \leq W - \frac{c}{p} + c = W + (1 - \frac{1}{p})c$. We conclude that $mks \leq W + (1 - \frac{1}{p})M$ because M is the maximum cost. □

A simple corollary:

Corollary 1. *Assume that the allocation μ_{best} output by Algorithm 1 is such that $\lambda(\mu_{\text{best}}) \leq$* OPT*, than Algorithm 2 produces a schedule whose makespan is at most twice* OPT.

Here comes the final result:

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

Proof. We consider the variant of Algorithm 1 that adds dummy tasks. By Lemma 1, it returns the same allocation. By Lemma 3, there exists an iteration j for which $W^1(\mu_{\text{inv}}) \leq W^2(\mu_{\text{inv}})$, $W^1(\mu'_{\text{inv}}) > W^2(\mu'_{\text{inv}})$ and there is no dominating task in μ_{inv}. By an abuse of notation, in this proof we will write about iteration inv rather than about iteration j. We apply Lemma 4 on allocation μ'_{inv}. Since by definition of inv, $W^2(\mu'_{\text{inv}}) < W^1(\mu'_{\text{inv}})$, Equation (1) applied to inv translates into:

$$\min(W^2(\mu'_{\text{inv}}), \min_{\substack{1 \leq i < \text{inv}, \\ \mu'_{\text{inv}}(i)=2}} c_i^1) \leq \text{OPT}. \tag{2}$$

In this equation, we have replaced i_1 by inv because the maximum index i_1 of tasks that are on processors of type 1 at iteration inv is inv itself.

We now distinguish two cases depending on how the minimum is achieved.

Case 1: The minimum in Equation (2) is achieved on W^2 ($W^2(\mu'_{\text{inv}}) \leq \min_{1 \leq i < \text{inv}, \mu'_{\text{inv}}(i)=2} c_i^1$), and thus Equation 1 gives

$$W^2(\mu'_{\text{inv}}) \leq \text{OPT}.$$

We recall that

$$\lambda(\mu_{\text{inv}}) = \max(W^1(\mu_{\text{inv}}), W^2(\mu_{\text{inv}}), M^1(\mu_{\text{inv}}), M^2(\mu_{\text{inv}})).$$

First, we know that $W^1(\mu_{\text{inv}}) \leq W^2(\mu_{\text{inv}})$ (by definition of inv). We also know that there is no dominating task in μ_{inv} (by Lemma 3), and thus $\lambda(\mu_{\text{inv}}) > c^1_{\text{imax}(\mu_{\text{inv}})}$. Again, we distinguish two cases depending on how the maximum in the definition of $\lambda(\mu_{\text{inv}})$ is achieved:

(i) $\lambda(\mu_{\text{inv}}) = \max(M^1(\mu_{\text{inv}}), M^2(\mu_{\text{inv}}))$. Then, we apply Lemma 5 which proves that $\lambda(\mu_{\text{inv}}) \leq \text{OPT}$. Thanks to Corollary 1, we know that from this μ_{inv}, Algorithm 2 produces a schedule whose makespan is at most twice OPT and thus verifies the conclusion of this theorem.

(ii) $\lambda(\mu_{\text{inv}}) = W^2(\mu_{\text{inv}})$. We first apply Lemma 4 to μ_{inv}. We get

$$\min(W^1(\mu_{\text{inv}}), W^2(\mu_{\text{inv}}), \min_{\substack{1\leq i<\text{inv},\\ \mu_{\text{inv}}(i)=2}} c^1_i) \leq \text{OPT}. \tag{3}$$

As $\lambda(\mu_{\text{inv}}) = W^2(\mu_{\text{inv}})$, we have $W^1(\mu_{\text{inv}}) \leq W^2(\mu_{\text{inv}})$. We further distinguish two cases depending on when the minimum is achieved in Equation 3.

(a) The minimum is achieved on $W^1(\mu_{\text{inv}})$ and thus

$$W^1(\mu_{\text{inv}}) \leq \text{OPT}. \tag{4}$$

Let M be the task with largest cost allocated on processors of type 1 in μ_{inv}:

$$c^1_M = \max_{\mu_{\text{inv}}(j)=1} c^1_j = M^1(\mu_{\text{inv}}).$$

We again distinguish two (final) subcases:

(I) $c^1_M \leq W^1(\mu'_{\text{inv}})$. We have

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

and

$$c^1_{\text{inv}} \leq W^1(\mu'_{\text{inv}}).$$

Those two relations bring

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

which refines the bound on c^1_M:

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

Consider the schedule built by Algorithm 2 on allocation μ_{inv}. On processors of type 1, the largest task has a size $M^1(\mu_{\text{inv}}) = c^1_M$, with C^1_M being bounded as shown above, and the average work is $W^1(\mu_{\text{inv}})$, with $W^1(\mu_{\text{inv}}) \leq \text{OPT}$ as seen in Equation (4). Thanks to Lemma 7, 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

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

because $\text{OPT} \geq \min(c^1_{\text{inv}}, c^2_{\text{inv}}) = c^2_{\text{inv}}$ as task inv was on processors of type 2 in the initial allocation. A task which is on processors of type 2 either was always allocated on processors of type 2 or mas moved to this type of processors by Line 14. In both cases its execution time on processors of type 2 is no greater than on processors of type 1 (in the second case by definition of imax()). Therefore, $M^2(\mu_{\text{inv}}) \leq \text{OPT}$. Together with $W^2(\mu'_{\text{inv}}) \leq \text{OPT}$ (the assumption defining this case, case 1), we finally get

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

Thanks to Lemma 7, we know that the makespan of Algorithm 2 on processors of type 2 of allocation μ_{inv} is bounded by

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

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

(II) $c^1_M > W^1(\mu'_{\text{inv}})$. By the definition of inv, $W^1(\mu'_{\text{inv}}) > W^2(\mu'_{\text{inv}})$ and thus $\lambda(\mu'_{\text{inv}}) = \max(M^1(\mu'_{\text{inv}}), M^2(\mu'_{\text{inv}}))$. As there is no dominating task in μ_{inv} (by Lemma 3), Lemma 5 proves that $\lambda(\mu'_{\text{inv}}) \leq \text{OPT}$ and thus, using Lemma 2, $\lambda(\mu_{\text{best}}) \leq \lambda(\mu'_{\text{inv}}) \leq \text{OPT}$. Thanks to Corollary 1, we know that from this μ_{best}, Algorithm 2 produces a schedule whose makespan is at most twice OPT and thus verifies the conclusion of this theorem.

(b) The minimum in Equation 3 is achieved on the last term. Thus, there exists a cost c^1_j for some task j such that

$$c^1_j = \min_{\substack{1 \leq i < \text{inv}, \\ \mu_{\text{inv}}(i)=2}} c^1_i \leq \text{OPT}.$$

We apply Lemma 6 to this task j for iteration inv, which gives us $\lambda(\mu_{\text{best}}) \leq c^1_j \leq \text{OPT}$. As $\lambda(\mu_{\text{best}})$ may only decrease as Algorithm 1 continues its execution, this also holds at the end of the execution. Thanks to Corollary 1, we

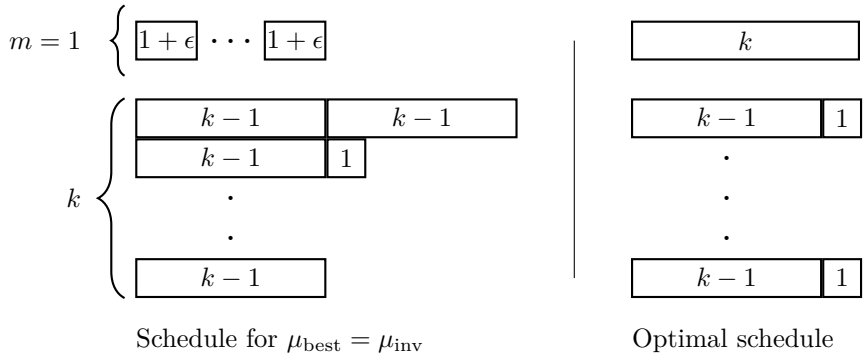


Figure 1: Example with $m = 1$ processor of type 1, an arbitrary number $k > 1$ processors of type 2 and two types of tasks: k tasks with costs $c_i^1 = 1+\epsilon$ (with $\epsilon < \frac{1}{k-1}$) and $c_i^2 = 1$, and $k+1$ tasks with costs $c_i^1 = k$ and $c_i^2 = k-1$. Both BalancedEstimate and BalancedMakespan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

know that from this μ_{best}, Algorithm 2 produces a schedule whose makespan is at most twice OPT and thus verifies the conclusion of this theorem.

Case 2: The minimum in Equation (2) is not achieved on a $W^2(\mu'_{\text{inv}})$ but on a cost c_j^1 for some task j:

$$c_j^1 = \min_{\substack{1 \le i < \text{inv}, \\ \mu'_{\text{inv}}(i)=2}} c_i^1 \le \text{OPT}.$$

We apply Lemma 6 to this task j for iteration inv, which gives us $\lambda(\mu_{\text{best}}) \le c_j^1 \le \text{OPT}$. Thanks to Corollary 1, we know that from this μ_{best}, Algorithm 2 produces a schedule whose makespan is at most twice OPT and thus verifies the conclusion of this theorem. □

Figure 1 provides an example showing that this 2-approximation ratio is tight.

Theorem 2. BalancedMakespan *(Algorithm 3) is a 2-approximation.*

Proof. This algorithm considers the same allocations as Algorithm 2, including μ_{best} and μ_{inv}, because the loop structure and condition for each allocation is the same. By Theorem 1, the minimum makespan obtained when applying LPT on these μ_{best} and μ_{inv} is at most 2OPT. BalancedMakespan ensures that the returned schedule has the lowest makespan among all the schedules obtained from the allocations. Therefore, this makespan is also at most 2OPT. □

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 Lemma 4 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_i^1/c_i^2 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 3. *Assume tasks are sorted so that* $\frac{c_i^1}{c_i^2} \leq \frac{c_j^1}{c_j^2}$ *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:

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

Proof. We consider a relaxation of the problem where tasks are perfectly divisible: each task i may be split in an arbitrary number of smaller sub-tasks $i_1, \ldots, i_m$ (with possibly non-integer processing times). Formally, the processing time of i_j on machine p is given by $c_{i_j}^p = \alpha_j c_i^p$ and we have $\sum_j \alpha_j = 1$. The sub-tasks of a given task may be processed simultaneously by different processors, possibly of different types. We denote by OPT the optimal makespan of the original problem, OPT' the optimal makespan of the divisible variant and S'_{OPT} an optimal schedule for this variant. Since a schedule for the original problem is also a valid schedule for the divisible variant, we have $\mathrm{OPT}' \leq \mathrm{OPT}$.

We now prove that $\mathrm{OPT}' = \mathrm{LB}$. Let i be the task defined as in the theorem. We consider the schedule S' for the divisible problem such that:

- Tasks j with $j < i$ are allocated to processors of type 1;
- Tasks j with $j > i$ are allocated to processors of type 2;
- Task i is split into two parts, i_1, i_2 with

$$\alpha_1 = \frac{k\sum_{j<i} c_j^1 - m\sum_{j>i} c_j^2}{k c_i^1 + m c_i^2},$$

and i_1 is allocated to processors of type 1 while i_2 is placed on processors of type 2.

As tasks are divisible, the computation time of all processors of type 1 is equal to their total load divided by m, and similarly for processors of type 2. It is easy to verify that in this schedule, all processors of both types complete their tasks at time LB. We now prove that $\mathrm{LB} = \mathrm{OPT}'$. If S'_{OPT} and S' differ, we progressively transform S'_{OPT} into S' as follows.

Consider j_1, the task with smallest index whose fraction f_1 allocated to processors of type 1 is larger in S'_{OPT} than in S' (note that $j_1 \geq i$). Similarly, we define j_2 as the task with larger index whose fraction f_2 allocated to processors of type 2 is larger in S'_{OPT} than is S' (and thus $j_2 \leq i$). Note that not both j_1 and j_2 can be equal to i, and thus $j_2 < j_1$. We consider the difference, f'_1 and f'_2, between the fraction of j_1 and j_2 in S'_{OPT} and in S':

$$f'_1 = \begin{cases} f_1 - \alpha_1 & \text{if } j_1 = i \\ f_1 & \text{otherwise} \end{cases} \quad \text{and} \quad f'_2 = \begin{cases} f_2 - (1-\alpha_1) & \text{if } j_2 = i \\ f_2 & \text{otherwise} \end{cases}$$

We now consider S'_{new} the schedule obtained from S'_{OPT} by exchanging a fraction

$$\epsilon_1 = \frac{\min\left(f'_1(c^1_{j_1} + c^2_{j_1}), f'_2(c^1_{j_2} + c^2_{j_2})\right)}{c^1_{j_1} + c^2_{j_1}} \leq f'_1$$

of j_1 from processors of type 1 to processors of type 2 and a fraction

$$\epsilon_2 = \frac{\min\left(f'_1(c^1_{j_1} + c^2_{j_1}), f'_2(c^1_{j_2} + c^2_{j_2})\right)}{c^1_{j_2} + c^2_{j_2}} \leq f'_2$$

of j_2 from processors of type 2 to processors of type 2. The difference in total work for processors of type 1 between S'_{OPT} and S'_{new} is given by:

$$\begin{aligned}
\Delta W_1 &= \epsilon_2 c^1_{j_2} - \epsilon_1 c^1_{j_1} \\
&= \min\left(f'_1(c^1_{j_1} + c^2_{j_1}), f'_2(c^1_{j_2} + c^2_{j_2})\right)\left(\frac{c^1_{j_2}}{c^1_{j_2} + c^2_{j_2}} - \frac{c^1_{j_1}}{c^1_{j_1} + c^2_{j_1}}\right) \\
&= \min\left(f'_1(c^1_{j_1} + c^2_{j_1}), f'_2(c^1_{j_2} + c^2_{j_2})\right)\frac{c^1_{j_2}c^2_{j_1} - c^1_{j_1}c^2_{j_2}}{(c^1_{j_1} + c^2_{j_1}) \times (c^1_{j_2} + c^2_{j_2})}
\end{aligned}$$

As $j_1 < j_2$ and because tasks are sorted by non-decreasing c^1_j/c^2_j, we have $c^1_{j_1}/c^2_{j_1} \leq c^1_{j_2}/c^2_{j_2}$ and thus $\Delta W_1 \leq 0$. Similarly, the difference in total wok for processors of type 2 is given by:

$$\begin{aligned}
\Delta W_2 &= \epsilon_1 c^2_{j_1} - \epsilon_2 c^2_{j_2} \\
&= \min\left(f'_1(c^1_{j_1} + c^2_{j_1}), f'_2(c^1_{j_2} + c^2_{j_2})\right)\left(\frac{c^2_{j_1}}{c^1_{j_1} + c^2_{j_1}} - \frac{c^2_{j_2}}{c^1_{j_2} + c^2_{j_2}}\right) \\
&= \min\left(f'_1(c^1_{j_1} + c^2_{j_1}), f'_2(c^1_{j_2} + c^2_{j_2})\right)\frac{c^2_{j_1}c^1_{j_2} - c^2_{j_2}c^1_{j_1}}{(c^1_{j_1} + c^2_{j_1}) \times (c^1_{j_2} + c^2_{j_2})} \\
\Delta W_2 &\leq 0.
\end{aligned}$$

Thus, this transformation does not increase the makespan of the schedule. Moreover, given the choice of ϵ_1 and ϵ_2, the fraction of j_1 or j_2 is the same in S'_{new} as in S' after these exchanges. We can repeat this process until S'_{new} and S' perfectly match without increasing the makespan, which proves that $\text{OPT}' = \text{LB}$ and thus $\text{LB} \leq \text{OPT}$. □

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

This bound can be computed in $O(n)$ steps with Algorithm 4.

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 [2, 7]. The following simulations consider 300 tasks, 20 CPU cores, and 4 GPU units. All the costs follow a gamma distribution with expected value 15 for the CPUs and 1 for the GPUs. The Coefficient of Variation (CV[2])

[2]The Coefficient of Variation is the ratio of the standard deviation to the mean.

Algorithm 4: Simple Lower Bound

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 lower bound

```
for i = 1...n do
    μ[i] ← 2
Sort tasks by non-decreasing c_i^1/c_i^2
for i = 1...n do
    μ[i] ← 1
    if W^1(μ) − c_i^1/m ≤ W^2(μ) + c_i^2/k and W^1(μ) > W^2(μ) then
        return (c_i^2/k W^1(μ) + c_i^1/m W^2(μ)) / (c_i^1/m + c_i^2/k)
```

of both types of costs 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 makespans obtained with six algorithms are divided by the lower bound given by Theorem 3.

The studied algorithms are the reference algorithms DualHP, DADA, CLB2C and HeteroPrio, and our two new algorithms, BalancedEstimate and BalancedMakespan (Algorithms 7, 9, 5, 6, 3 and 2). CLB2C and HeteroPrio both start by sorting the tasks by their acceleration ratios. 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. 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.

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.

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 [2, 7]. 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 and CLB2C, which both schedule the jobs incrementally, perform similarly for most of the cases. HeteroPrio is consistently the worst algorithm in this setting at more than 10% from the lower bound. There are classes of problems for which both CLB2C and HeteroPrio have 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, CLB2C, and HeteroPrio 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 95% of the cases. Moreover, the makespan is always within 0.5% 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

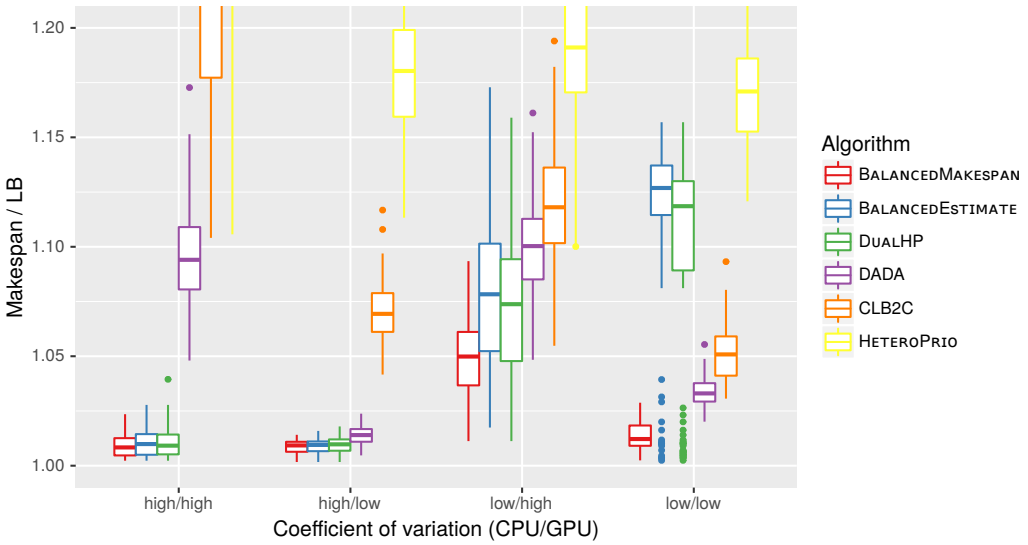


Figure 2: Ratios of the makespan over a lower bound for 6 algorithms over 400 hundreds instances. For each instance, there are $n = 300$ tasks, $m = 20$ CPUs and $k = 4$ GPUs. The costs follow a gamma distribution with expected value 15 for the CPUs and 1 for the GPUs, while the coefficient of variation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

in more than 39% of the cases and their makespan is always within 16% of the best makespan.

7 Related Work

The closest approaches in terms of cost, behavior and guarantee are HeteroPrio [8], a $(2+\sqrt{2})$-approximation, and CLB2C [17], a 2-approximation assuming that no single is greater than the optimal makespan. They 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 another task if it reduces its finish time. In CLB2C, the two tasks that are the best for each type of processors are considered and the one that can finish the soonest is scheduled.

In the online case, tasks arrive at random times and need to be executed immediately by a processor or added to a waiting queue for future execution. The competitive ratios of LG and MG [23] are $2 + \frac{m-1}{k}$ and $4 - \frac{2}{m}$, respectively (assuming that $m > k$). Al5 [15, 16] is a 3.85-competitive algorithm and its simplification, Al4, is a 4-competitive algorithm. These strategies are the fastest in the literature because they do not sort the tasks.

More costly schemes have also been proposed. Bleuse et al. [12] propose a $(\frac{4}{3} + \frac{1}{3k} + \epsilon)$-approximation with a time complexity $O(n^2m^2k^3)$. It relies on dynamic programming and a dual approximation [22]. The dynamic programming was further simplified into a 2-approximation, DP, with a time complexity $O(n^2k)$. DADA [10] and DualHP [7] are two simpler dual approximations, which have an approximation ratio of 2.

Algorithms have also been proposed for a similar problem in which each task has a different cost on each machine ($R||C_{\max}$). Lenstra et al. [26] propose a 2-approximation algorithm based on a linear program with a rounding technique and show that no algorithm with a ratio better

Name/authors	ratio	restriction	running time cost	year
HeteroPrio [8]	$\begin{cases} 2+\sqrt{2} \approx 3.42 \\ \frac{3+\sqrt{5}}{2} \approx 2.62 \\ \frac{1+\sqrt{5}}{2} \approx 1.62 \end{cases}$	$\begin{cases} \text{none} \\ \min(m,k)=1 \\ m=k=1 \end{cases}$		2016
CLB2C [17]	2	$\max c_i^l \leq \text{OPT}$		2015
LG [23]	$2+\frac{\max(m,k)-1}{\min(m,k)}$	online		2003
MG(1,1) [23]	$4-\frac{2}{\max(m,k)}$	online		2003
Al5 [16]	3.85	online		2014
Al4 [16]	4	online		2014
Bleuse et al. [12]	$\frac{4}{3}+\frac{1}{3k}+\epsilon$		$O(n^2m^2k^3)$	2015
DP [12]	2		$O(n^2k)$	2015
DADA [10]	2			2014
DualHP [8]	2			2016
Lenstra et al. [26]	2			1990
Shmoys et al. [32]	2			1993
Shchepin et al. [31]	$2-\frac{1}{m+k}$			2005
Gairing et al. [19]	2		$O((m+k)^3\times \log(m+k)\times n\log(n\max c_i^l))$	2007
Bonifaci et al. [13]	PTAS			2012
Gehrke et al. [21]	PTAS		$O(n)+(m+k)^{O(1/\epsilon^2)}\times\left(\frac{\log(m+k)}{\epsilon}\right)^{O(1)}$	2016

Table 2: State-of-the-art guaranteed algorithms. Restrictions and complexities are given using the notation of the $(Pm|Pk)||C_{\max}$ problem. ϵ is the error parameter.

than $\frac{3}{2}$ exists. This algorithm is later generalized to a bicriteria version of the problem with the same guarantee [32]. Using the same rounding technique, this ratio is improved to $2-\frac{1}{m+k}$ [31]. Gairing et al. propose a 2-approximation using a purely combinatorial approach.

A Polynomial Time Approximation Scheme has been proposed for the problem $(Pm, Pk)||C_{\max}$ [13]. Gehrke et al. [21] improve this PTAS for the problem $Pm_1, \ldots, Pm_K||C_{\max}$ (when there is a constant number K of processor types) and provide a comprehensive list of related PTAS, in particular for $R||C_{\max}$. The cost of these approaches is however prohibitive.

The problem has also been studied in practical situations using specific runtime environments as XKaapi [10] and StarPU [2]. The study with StarPU has been focused on a single application, the Cholesky factorization [1], and later extended to more than 2 types of processors [7].

Variations of the problem have also been theoretically studied: with moldable tasks [11], with precedence constraints [24, 25] and with preemption [9]. Finally, constraints and particularities of real-time systems (recurrent tasks, deadlines and migration) have also been considered [3, 4, 6, 18, 27, 29, 30, 33] for systems like ARM big.LITTLE hardware.

Table 2 summarizes the related guaranteed algorithms for this problem.

8 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 95% 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 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 and more than 2 types of processors.

References

[1] Emmanuel Agullo, Olivier Beaumont, Lionel Eyraud-Dubois, Julien Herrmann, Suraj Kumar, Loris Marchal, and Samuel Thibault. Bridging the gap between performance and bounds of cholesky factorization on heterogeneous platforms. In *Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International*, pages 34–45. IEEE, 2015.

[2] Emmanuel Agullo, Olivier Beaumont, Lionel Eyraud-Dubois, and Suraj Kumar. Are Static Schedules so Bad? A Case Study on Cholesky Factorization. In *Parallel and Distributed Processing Symposium, 2016 IEEE International*, pages 1021–1030. IEEE, 2016.

[3] Björn Andersson and Gurulingesh Raravi. Provably good task assignment for two-type heterogeneous multiprocessors using cutting planes. *ACM Transactions on Embedded Computing Systems (TECS)*, 13(5s):160, 2014.

[4] Björn Andersson and Gurulingesh Raravi. Scheduling constrained-deadline parallel tasks on two-type heterogeneous multiprocessors. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, pages 247–256. ACM, 2016.

[5] Cédric Augonnet, Samuel Thibault, Raymond Namyst, and Pierre-André Wacrenier. Starpu: a unified platform for task scheduling on heterogeneous multicore architectures. *Concurrency and Computation: Practice and Experience*, 23(2):187–198, 2011.

[6] Sanjoy K Baruah. Task partitioning upon heterogeneous multiprocessor platforms. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 536–543. Citeseer, 2004.

[7] Olivier Beaumont, Terry Cojean, Lionel Eyraud-Dubois, Abdou Guermouche, and Suraj Kumar. Scheduling of linear algebra kernels on multiple heterogeneous resources. In *International Conference on High Performance Computing, Data, and Analytics (HiPC)*, 2016.

[8] Olivier Beaumont, Lionel Eyraud-Dubois, and Suraj Kumar. Approximation Proofs of a Fast and Efficient List Scheduling Algorithm for Task-Based Runtime Systems on Multicores and GPUs. To appear in IEEE IPDPS 2017, 2016.

[9] Jacek Błażewicz, Safia Kedad-Sidhoum, Florence Monna, Grégory Mounié, and Denis Trystram. A study of scheduling problems with preemptions on multi-core computers with gpu accelerators. *Discrete Applied Mathematics*, 196:72–82, 2015.

[10] Raphaël Bleuse, Thierry Gautier, João VF Lima, Grégory Mounié, and Denis Trystram. Scheduling data flow program in XKaapi: A new affinity based algorithm for heterogeneous architectures. In *European Conference on Parallel Processing*, pages 560–571. Springer, 2014.

[11] Raphaël Bleuse, Sascha Hunold, Safia Kedad-Sidhoum, Florence Monna, Grégory Mounié, and Denis Trystram. Scheduling Independent Moldable Tasks on Multi-Cores with GPUs. Research Report RR-8850, Inria Grenoble Rhône-Alpes, Université de Grenoble, January 2016.

[12] Raphael Bleuse, Safia Kedad-Sidhoum, Florence Monna, Grégory Mounié, and Denis Trystram. Scheduling independent tasks on multi-cores with gpu accelerators. *Concurrency and Computation: Practice and Experience*, 27(6):1625–1638, 2015.

[13] Vincenzo Bonifaci and Andreas Wiese. Scheduling unrelated machines of few different types. *arXiv preprint arXiv:1205.0974*, 2012.

[14] Louis-Claude Canon. Code for Low-Cost Approximation Algorithms for Scheduling Independent Tasks on Hybrid Platforms. *https://doi.org/10.6084/m9.figshare.4674841.v1*, February 2017.

[15] Lin Chen, Deshi Ye, and Guochuan Zhang. Online scheduling on a cpu-gpu cluster. In *International Conference on Theory and Applications of Models of Computation*, pages 1–9. Springer, 2013.

[16] Lin Chen, Deshi Ye, and Guochuan Zhang. Online scheduling of mixed cpu-gpu jobs. *International Journal of Foundations of Computer Science*, 25(06):745–761, 2014.

[17] Nathanaël Cheriere and Erik Saule. Considerations on distributed load balancing for fully heterogeneous machines: Two particular cases. In *Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International*, pages 6–16. IEEE, 2015.

[18] Hoon Sung Chwa, Jaebaek Seo, Jinkyu Lee, and Insik Shin. Optimal real-time scheduling on two-type heterogeneous multicore platforms. In *Real-Time Systems Symposium, 2015 IEEE*, pages 119–129. IEEE, 2015.

[19] Martin Gairing, Burkhard Monien, and Andreas Woclaw. A faster combinatorial approximation algorithm for scheduling unrelated parallel machines. *Theoretical Computer Science*, 380(1):87–99, 2007.

[20] Thierry Gautier, Xavier Besseron, and Laurent Pigeon. Kaapi: A thread scheduling runtime system for data flow computations on cluster of multi-processors. In *Proceedings of the 2007 international workshop on Parallel symbolic computation*, pages 15–23. ACM, 2007.

[21] Jan Clemens Gehrke, Klaus Jansen, Stefan EJ Kraft, and Jakob Schikowski. A ptas for scheduling unrelated machines of few different types. In *SOFSEM*, pages 290–301. Springer, 2016.

[22] Dorit S Hochbaum and David B Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *Journal of the ACM (JACM)*, 34(1):144–162, 1987.

[23] Csanad Imreh. Scheduling problems on two sets of identical machines. *Computing*, 70(4):277–294, 2003.

[24] Safia Kedad-Sidhoum, Florence Monna, Grégory Mounié, and Denis Trystram. Scheduling independent tasks on multi-cores with GPU accelerators. In *European Conference on Parallel Processing*, pages 228–237. Springer, 2013.

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

[26] Jan Karel Lenstra, David B Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical programming*, 46(1-3):259–271, 1990.

[27] Alberto Marchetti-Spaccamela, Cyriel Rutten, Suzanne Van Der Ster, and Andreas Wiese. Assigning sporadic tasks to unrelated machines. *Mathematical Programming*, 152(1-2):247–274, 2015.

[28] Judit Planas, Rosa M. Badia, Eduard Ayguadé, and Jesús Labarta. Hierarchical task-based programming with StarSs. *IJHPCA*, 23(3):284–299, 2009.

[29] Gurulingesh Raravi and Vincent Nélis. A ptas for assigning sporadic tasks on two-type heterogeneous multiprocessors. In *Real-Time Systems Symposium (RTSS), 2012 IEEE 33rd*, pages 117–126. IEEE, 2012.

[30] Gurulingesh Raravi and Vincent Nélis. Task assignment algorithms for heterogeneous multiprocessors. *ACM Transactions on Embedded Computing Systems (TECS)*, 13(5s):159, 2014.

[31] Evgeny V Shchepin and Nodari Vakhania. An optimal rounding gives a better approximation for scheduling unrelated machines. *Operations Research Letters*, 33(2):127–133, 2005.

[32] David B Shmoys and Éva Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical programming*, 62(1-3):461–474, 1993.

[33] Mason Thammawichai and Eric C Kerrigan. Energy-efficient real-time scheduling for two-type heterogeneous multiprocessors. *arXiv preprint arXiv:1607.07763*, 2016.

[34] TOP500 Supercomputer Site. List of November 2016.

A Notation

Table 3 provides a list of the most used notations in this report.

B Worst-Case Examples

The example from Figure 1 relies on the weakness of the scheduling phase. Let us consider examples that rely on the weakness of the allocation phase with a limited number of processors:

Example 1. *Ratio* $\varphi = \frac{1+\sqrt{5}}{2} \approx 1.618$*:*

- $m = k = 1$
- *two tasks with costs* $c_1^1 = 1$, $c_1^2 = c_2^1 = \frac{1}{\varphi}$, $c_2^2 = \frac{1}{\varphi^2}$
- $\mathrm{OPT} = \frac{1}{\varphi}$ *and the makespan is 1*

Symbol	Definition
n	number of tasks
m	identical processors of type 1
k	identical processors of type 2
c_i^l	cost of task i on processors of type l
$\mu : \{1..n\} \to \{1,2\}$	allocation defining on which type of processors each task must be executed
$\sigma : \{1..n\} \to \{1..m+k\}$	schedule ($\sigma(i) \leq m \Leftrightarrow \mu(i) = 1$)
C_j	load of processor j ($\sum_{i:\sigma(i)=j} c_i^{\mu(i)}$)
$C_{\max}$	makespan ($\max_{1\leq j\leq m+k} C_j$)
OPT	makespan of an optimal schedule
$W^l(\mu)$	average work of processors of type l ($\frac{1}{m,k} \sum_{i:\mu(i)=l} c_i^l$)
$M^l(\mu)$	maximum cost of the tasks allocated to processors of type l ($\max_{i:\mu(i)=l} c_i^l$)
$\lambda(\mu)$	$\max(W^1(\mu), W^2(\mu), M^1(\mu), M^2(\mu))$
$\text{imax}(\mu)$	index of the largest task that is allocated to a processor of type 1 and that would be more efficient on a processor of type 2 ($\arg\max_{i:\mu(i)=1 \wedge c_i^1 > c_i^2} c_i^1$)
i_{start}	smallest index of tasks that are on processors of type 2 in the initial allocation of BalancedEstimate ($\min\{i : \mu(i) = 2\}$)
μ_i, μ'_i	allocations before and after task i is allocated to processors of type 1 at iteration i in BalancedEstimate

Table 3: List of notations.

This example may require a small task on processors of type 1 (dust tasks) to avoid task 1 to be moved back. It is possible to generalize it to arbitrary k by adding $k-1$ tasks with cost $\frac{1}{\varphi}$ on processors of type 2 and ∞ on processors of type 1.

Another similar example with $m = 2$:

Example 2. *Ratio* $\sqrt{2} \approx 1.414$*:*

- $m = 2$, $k = 1$
- *four tasks with costs* $c_1^1 = c_4^2 = 1 - \frac{\sqrt{2}}{2}$, $c_1^2 = c_2^2 = \infty$, $c_2^1 = c_3^2 = \frac{\sqrt{2}}{2}$, $c_3^1 = 1$, $c_4^1 = \sqrt{2} - 1$
- OPT $= \frac{\sqrt{2}}{2}$ *and the makespan is 1*

It is also possible to generalize this example to arbitrary k using the same principle (the cost on processors of type 2 must be $\frac{\sqrt{2}}{2}$).

Another similar example with $m = 3$:

Example 3. *Ratio* $\frac{6}{1+\sqrt{13}} \approx 1.302$*:*

- $m = 3$, $k = 1$
- *six tasks with costs* $c_1^1 = \frac{5-\sqrt{13}}{3}$, $c_1^2 = c_2^2 = c_3^2 = c_4^2 = \infty$, $c_2^1 = c_6^2 = \frac{5-\sqrt{13}}{6}$, $c_3^1 = \frac{\sqrt{13}-2}{3}$, $c_4^1 = c_5^2 = \frac{1+\sqrt{13}}{6}$, $c_5^1 = 1$ *and* $c_6^1 = \frac{\sqrt{13}-3}{2}$
- OPT $= \frac{1+\sqrt{13}}{6}$ *and the maskepan is 1*

C Related Algorithms

Table 1 summarizes the implemented approaches. The time cost of dynamic programming methods were considered to be prohibitive. The related code, data and analysis are available in [14].

Several algorithms from the literature have been implemented in StarPU [5] and XKaapi [20]. For StarPU[3], a specific project is dedicated to the examples[4]. The code is available in the directory `cholesky-bounds/src/pmtool` of the SVN repository[5] (revision 197). For XKaapi[6], the code is available in the directory `src/sched/` of the branch `origin/bleuser/wip/dual43` (commit `d53fb82`) of the git repository[7].

CLB2C (Algorithm 5), HeteroPrio (Algorithm 6) and BalancedEstimate (Algorithm 2) all relies on a similar principle: tasks are balanced to each type of processors depending of their acceleration factors and while minimizing the makespan. Their time complexities are similar. HeteroPrio is specially adapted to an online execution where tasks may arrive on the fly. It proceeds by transparently adding the new tasks in the ordered set of waiting tasks. CLB2C and BalancedEstimate may also be used in such a context but it may cause the predicted schedule to be partially reconsidered.

Algorithm 5: CLB2C [17]

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 schedule σ

Sort tasks by non-decreasing c_i^1/c_i^2
$i^1 \leftarrow 1$
$i^2 \leftarrow n$
while $i^1 \leq i^2$ **do**
  if $\min_{1\leq j\leq m} C_j + c_{i^1}^1 \leq \min_{m+1\leq j\leq m+k} C_j + c_{i^2}^2$ **then**
    $\sigma[i^1] \leftarrow \arg\min_{1\leq j\leq m} C_j$
    $i^1 \leftarrow i^1 + 1$
  else
    $\sigma[i^2] \leftarrow \arg\min_{m+1\leq j\leq m+k} C_j$
    $i^2 \leftarrow i^2 - 1$
return σ

DualHP (Algorithm 7 with the sub-procedure in Algorithm 8) and DADA (Algorithm 9 with the sub-procedure in Algorithm 10) are dual approximations [22]. Contrary to their original descriptions [8,10], DualHP and DADA sub-procedures do not fail if the minimum cost of any task is greater than λ because the initial lowest guess ensures this does not occur ($\max_{1\leq i\leq n} c_i^{\pi[i]} \leq \lambda$). To the best or our knowledge, we provide the first complete proofs of their approximation ratio below.

Theorem 4. DADA *(Algorithm 9) is a 2-approximation.*

Proof. The algorithm relies on a dual approximation mechanism. It considers a guess λ and tries to deliver a schedule of makespan at most 2λ and fails it is not possible. The process is

[3] `http://starpu.gforge.inria.fr/`
[4] `https://gforge.inria.fr/projects/starpu-examples/`
[5] `svn://scm.gforge.inria.fr/svnroot/starpu-examples`
[6] `http://kaapi.gforge.inria.fr`
[7] `https://scm.gforge.inria.fr/anonscm/git/kaapi/kaapi.git`

Algorithm 6: HeteroPrio [8]

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 schedule σ

Sort tasks by non-increasing c_i^1/c_i^2
$i^1 \leftarrow 1$
$i^2 \leftarrow n$
$\text{last}[j] \leftarrow 0$ for $1 \leq j \leq m+k$
for $i = 1 \dots n$ **do**
  $j \leftarrow \arg\min_{1 \leq j \leq m+k} C_j$
  if $j \leq m$ **then**
    $\sigma[i^1] \leftarrow j$
    $i^1 \leftarrow i^1 + 1$
    $\text{last}[j] \leftarrow i^1$
  else
    $\sigma[i^2] \leftarrow j$
    $i^2 \leftarrow i^2 - 1$
    $\text{last}[j] \leftarrow i^2$
for *each processor in non-decreasing* $C_{j'}$ **do**
  if *processor* j' *is a processor of type 1* **then**
    $j \leftarrow m + 1$
    do
      $j \leftarrow j + 1$
    while $j < m + k$ **and** ($\text{last}[j] = 0$ **or** $C_{j'} + c^1_{\text{last}[j]} \geq C_j$)
    if $\text{last}[j] = 0$ **and** $C_{j'} + c^1_{\text{last}[j]} < C_j$ **then**
      $\sigma[\text{last}[j]] \leftarrow j'$
      $\text{last}[j] \leftarrow 0$
      $\text{last}[j'] \leftarrow 0$
  else
    $j \leftarrow 1$
    do
      $j \leftarrow j + 1$
    while $j < m$ **and** ($\text{last}[j] = 0$ **or** $C_{j'} + c^2_{\text{last}[j]} \geq C_j$)
    if $\text{last}[j] = 0$ **and** $C_{j'} + c^2_{\text{last}[j]} < C_j$ **then**
      $\sigma[\text{last}[j]] \leftarrow j'$
      $\text{last}[j] \leftarrow 0$
      $\text{last}[j'] \leftarrow 0$
return σ

Algorithm 7: DUALHP [8] (with $\epsilon = 10^{-5}$)

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 schedule σ

$\pi_1 \leftarrow$ call horizon subprocedure with $\lambda_1 = \max_i \min(c_i^1, c_i^2)$
$\pi_2 \leftarrow$ call horizon subprocedure with $\lambda_2 = \sum_i \max(c_i^1, c_i^2)$
while $\lambda_2 - \lambda_1 > \epsilon$ **do**
    $\pi \leftarrow$ call horizon subprocedure with $\lambda = \frac{\lambda_1 + \lambda_2}{2}$
    if π *is a successful allocation* **then**
        $\pi_2 \leftarrow \pi$
        $\lambda_2 \leftarrow \lambda$
    else
        $\pi_1 \leftarrow \pi$
        $\lambda_1 \leftarrow \lambda$
Schedule tasks allocated to processors of type 1 in π_2 with LPT and EFT
Schedule tasks allocated to processors of type 2 in π_2 with LPT and EFT
return σ

Algorithm 8: DUALHP horizon subprocedure [8]

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 $1 \leq i \leq n, 1 \leq l \leq 2$,
Input : a horizon λ
Output: an allocation π

Sort tasks by non-increasing c_i^1/c_i^2
$\pi[i] \leftarrow 2$ for $i \in \{i : c_i^1 > \lambda\}$
$\pi[i] \leftarrow 1$ for $i \in \{i : c_i^2 > \lambda\}$
for $i = 1 \dots n$ **do**
    if $\pi[i]$ *is undefined* **and** $\frac{\sum_{1 \leq j \leq n, \pi[j]=2} c_j^2}{k} < \lambda$ **then**
        $\pi[i] \leftarrow 2$
$\pi[i] \leftarrow 1$ for $i \in \{i : \pi[i] \neq 2\}$
if $\frac{\sum_{1 \leq i \leq n, \pi[i]=1} c_i^1}{m} < \lambda$ **then**
    return Failure
return π

Algorithm 9: DADA [10] (with $\epsilon = 10^{-5}$)

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 schedule σ

$\sigma_1 \leftarrow$ call horizon subprocedure with $\lambda_1 = \max_i \min(c_i^1, c_i^2)$
$\sigma_2 \leftarrow$ call horizon subprocedure with $\lambda_2 = \sum_i \max(c_i^1, c_i^2)$
while $\lambda_2 - \lambda_1 > \epsilon$ **do**
  $\sigma \leftarrow$ call horizon subprocedure with $\lambda = \frac{\lambda_1 + \lambda_2}{2}$
  if σ *is a successful schedule* **then**
    $\sigma_2 \leftarrow \sigma$
    $\lambda_2 \leftarrow \lambda$
  else
    $\sigma_1 \leftarrow \sigma$
    $\lambda_1 \leftarrow \lambda$
return σ_2

Algorithm 10: DADA horizon subprocedure [10]

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 $1 \leq i \leq n, 1 \leq l \leq 2$,
Input : a horizon λ
Output: a schedule σ

Sort tasks by non-increasing c_i^1/c_i^2
for $i = 1 \ldots n$ **do**
  if $c_i^1 > \lambda$ **then**
    $\sigma[i] \leftarrow \arg\min_{m+1 \leq j \leq m+k} C_j$
  if $c_i^2 > \lambda$ **then**
    $\sigma[i] \leftarrow \arg\min_{1 \leq j \leq m} C_j$
for $i = 1 \ldots n$ **do**
  if $\sigma[i]$ *is undefined* **then**
    if $\min_{m+1 \leq j \leq m+k} C_j + c_i^2 \leq 2\lambda$ **then**
      $\sigma[i] \leftarrow \arg\min_{m+1 \leq j \leq m+k} C_j$
    else
      break
for $i = 1 \ldots n$ **do**
  if $\sigma[i]$ *is undefined* **then**
    $\sigma[i] \leftarrow \arg\min_{1 \leq j \leq m} C_j$
if $C_{\max} > 2\lambda$ **then**
  return Failure
return σ

repeated with a binary process to find the lowest guess λ that yields a schedule. To prove that this is a 2-approximation, it remains to prove that the sub-procedure in Algorithm 10 produces a schedule of makespan at most 2λ if there exists an optimal schedule with a makespan lower than or equal to λ.

In the remaining of this proof, we assume that such a schedule exists. We need to prove that the algorithm necessarily builds a schedule of makespan at most 2λ.

First, we need to prove that $C_{\max} \leq 2\lambda$ after the first loop. The tasks that have been scheduled on their best processors are scheduled on the same processors in any optimal schedule (otherwise, the optimal makespan is greater than or equal to λ, which is false by assumption). Thus, for each type of processors, the average work in σ is lower than or equal to the average work in an optimal schedule, which is lower than or equals to λ. We can use Lemma 7 to show that $C_{\max} \leq 2\lambda$ after the first loop.

To conclude this proof, we need to prove that the makespan does not exceed 2λ after the last two loops. It is straightforward when the condition on Line 9 is always true (all tasks are scheduled on GPUs). We thus assume that it is false for some task and that the CPUs have at least one task scheduled on them in the last loop. In this case, the average work on GPUs, noted W^2, is larger than λ because it is larger than or equal to $\min_{m+1\leq j\leq m+k} C_j$ and for some task i the condition on Line 9 was false (we also know by assumption that any cost is lower than or equal to λ). We will show that the average work on the CPU, noted W^1, is lower than or equal to λ, which will concludes the proof by application of Lemma 7. The principle is similar to the proof of Lemma 4: we consider an optimal schedule and the sets S^1 and S^2 of task indices that are on different types of processors in σ and this optimal schedule. Let W^1_{OPT} (resp. W^2_{OPT}) be the average work on processors of type 1 (resp. 2) in this optimal schedule. Then, $W^2 = W^2_{\text{OPT}} + \frac{\sum_{i\in S^2} c_i^2 - \sum_{i\in S^1} c_i^2}{k}$. Thus, $\sum_{i\in S^2} c_i^2 < \sum_{i\in S^1} c_i^2$ because $W^2 > \lambda$ and $W^2_{\text{OPT}} \leq \lambda$. Furthermore, $\min_{i\in S^2} \frac{c_i^2}{c_i^1} \sum_{i\in S^2} c_i^1 < \max_{i\in S^1} \frac{c_i^2}{c_i^1} \sum_{i\in S^1} c_i^1$ and $\sum_{i\in S^2} c_i^1 < \sum_{i\in S^1} c_i^1$ because $\max(S^1) < \min(S^2)$ and tasks are ordered by non-increasing $\frac{c_i^1}{c_i^2}$. Finally, we have $W^1 = W^1_{\text{OPT}} + \frac{\sum_{i\in S^1} c_i^1 - \sum_{i\in S^2} c_i^1}{k}$ and thus $W^1 \leq \lambda$ because $W^1_{\text{OPT}} \leq \lambda$, which concludes the proof. □

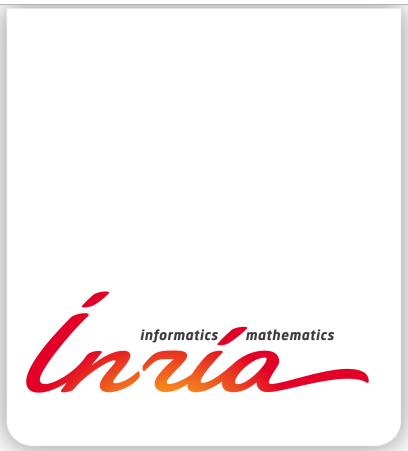
Theorem 5. DUALHP *(Algorithm 7) is a 2-approximation.*

Proof. The proof is analogous to the proof of Theorem 4. We need to prove that if λ is greater than or equal to the makespan of an optimal schedule OPT, then the subprocedure in Algorithm 8 returns an allocation from which it is possible to build a schedule with a makespan lower than or equal to 2λ. We assume that $\lambda \leq \text{OPT}$. Then, the minimum cost for each task is lower than or equal to λ. Tasks that have one cost greater than λ are allocated to their best processors on Lines 2 and 3. The costs of the remaining unallocated tasks are all lower than or equal to λ. After the loop, either all the tasks are then allocated to processors of type 2 or the second part of the condition on Line 5 is false for at least one unallocated task.

Case 1: In the former case, the average work of processors of type 2 is lower than $(1+\frac{1}{k})\lambda$ because the condition ensures that the average work is lower than λ until the last allocated task, which has a cost lower than or equal to λ. In this case, we can use Lemma 7 to show that it is possible to build a schedule of makespan at most 2λ.

Case 2: In the latter case, the average work on processors of type 2 is greater than or equal to λ. Using a similar technique as in the proof of Theorem 4, we can show this implies that the average work on processors of type 1 is lower than or equal to λ. We use Lemma 7 to conclude the proof.

□



RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399