

# 3/2-Dual Approximation for CPU/GPU Scheduling

Bernhard Sebastian Germann ✉


Kiel University, Germany

Klaus Jansen ✉ 

Kiel University, Germany

Felix Ohnesorge ✉

Kiel University, Germany

Malte Tutas ✉ 

Kiel University, Germany

---

## Abstract

We present a fast and efficient  $3/2$  dual approximation algorithm for CPU/GPU scheduling under the objective of makespan minimization. In CPU/GPU scheduling tasks can be scheduled on two different architectures. When executed on the CPU, a task is moldable and can be assigned to multiple cores. The running time becomes a function in the assigned cores. On a GPU, the task is a classical job with a set processing time. Both settings have drawn recent independent scientific interest. For the moldable CPU scheduling, the current best known constant rate approximation is a  $3/2$  approximation algorithm [Wu et al. EJOR volume 306]. The best efficient algorithm for this setting is a  $3/2+\epsilon$  approximation [Mounie et al. SIAM '07] whereas GPU scheduling admits a  $13/11$  approximation [Coffman, Garey, Johnson SIAM'78]. We improve upon the current best known algorithms for CPU/GPU scheduling by Bleuse et al. by formulating a novel multidimensional multiple choice knapsack to allot tasks to either architecture and schedule them there with known algorithms. This yields an improved running time over the current state of the art. We complement our theoretical results with experimentation that shows a significant speedup by using practical optimizations and explore their efficacy.

**2012 ACM Subject Classification** Theory of computation → Rounding techniques; Mathematics of computing → Discrete optimization; Theory of computation → Scheduling algorithms

**Keywords and phrases** computing, machine scheduling, moldable, CPU/GPU

**Digital Object Identifier** 10.4230/LIPIcs.SEA.2024.13

## Supplementary Material

*Software (Source Code)*: <https://github.com/Feliox/CPU-GPU-Scheduling> [5]  
archived at `swh:1:dir:6b62fa88e3d3dde792b3c7a9803a04ef24651239`

**Funding** Supported in part by DFG-Project JA 612/25-1.

## 1 Introduction

Scheduling tasks in an efficient manner is one of the most ubiquitous problems in operations research. Applications range from the minute details of computing to large scale factory management and concern most facets of modern society. Computers are becoming more and more powerful, but to fully leverage that power efficient algorithms are required. Tasks that can be executed on parallel and distributed systems such as GPUs and CPUs require sophisticated algorithms to schedule. While sequential tasks on the GPU can be executed without any overhead by simply assigning a starting time and machine to each task, the CPU tasks require a different approach. Tasks on CPUs are *moldable*, i.e. a task that can be executed on several processors. Their execution time is a function dependent on the number of assigned machines. To account for communication costs between the processors



© Bernhard Sebastian Germann, Klaus Jansen, Felix Ohnesorge, and Malte Tutas;  
licensed under Creative Commons License CC-BY 4.0

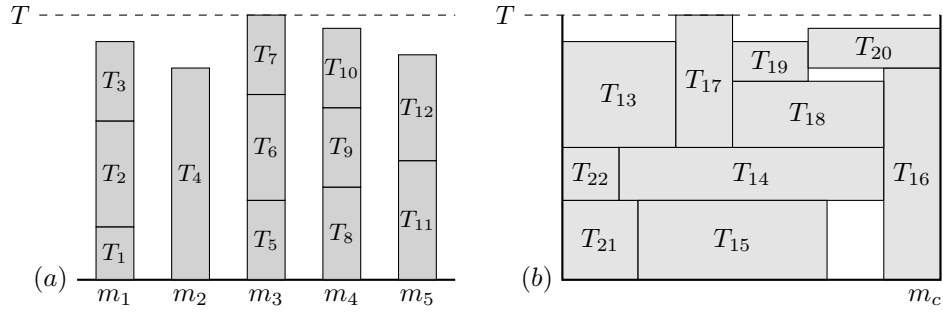
22nd International Symposium on Experimental Algorithms (SEA 2024).

Editor: Leo Liberti; Article No. 13; pp. 13:1–13:18

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** An illustration of a CPU/GPU schedule. On the GPU side (a) each task occupies a single processor, where the height represents the processing time. On the CPU side (b), the width of tasks represents the number of machines assigned to a task, while the height represents the processing time on as many machines. The makespan is defined by  $T_7$  and  $T_{17}$  and denoted with  $T$ .

executing the task this function is not linear in the amount of used processors. Both of these settings have been studied in the context of makespan minimization [17, 8]. The makespan corresponds to the earliest time at which all tasks in the given instance finish when all processors start at the same point in time. Applications of makespan scheduling are again widespread.

In this paper, we consider the CPU/GPU scheduling problem. Here, each task can be executed on either the GPU or CPU architecture, i.e. it has both a processing time when executed on the GPU and a function representing the moldable work for execution on the CPU. We present a low complexity algorithm that yields a  $3/2 + \varepsilon$  approximation guarantee of the maximum makespan.

## 1.1 Notation

We define the natural numbers as  $\mathbb{N} := \{1, 2, 3, \dots\}$  and  $\mathbb{N}_0 := \mathbb{N} \sqcup \{0\}$ , where  $\sqcup$  denotes the disjoint union. More, we define  $[n] := \{m \in \mathbb{N} : m \leq n\}$  and  $[n]_0 := [n] \sqcup \{0\}$  for all  $n \in \mathbb{N}_0$ .

## 1.2 Problem Definition

In  $P|C_{\max}$  (or GPU scheduling) a task can be executed only sequentially on one machine. An instance is given by a number  $n \in \mathbb{N}$  of tasks, a number  $m \in \mathbb{N}$  of machines and processing times  $p_j \in \mathbb{Q}_{>0}$  for all tasks  $j \in [n]$ . The goal is to find a schedule  $\pi : [n] \rightarrow [m]$  of tasks to machines, that minimizes the maximum makespan  $\max_{i \in [m]} \sum_{j \in \pi^{-1} i} p_j$ .

In  $P|\text{fnc}_j|C_{\max}$  (or CPU scheduling, also often called *scheduling independent monotonic moldable tasks*) a task can be executed on multiple machines in parallel. An instance is given by a number of tasks  $n \in \mathbb{N}$ , a number of machines  $m \in \mathbb{N}$ , and for each task  $j \in [n]$  a function  $p_j : [m] \rightarrow \mathbb{Q}_{>0}$ , where  $p_j(\ell)$  describes its processing time when scheduled on  $\ell$  machines. We define the work (area) function  $w_j : [m] \rightarrow \mathbb{Q}_{>0}$  for every task  $j \in [n]$  by  $w_j(\ell) := \ell p_j(\ell)$ . Also, we assume the monotony properties that first,  $p_j$  is non-increasing and second,  $w_j$  is non-decreasing for all tasks  $j \in [n]$ . These represent the assumption that assigning more machines to a task does not increase its processing time but may come with an increase in work due to communication overhead. Further, we define for each task  $j \in [n]$  the function for the *canonical number of machines*  $\gamma_j : \mathbb{Q}_{>0} \rightarrow [m]$  such that  $\gamma_j(h)$  is the minimal number of machines needed to execute task  $j$  in time  $h$ , and by convention set to  $+\infty$  if no such number exists.

A schedule is a mapping  $\pi : [n] \rightarrow \mathbb{Q}_{>0} \times \mathbb{N}$ , that maps to each task a starting time and a number of machines. The maximum makespan of a schedule is  $\max\{s + p_j(\ell) : j \in [n], (s, \ell) = \pi(j)\}$ . A schedule is valid if the total number of machines that tasks are scheduled to does not exceed  $m$  at any point in time. The goal is to find a schedule that minimizes the maximum makespan. For an illustration of such a schedule see Figure 1(b).

CPU/GPU scheduling is a combination of CPU scheduling and GPU scheduling, where each task can be scheduled to either the CPU or the GPU. An instance consists of a number of tasks  $n \in \mathbb{N}$ , the number of CPU machines  $m_c$  and the number of GPU machines  $m_g$ , as well as a processing time on the GPU and a processing time function on the CPU for each task. A valid schedule is a partition of the tasks onto the CPU and GPU and a valid schedule on each for its respective partition. The maximum makespan is the maximum of both makespans. The goal is to find a schedule that minimizes the overall maximum makespan.

### 1.3 Related Work

CPU/GPU scheduling combines two scheduling paradigms. It is known that, on their own, both CPU and GPU scheduling are NP-complete [15, 11]. Furthermore, without monotony assumptions CPU-scheduling does not admit an approximation algorithm with an approximation rate less than  $(3/2)$  unless  $P = NP$  [3].

For moldable CPU scheduling, there exist several approximation algorithms. Most relevant to this work is the algorithm by Mounié et al. [17]. They present a  $(3/2 + \varepsilon)$ -approximation algorithm that runs in time  $\mathcal{O}(nm_c \log(n/\varepsilon))$ . Their paper is in turn based on a two-phase approach by Turek et al. [18]. There have been several improvements made to the running time of the algorithm by Mounié. First, Jansen and Land reduced the dependence of the running time regarding  $m$  to a poly-logarithmic factor [10]. Later on, Grage et al. further improved this poly-logarithmic dependence, yielding a running time of  $\mathcal{O}(n \log^2(\frac{1}{\varepsilon} + \frac{\log(\varepsilon m_c)}{\varepsilon}) + \frac{n}{\varepsilon} \log(\frac{1}{\varepsilon}) \log(\varepsilon m_c))$ . The best approximation ratio for this problem is a  $(3/2)$  approximation algorithm given by Chen et al. [19]. Furthermore, it is known that this problem admits a PTAS [12] however its running time depends on a superexponential function. Jansen and Land show that there exists an FPTAS for the case where there are many ( $m > 8 \cdot n/\varepsilon$ ) machines [10]. The running time of the FPTAS in this case was again improved in [6].

On the GPU side we model one of the most widespread and well-researched scheduling problems, makespan scheduling on parallel machines. This problem has been researched continuously since at least 1969 [7]. As the research in this field is so varied, we focus only on the results relevant to our work. For constant rate approximations, there exists the  $(4/3)$  approximation LPT-algorithm [7], which runs efficiently in  $\mathcal{O}(n \log(n))$ . Furthermore, there is the best known constant rate approximation algorithm provided by MULTIFIT, with a ratio of  $(13/11)$  [13]. It is known that this problem admits an EPTAS [9]. More relevant to our result is the PTAS developed by Hochbaum and Shmoys [8]. They present the dual approximation framework, which we also employ to generate our solution.

The combination of both settings, i.e. CPU/GPU scheduling, was first considered by Bleuse et al. in [1]. They present an ILP based  $(3/2 + \varepsilon)$ -approximation algorithm whose complexity is not polynomial in theory. Additionally, the authors claim that adapting a technique from [2] yields a dual approximation algorithm with running time  $\mathcal{O}(n^2 m_c^4 m_g^2)$  and the same approximation ratio however, they do not explain how to adapt the technique.

## 1.4 Our Contribution

We present a low complexity  $(3/2 + \varepsilon)$ -algorithm for CPU/GPU scheduling that runs in time  $\mathcal{O}(\log(nm_cm_g/\varepsilon) \cdot n^2m_cm_g)$  and optimize it for the case  $n > m_g^2$  such that it then runs in time  $\mathcal{O}(\log(nm_cm_g/\varepsilon) \cdot nm_c \max(n, m_g^2))$ . Our algorithm follows the idea and structure of the CPU scheduling algorithm by Mounié et al. [17] and extends it to the CPU/GPU setting. Further, we implemented our algorithms and evaluated them experimentally.

## 2 Description of the algorithm

In this section, we describe the algorithm that comprises our main result. We begin by describing the framework in which we embed the algorithm. Then we describe how to generate suitable bounds to apply the framework. Next, we present the main part of the algorithm, the allocation of tasks onto machines. We finish the description of the algorithm by describing how we calculate a feasible schedule for the sequential machines and argue the complexity of the entire algorithm in its framework.

### 2.1 Dual Approximation Framework

Our algorithm uses the dual approximation framework [8].

► **Definition 1** ( $\lambda$ -dual approximation problem). *Let  $\lambda > 1$ . The  $\lambda$ -dual approximation problem to a makespan minimization scheduling problem consists of a scheduling instance  $I$  and a makespan guess  $d$ . The goal is to either find a schedule with a makespan of at most  $\lambda d$  or decide that no schedule with a makespan at most  $d$  exists.*

Now, given lower and upper bounds for the minimal makespan, one can use a  $\lambda$ -dual approximation algorithm to binary search for the minimal makespan and obtain a  $\lambda(1 + \varepsilon)$ -approximation for the primal problem. This yields the following lemma, a proof of which can be derived from [8].

► **Lemma 2** (primal approximation). *Let  $I$  be a makespan minimisation scheduling instance,  $d^*$  its minimal makespan,  $k \geq 0$ ,  $\lambda > 1$ ,  $\varepsilon > 0$ , and  $\omega \in \mathbb{Q}_{>0}$  a lower bound such that  $\omega \leq d^* \leq (2 + k)\omega$ . Then we can construct from a  $\lambda$ -dual approximation algorithm that runs in time  $\mathcal{O}(f(I))$  a  $\lambda(1 + \varepsilon)$ -approximation algorithm for the primal problem that runs in time  $\mathcal{O}(\log(1/\varepsilon + k)f(I))$ .*

The goal of this paper is to construct a  $3/2$ -dual approximation algorithm for CPU/GPU scheduling to then use Lemma 2. But first, we have to find an algorithm that constructs a suitable lower bound on the optimal makespan. For CPU scheduling, Ludwig and Tiwari [16] describe an algorithm that computes a lower bound in time  $\mathcal{O}(mn \log n)$ . Next, we show that their approach can be adapted to CPU/GPU scheduling.

► **Lemma 3.** *For an instance of CPU/GPU scheduling with optimal makespan  $d^*$ , a lower bound  $\omega \in \mathbb{Q}_{>0}$  can be computed in time  $\mathcal{O}((m_cn + m_g) \log(m_cn + m_g))$ , with  $\omega \leq d^* \leq 4\omega$ .*

**Proof.** Let  $t_j : [m_c] \rightarrow \mathbb{Q}_{>0}$  be the processing time function,  $w_j$  the work area function on the CPU, and  $p_j$  the processing time on the GPU for each task  $j \in [n]$ .

Now we define the mapping  $\omega : [m_c]^{[n]} \times \mathcal{P}([n]) \rightarrow \mathbb{Q}_{>0}$  by

$$\omega(\alpha, A) := \max \left\{ \frac{1}{m_c} \sum_{j \in A} w_j(\alpha(j)), \max_{j \in A} \{t_j(\alpha(j))\}, \frac{1}{m_g} \sum_{j \in A^c} p_j, \max_{j \in A^c} \{p_j\} \right\}.$$

There

- $A \subseteq [n]$  represents the tasks scheduled on the CPU, and  $A^C = [n] \setminus A$  on the GPU,
- the mapping  $\alpha : A \rightarrow [m_c]$  assigns to each task a number of CPU machines,
- $\frac{1}{m_c} \sum_{j \in A} w_j(\alpha(j))$  is the average work per machine on the CPU,
- $\max_{j \in A} \{t_j(\alpha(j))\}$  is the maximum processing time of any task on the CPU,
- $\frac{1}{m_g} \sum_{j \in A^C} p_j$  is the average work per machine on the GPU, and
- $\max_{j \in A^C} \{p_j\}$  is the maximum processing time of any task on the GPU.

Given an assignment  $\alpha$  and a partition  $A \cup A^C = [n]$ , the 2-approximate algorithm by Garey and Graham yields a CPU/GPU schedule with a makespan that does not exceed  $2\omega(\alpha, A)$  [4]. This suggests the following strategy to find the desired lower bound: Find an assignment and partition such that  $\omega(\alpha, A) \leq d^* \leq 2\omega(\alpha, A)$ .

Now,  $\omega$  defined by

$$\omega := \min_{A \subseteq [n]} \min_{\alpha: [n] \rightarrow [m_c]} \omega(\alpha, A)$$

is a trivial lower bound on the minimum makespan for CPU/GPU scheduling, and, by Garey and Graham's algorithm, satisfies  $\omega \leq d^* \leq 2\omega$ . Next, we will derive an efficient way to compute  $\omega$ .

Let  $H := \{p_j : j \in [n]\} \cup \{t_j(p) : j \in [n], p \in [m_c]\}$  be the set of processing times of tasks on the CPU/GPU. Further, let  $h_c(\alpha, A) := \max_{j \in A} t_j(\alpha(j))$ , and  $h_g(A^C) := \max_{j \in A^C} p_j$  be the maximum processing times on CPU and GPU respectively, given an assignment  $\alpha : A \rightarrow [m_c]$  and a subset  $A \subseteq [n]$ . Then we can rewrite  $\omega$  as follows (see the Appendix for the complete derivation):

$$\omega = \min_{\tau \in H} \max \left\{ \tau, \min_{\substack{A \subseteq [n] \\ \text{s.t. } h_g(A^C) \leq \tau}} \max \left\{ \sum_{j \in A} \frac{w_j(\gamma_j(\tau))}{m_c}, \sum_{j \in A^C} \frac{p_j}{m_g} \right\} \right\}.$$

Here, computing the term

$$W(\tau) := \min_{\substack{A \subseteq [n] \\ \text{s.t. } h_g(A^C) \leq \tau}} \max \left\{ \sum_{j \in A} \frac{w_j(\gamma_j(\tau))}{m_c}, \sum_{j \in A^C} \frac{p_j}{m_g} \right\}$$

for some  $\tau \in H$  can be interpreted as minimizing the maximum makespan of two unrelated machines ( $R2||C_{\max}$ ), where  $\frac{w_j(\gamma_j(\tau))}{m_c}$  is the processing time for task  $j$  on the first machine, and  $\frac{p_j}{m_g}$  for the second machine, with the additional constraint that tasks  $j \in [n]$  with  $p_j > \tau$  must be assigned to the first machine, which can be modeled by setting their processing time for the second machine to  $+\infty$ .

Now note that the identity mapping  $\tau \mapsto \tau$  is increasing, while  $\tau \mapsto W(\tau)$  is non-increasing. So one can find  $\min_{\tau \in H} \max\{\tau, W(\tau)\}$  by binary searching over the sorted processing times in  $H$ , as described by Ludwig and Tiwari in [16].

The problem here is that we can not compute  $W(\tau)$  in polynomial time, since makespan minimization for two identical machines is already NP-complete [15]. But the greedy algorithm, that assigns each task to the machine on which it takes the least time, is a 2-approximate algorithm for  $R2||C_{\max}$  [15].

When using a 2-approximation  $\widetilde{W}(\tau)$  with  $W(\tau) \leq \widetilde{W}(\tau) \leq 2W(\tau)$ , the binary search yields a 2-approximation  $\widetilde{\omega}$  of  $\omega$  with  $\omega \leq \widetilde{\omega} \leq 2\omega$ . Thus

$$\widetilde{\omega}/2 \leq \omega \leq d^* \leq 2\omega \leq 2\widetilde{\omega}.$$

So  $\widetilde{\omega}$  satisfies our criteria.

Finally, we analyze the complexity of computing  $\tilde{\omega}$ . Sorting  $H$  is in  $\mathcal{O}((m_c n + m_g) \log(m_c n + m_g))$ . The binary search takes  $\mathcal{O}(\log(m_c n + m_g))$  steps, each step costing  $\mathcal{O}(n)$  to solve the unrelated machine scheduling and  $\mathcal{O}(n \log m_c)$  to find  $\gamma_j(\tau)$  for each task  $j$  via binary search. Therefore the overall runtime is dominated by sorting  $H$ . ◀

Now that we can bound the optimal makespan for CPU/GPU scheduling, the next step is to derive a 3/2-dual approximate algorithm. We closely follow the approach from [17].

Let  $d$  be a makespan guess of some CPU/GPU scheduling instance. If a schedule of length at most  $d$  exists, then a 3/2-dual approximate algorithm has to find a schedule of length at most  $3d/2$ , but else it can reject the guess. We can exploit the assumption, that a schedule of at most  $d$  exists, to reduce the search space by finding properties, that such schedules must satisfy and restrict the search to those. We list some basic such properties:

► **Remark 4 (structure of schedules with bounded length)**. Every CPU/GPU schedule of length at most  $d \in \mathbb{Q}_{>0}$  satisfies:

1. The processing time of each task in that schedule is at most  $d$ , the total work area on the CPU is at most  $m_c d$ , and the total work area in the GPU is at most  $m_g d$ .
2. If two tasks are scheduled to the CPU on the same machine, then at least one of them has a processing time at most  $d/2$ . This implies, that tasks on the CPU with more processing time than  $d/2$  use at most  $m_c$  machines in total.
3. For each machine on the GPU: First, if a task with processing time more than  $2d/3$  is scheduled to it, then other tasks scheduled to it have processing time at most  $d/3$ . Second, at most two tasks with processing time more than  $d/3$  can be scheduled on it.

Derived from Remark 4, the basic idea from [17] for CPU scheduling is to first partition the tasks into two sets,  $\mathcal{T}_1$  for tasks with processing time more than  $d/2$ , and  $\mathcal{T}_2$  for those with time at most  $d/2$ , such that the properties from Remark 4 are satisfied, and then compute a schedule from that partition. Note that the processing time of a task depends on the number of machines they are assigned to. But by also noting, that if those properties are satisfied under some assignment of tasks to machines, then they stay satisfied when assigning each task  $j$  from  $\mathcal{T}_1$  to  $\gamma_j(d)$  machines, and from  $\mathcal{T}_2$  to  $\gamma_j(d/2)$ . In particular, they showed:

► **Lemma 5 (BuildFeasible [17])**. Let  $I = (n, m, (t_j)_{j \in [n]})$  be an instance of CPU scheduling and  $d \in \mathbb{Q}_{>0}$  a makespan guess. Given a partitioning of tasks  $\mathcal{T}_1 \sqcup \mathcal{T}_2 = [n]$  with

- $\sum_{j \in \mathcal{T}_1} \gamma_j(d) \leq m$  and
- $\sum_{j \in \mathcal{T}_1} w_j(\gamma_j(d)) + \sum_{j \in \mathcal{T}_2} w_j(\gamma_j(d/2)) \leq md$

one can compute a schedule of length at most  $3d/2$  in time  $\mathcal{O}(mn)$ .

We can extend their idea to CPU/GPU scheduling by adding a third partition  $\mathcal{T}_g$  for tasks that are to be scheduled on the GPU. Now it is left to show how to compute such a partitioning, and how to compute a schedule of length at most  $3d/2$  on the GPU from  $\mathcal{T}_g$ .

## 2.2 The 3/2-dual approximate algorithm

Next, we describe how to compute a partitioning into sets  $\mathcal{T}_1$ ,  $\mathcal{T}_2$ ,  $\mathcal{T}_g$  as described previously, that satisfies the properties from Remark 4. As preparation we want to express also Property 3 as a simple inequality constraint as we did Property 2. For that, we define a size function that classifies tasks as being either *small*, *medium*, or *big* on the GPU.

► **Definition 6 (size function  $\varphi$ )**. Let there be an instance of CPU/GPU scheduling and a makespan guess  $d$ . Then we define the size function  $\varphi : [n] \rightarrow \{0, 1, 2, +\infty\}$  by  $\varphi(j) := \max\{k \in \{0, 1, 2\} : kd/3 < p_j\}$  if  $p_j \leq d$ , and  $\varphi(j) := +\infty$  if  $p_j > d$ .

Then Property 3 and  $p_j \leq d$  for every  $j \in [n]$  implies, that the total size of tasks on the GPU is at most  $2m_g$ . The remaining properties of Remark 4 can be expressed by the following constraints:

► **Definition 7** (Constraints 1-4). *Let there be a CPU/GPU scheduling instance, let  $d \in \mathbb{Q}_{>0}$  be a makespan guess, and  $\mathcal{T}_1 \sqcup \mathcal{T}_2 \sqcup \mathcal{T}_g = [n]$  a partitioning. We define constraints*

1.  $\sum_{j \in \mathcal{T}_1} w_j(\gamma_j(d)) + \sum_{j \in \mathcal{T}_2} w_j(\gamma_j(d/2)) \leq m_c d$
2.  $\sum_{j \in \mathcal{T}_1} \gamma_j(d) \leq m_c$
3.  $\sum_{j \in \mathcal{T}_g} \varphi(j) \leq 2m_g$
4.  $\sum_{j \in \mathcal{T}_g} p_j \leq m_g d$

Now, a core insight is that partitioning the tasks under these constraints can be modeled as a multidimensional multiple-choice knapsack problem (MMCKP), as described in [14]. Here we interpret the first constraint as the cost that has to be minimized, and the other three constraints from Definition 7 are represented as weights in the MMCKP, where the right-hand sides (RHSs) of Constraints 2-4 can be seen as a capacity vector  $(a, b, c)$ . For now, we ignore that for a knapsack formulation weights must be integer, and thus the instance processing times would need to be scaled first. While the MMCKP is NP-hard, there is a pseudopolynomial algorithm that uses dynamic programming and runs in time  $\mathcal{O}(n \cdot abc)$  [14], similar to the standard 0-1-knapsack problem. Here, the problem is, that the two RHSs  $m_c d$  and  $m_g d$  are in general not polynomial in  $n, m_c, m_g$ , and thus neither would be the complexity of solving the knapsack, since at least one of them would turn up in the capacity vector. We manage this by choosing Constraint 1 as the minimization target and employing a scaling and rounding scheme for Constraint 4, that bounds its RHS but introduces a rounding error. Next, we define the scaled and rounded constraint and discuss its choice afterward.

► **Definition 8** ( $\delta, \mu, \tilde{p}_j$ , Constraint 5). *Let there be a CPU/GPU scheduling instance, and a makespan guess  $d \in \mathbb{Q}_{>0}$ , and  $\mathcal{T}_g \subseteq [n]$ . Let there also be a error coefficient  $\delta \in \mathbb{Q}_{>0}$ . We define the scaling factor  $\mu := \frac{n}{\delta d m_g}$ , and the scaled and rounded GPU processing times  $\tilde{p}_j := \lfloor \mu p_j \rfloor$  for all tasks  $j \in [n]$ . Then we define the constraint*

5.  $\sum_{j \in \mathcal{T}_g} \tilde{p}_j \leq \mu \cdot m_g d = n/\delta$ .

We need these definitions to satisfy some requirements that are critical for us. First, the  $\tilde{p}_j$  are integers, which is necessary for the dynamic program that solves the MMCKP. Second, for a constant  $\delta \in \mathbb{Q}_{>0}$  the RHS is a small polynomial in  $n, m_c$  and  $m_g$  such that we can solve the MMCKP efficiently. These two are obviously satisfied. But third, since we want to compute a GPU schedule of length at most  $3d/2$  from a partitioning that satisfies the knapsack constraints, we also need the constraint to imply a good bound on the total work area on the GPU. For example, if the total work area on the GPU of the tasks in  $\mathcal{T}_g$  is more than  $2dm_g$ , then deriving a GPU schedule of length at most  $3d/2$  would be impossible. The next lemma shows the bound that Constraint 5 implies for the total work on the GPU, which puts it in relation to Constraint 4.

► **Lemma 9.** *Let there be a CPU/GPU scheduling instance,  $d \in \mathbb{Q}_{>0}$  a makespan guess,  $\delta \in \mathbb{Q}_{>0}$ , and  $\mathcal{T}_g \subseteq [n]$ . If Constraint 5 holds, then  $\sum_{j \in \mathcal{T}_g} p_j \leq m_g \cdot (1 + \delta)d$ . If Constraint 5 does not hold, then  $\sum_{j \in \mathcal{T}_g} p_j > m_g d$ .*

**Proof.** The definition of  $\tilde{p}_j$  yields  $\tilde{p}_j/\mu \leq p_j \leq (\tilde{p}_j + 1)/\mu$  for all  $j \in [n]$ . If Constraint 5 holds:

$$\begin{aligned} \sum_{j \in \mathcal{T}_g} p_j &\leq \sum_{j \in \mathcal{T}_g} \left( \frac{\tilde{p}_j}{\mu} + \frac{1}{\mu} \right) \leq \frac{1}{\mu} \sum_{j \in \mathcal{T}_g} \tilde{p}_j + \frac{n}{\mu} \\ &\leq \frac{1}{\mu} \cdot \frac{n}{\delta} + \frac{n}{\mu} = m_g d + m_g \delta d = m_g \cdot (1 + \delta) d, \end{aligned}$$

and if Constraint 5 does not hold:  $\sum_{j \in \mathcal{T}_g} p_j \geq \frac{1}{\mu} \sum_{j \in \mathcal{T}_g} \tilde{p}_j > \frac{1}{\mu} \cdot \frac{n}{\delta} = m_g d$ .  $\blacktriangleleft$

The preceding Lemma 9 shows that the scaling and rounding scheme implies a relaxation of Constraint 4 by the factor  $(1 + \delta)$ . Therefore reducing the error coefficient  $\delta$  improves the bound on the total work by reducing the rounding error, but increases the time complexity of solving the MMCKP by increasing the RHS. The maximal possible choice of  $\delta$  depends on the algorithm used to build a GPU schedule from the partition  $\mathcal{T}_g$ . Next, we describe this algorithm and bound the makespan of the resulting schedule under the assumption that Constraint 3 and a relaxed version of Constraint 4 hold.

► **Lemma 10** (Build feasible schedule for the GPU). *Let there be a CPU/GPU scheduling instance, a makespan guess  $d \in \mathbb{Q}_{>0}$ ,  $c \geq 0$ , and  $\mathcal{T}_g \subseteq [n]$ . If Constraint 3 holds, and  $\sum_{j \in \mathcal{T}_g} p_j \leq m_g(1 + c)d$ , then we can construct a GPU schedule of length at most  $(4/3 + c)d$  in time  $\mathcal{O}(n \log n)$ .*

**Proof.** We can build a schedule with makespan at most  $(4/3 + c)d$  using LPT-scheduling:

1. Sort the tasks  $j \in \mathcal{T}_g$  by descending processing times  $p_j$ .
2. In that order, assign each task to the machine with the smallest workload.

This algorithm runs in time  $\mathcal{O}(n \log n)$  since the runtime is dominated by sorting the tasks. Now, we bound the makespan of the resulting schedule by  $(4/3 + c)d$ .

With LPT, the tasks  $j \in \mathcal{T}_g$  of size  $\varphi(j) = 2$  get assigned first. Since by Constraint 3 there can be at most  $m_g$  such tasks in  $\mathcal{T}_g$ , every machine is assigned at most one task and therefore has a workload of at most  $d$ .

Next, the tasks  $j \in \mathcal{T}_g$  of size  $\varphi(j) = 1$  get assigned. Suppose task  $j \in \mathcal{T}_g$  is the first such task that is assigned to a machine with a load of more than  $(2/3)d$ . Then, by construction, all machines have a load of more than  $(2/3)d$ . If a machine has a load of more than  $(2/3)d$  while only tasks of size 2 or 1 are assigned to it, then at least one task with size 2 is assigned to it, or at least two tasks with size 1. So the total size on all machines is already at least  $2m_g$ , excluding task  $j$ . Including task  $j$  yields a contradiction to Constraint 3.

Thus, tasks with size 1 get assigned only to machines with a load of at most  $(2/3)d$ , and after assigning such a task to a machine, its load is at most  $(4/3)d$ .

Tasks  $j \in \mathcal{T}_g$  of size  $\varphi(j) = 0$  get assigned last. When some task  $j$  with size 0 gets assigned to some machine  $k \in [m_g]$ , such that its workload changes from less than  $(1 + c)d$  to more than  $(1 + c)d$ , then two properties hold. First, its workload is still at most  $(4/3 + c)d$ . Second, no other task will get assigned to machine  $k$ , before all machines have a workload of more than  $(1 + c)d$  because of the second step of the algorithm. If at any point in time all machines had a workload of more than  $(1 + c)d$ , though, then that state would contradict our assumption that  $\sum_{j \in \mathcal{T}_g} p_j \leq m_g(1 + c)d$ . Thus, no task with size 0 gets assigned to a machine with a workload of more than  $(1 + c)d$ .

In conclusion, no step of assigning a task to a machine will increase the workload of a machine to more than  $(4/3 + c)d$ , and thus the algorithm constructs a schedule with makespan at most  $(4/3 + c)d$ .  $\blacktriangleleft$



Now we can see that  $\delta = 1/6$  is the maximal constant such that Lemma 10 guarantees a schedule of length at most  $3d/2$ , when given a partition  $\mathcal{T}_g$  that satisfies Constraints 3 and 5.

To summarise we now define the MMCKP for CPU/GPU scheduling.

► **Definition 11** (MMCKP for CPU/GPU scheduling). *Let there be a CPU/GPU scheduling instance,  $d \in \mathbb{Q}_{>0}$  a makespan guess and  $\delta := 1/6$ . The goal of the Multidimensional Multiple Choice Knapsack Problem for CPU/GPU scheduling is find to find a partitioning  $\mathcal{T}_1 \sqcup \mathcal{T}_2 \sqcup \mathcal{T}_g = [n]$  that minimises under the Constraints 2, 3, and 5 the function*

$$W(\mathcal{T}_1, \mathcal{T}_2) := \sum_{j \in \mathcal{T}_1} w_j(\gamma_j(d)) + \sum_{j \in \mathcal{T}_2} w_j(\gamma_j(d/2)).$$

► **Remark 12.** As discussed previously, a solution of the MMCKP for CPU/GPU scheduling can be computed in time  $\mathcal{O}(nm_c 2m_g n / \delta) = \mathcal{O}(n^2 m_c m_g)$  by dynamic programming [14].

Now we can describe a 3/2-dual approximate algorithm for CPU/GPU scheduling, see Algorithm 1. We begin by constructing a partition  $\mathcal{T}_1 \sqcup \mathcal{T}_2 \sqcup \mathcal{T}_g = [n]$  using the multidimensional multiple choice knapsack algorithm in Remark 12. If we find a valid partition of the tasks with  $W(\mathcal{T}_1, \mathcal{T}_2) \leq m_c d$ , we use Lemma 5 to construct the schedule for the CPU. Next, we use Lemma 10 to build a schedule for the GPU. Finally, we combine both schedules, resulting in a CPU/GPU schedule.

On the other hand, if there is no valid partition found, we can reject the makespan guess  $d$  and return NO, as we know that no feasible schedule exists for it.

■ **Algorithm 1** Description of the 3/2-dual approximation algorithm.

---

**Require:** CPU/GPU scheduling instance  $I$

- 1: Construct  $\mathcal{T}_1 \sqcup \mathcal{T}_2 \sqcup \mathcal{T}_g = [n]$  using MMCKP ▷ Remark 12
  - 2: **if**  $W(\mathcal{T}_1, \mathcal{T}_2) \leq m_c d$  **then**
  - 3:     Build schedule  $\sigma_c$  for CPU from  $\mathcal{T}_1, \mathcal{T}_2$  ▷ Lemma 5
  - 4:     Build schedule  $\sigma_g$  for GPU from  $\mathcal{T}_g$  ▷ Lemma 10
  - 5:     Return schedule  $(\sigma_c, \sigma_g)$
  - 6: **else** Return NO
  - 7: **end if**
- 

We continue by showing the correctness of Algorithm 1.

► **Lemma 13.** *Let there be a CPU/GPU scheduling instance and  $d \in \mathbb{Q}_{>0}$  a makespan guess and  $\delta = 1/6$ . If a schedule of length at most  $d$  exists, then the 3/2-dual approximate algorithm returns a schedule of length at most  $3d/2$ .*

**Proof.** Consider a schedule of length at most  $d$ . The schedule implies a partitioning of tasks into those processed on the CPU with processing time more than  $d/2$  (and at most  $d$ ), those on the CPU with time at most  $d/2$ , and those processed on the GPU, called  $\mathcal{T}_1$ ,  $\mathcal{T}_2$ , and  $\mathcal{T}_g$  respectively. The schedule also implies a mapping  $\pi : \mathcal{T}_1 \cup \mathcal{T}_2 \rightarrow \mathbb{N}$  of tasks to the number of machines on the CPU they are processed on. Then, by Remark 4 the partitioning satisfies:

- $\sum_{j \in \mathcal{T}_1 \cup \mathcal{T}_2} w_j(\pi(j)) \leq m_c d$ ,
- $\sum_{j \in \mathcal{T}_1} \pi(j) \leq m_c$ ,

as well as Constraints 3 and 4. By definition we have  $\gamma_j(d) \leq \pi(j)$  for all  $j \in \mathcal{T}_1$  and  $\gamma_j(d/2) \leq \pi(j)$  for all  $j \in \mathcal{T}_2$ . Then, we get from the first point and the monotony of  $w_j$ , that Constraint 1 holds, and from the second point that Constraint 2, too. Finally, Constraint 5 holds, since it would otherwise violate Constraint 4 with Lemma 9.

## 13:10 Scheduling on CPU/GPU

In conclusion, the constructed partitioning is a feasible solution to the MMCKP for CPU/GPU scheduling with  $W(\mathcal{T}_1, \mathcal{T}_2) \leq m_c d$ , therefore the set of solutions is non-empty and the MMCKP algorithm delivers a feasible solution. Lemma 5 and 10 guarantee that a CPU/GPU schedule of length at most  $3d/2$  is returned. ◀

### 2.3 Optimizing the Algorithm

If we look closely, the MMCKP for CPU/GPU scheduling processes some unnecessary or redundant information:

1. For Constraint 3 the tasks  $j \in [n]$  with size  $\varphi(j) = 0$  are irrelevant.
2. In Definition 8 we chose  $\mu$  such that  $n\mu = \delta m_g d$ , because in the proof of Lemma 9 we can bound the number of tasks in  $\mathcal{T}_g$  only by  $n$ . But the number of tasks  $j \in [n]$  with size  $\varphi(j) \geq 1$  can be bound by  $2m_g$  due to Constraint 3. Thus, when considering only those tasks we could choose a scaling factor  $\lambda$  such that  $2m_g/\lambda = \delta m_g d \Leftrightarrow \lambda = 2/(\delta d)$ . When scaling Constraint 4 with  $\lambda$ , then its RHS becomes  $2m_g/\delta$ , which is smaller than  $n/\delta$  if  $2m_g < n$ .
3. The size function  $\varphi$  and Constraints 3 and 5 carry duplicate information about the processing time of tasks on the GPU. This suggests that we could reduce the RHS of the constraint for the total work on the GPU, if we cut out that redundant information.

The first two points suggest splitting the MMCKP for CPU/GPU scheduling into two stages, one for the tasks with size at least 1, and one for those with size 0. This justifies giving names to those classes of tasks. The third point can be treated separately.

► **Definition 14** (small and big tasks,  $\mathcal{B}$ ,  $\mathcal{S}$ ). *Let there be a CPU/GPU scheduling instance. Tasks  $j \in [n]$  with  $\varphi(j) \geq 1$  we call big (on the GPU), and those with  $\varphi(j) = 0$  we call small (on the GPU). Further we define the set of big tasks  $\mathcal{B} := \varphi^{-1}(\{1, 2, +\infty\})$ , and the set of small tasks  $\mathcal{S} := \varphi^{-1}(0)$ .*

#### Split MMCKP

► **Definition 15** ( $\lambda$ , Constraint 6). *Let there be a CPU/GPU scheduling instance,  $d \in \mathbb{Q}_{>0}$  a makespan guess,  $\mathcal{T}_g \subseteq [n]$ , and an error coefficient  $\delta \in \mathbb{Q}_{>0}$ . We define the scaling factor  $\lambda := \frac{2}{\delta d}$ , and the constraints*

$$6. \sum_{j \in \mathcal{T}_g} \lfloor \lambda p_j \rfloor \leq \lambda m_g d = 2m_g/\delta.$$

We want to split the knapsack into two stages, the first processing only big tasks  $j \in \mathcal{B}$  and the second only processing small tasks  $j \in \mathcal{S}$ . But we also need to compose them in some way, to ensure global optimality of the resulting partitioning, and not just locally for each set of big or small tasks. The dynamic programming algorithm for the MMCKP internally computes a table, where each entry encodes a solution for a smaller subproblem [14]. So we use the whole table that the big task knapsack computes as an input for the small task knapsack. Then the multiple-choice knapsack for the small tasks first has to choose a partitioning of the big tasks, among those encoded in that table.

Now we define the knapsack for big tasks. We will see later in Remark 19 how to choose the error coefficient  $\delta$ .

► **Definition 16** (Big Tasks MMCKP for CPU/GPU scheduling). *Let there be a CPU/GPU scheduling instance, a makespan guess  $d \in \mathbb{Q}_{>0}$ , and an error coefficient  $\delta \in \mathbb{Q}_{>0}$ . Let  $\mathcal{B} = \{b_1, \dots, b_K\}$  be an enumeration.*

One goal of the Big Tasks Multidimensional Multiple Choice Knapsack Problem for CPU/GPU scheduling is to compute a table  $W_{\mathcal{B}}^*$ , where for all  $(j, a, b, c) \in [K] \times [m_c]_0 \times [2m_g]_0 \times [2m_g/\delta]_0$  the entry  $W_{\mathcal{B}}^*(j, a, b, c)$  minimises the target function

$$W(\mathcal{T}_1, \mathcal{T}_2) := \sum_{j \in \mathcal{T}_1} w_j(\gamma_j(d)) + \sum_{j \in \mathcal{T}_2} w_j(\gamma_j(d/2))$$

over all partitions  $\mathcal{T}_1 \sqcup \mathcal{T}_2 \sqcup \mathcal{T}_g = \{b_1, \dots, b_j\}$  such that the left-hand-sides of Constraints 2, 3, and 6 are equal to  $a$ ,  $b$ , and  $c$  respectively.

Another goal is to compute a minimizing partitioning given such a tuple  $(j, a, b, c)$ .

$W_{\mathcal{B}}^*(K, a, b, c)$  encodes a partitioning of  $\mathcal{B}$  that satisfies above properties. The knapsack for small tasks first chooses one of those entries. The total work on the GPU, encoded in term  $c$  by the big tasks knapsack is scaled by factor  $\lambda$ , but the small tasks knapsack uses the scaling factor  $\mu$ . Thus we need to re-scale that information to use it in the constraint for the small tasks knapsack:

► **Definition 17** (Small Tasks MMCKP for CPU/GPU scheduling). *Let there be a CPU/GPU scheduling instance, a makespan guess  $d \in \mathbb{Q}_{>0}$ , and an error coefficient  $\delta \in \mathbb{Q}_{>0}$ . Let the table  $W_{\mathcal{B}}^*$  be a solution to the big tasks MMCKP for CPU/GPU scheduling. The goal of the Small Tasks Multidimensional Multiple Choice Knapsack Problem for CPU/GPU scheduling is to choose a triple  $(a, b, c) \in [m_c]_0 \times [2m_g]_0 \times [2m_g/\delta]_0$  and a partitioning  $\mathcal{T}_1 \sqcup \mathcal{T}_2 \sqcup \mathcal{T}_g = \mathcal{S}$  that minimizes*

$$W_{\mathcal{S}}((a, b, c), \mathcal{T}_1, \mathcal{T}_2) := W_{\mathcal{B}}^*(K, a, b, c) + \sum_{j \in \mathcal{T}_1} w_j(\gamma_j(d)) + \sum_{j \in \mathcal{T}_2} w_j(\gamma_j(d/2))$$

under the following modifications of Constraints 2 and 5:

- $a + \sum_{j \in \mathcal{T}_1} \gamma_j(d) \leq m_c$ , and
- $\lfloor \frac{\mu}{\lambda} c \rfloor + \sum_{j \in \mathcal{T}_g} \tilde{p}_j \leq n/\delta$ .

Then, in total, we compute a partitioning  $\mathcal{T}_1 \sqcup \mathcal{T}_2 \sqcup \mathcal{T}_g = [n]$  by first computing the table  $W_{\mathcal{B}}^*$  from the big tasks MMCKP, then with that table we compute a triple  $(a, b, c)$  and a partitioning  $\mathcal{T}'_1 \sqcup \mathcal{T}'_2 \sqcup \mathcal{T}'_g = \mathcal{S}$  from the small tasks MMCKP, and then with the tuple  $(K, a, b, c)$ , where  $K = |\mathcal{B}|$ , we compute a partitioning  $\mathcal{T}''_1 \sqcup \mathcal{T}''_2 \sqcup \mathcal{T}''_g = \mathcal{B}$  from the big tasks MMCKP. Finally we set  $\mathcal{T}_1 := \mathcal{T}'_1 \cup \mathcal{T}''_1$ ,  $\mathcal{T}_2 := \mathcal{T}'_2 \cup \mathcal{T}''_2$ , and  $\mathcal{T}_g := \mathcal{T}'_g \cup \mathcal{T}''_g$ .

We still need to show that the scaled and rounded constraint for the total work on the GPU in the small tasks MMCKP implies a good bound on the actual work area, as we did in the previous section.

► **Lemma 18.** *Let there be a CPU/GPU scheduling instance, a makespan guess  $d \in \mathbb{Q}_{>0}$ , and an error coefficient  $\delta \in \mathbb{Q}_{>0}$ . Let  $\mathcal{T}_b \subseteq \mathcal{B}$  such that Constraints 3 and 6 hold. Let  $\mathcal{T}_s \subseteq \mathcal{S}$  such that  $\lfloor \frac{\mu}{\lambda} \sum_{j \in \mathcal{T}_b} \lfloor \lambda p_j \rfloor \rfloor + \sum_{j \in \mathcal{T}_s} \tilde{p}_j \leq n/\delta$  holds. Then  $\sum_{j \in \mathcal{T}_b \cup \mathcal{T}_s} p_j \leq m_g(1 + 2\delta + \delta/n)d$ .*

**Proof.** (See the Appendix.) ◀

► **Remark 19.** Lemma 18 shows that if we set the error coefficient to  $\delta := 1/\lfloor 12 + \frac{6}{n} \rfloor$ , then Lemma 10 builds a schedule of length at most  $3d/2$  from the set of tasks  $\mathcal{T}_g := \mathcal{T}_b \cup \mathcal{T}_s$ .

We skip the correctness proof for the dual approximation that uses the split MMCKP, as it works with arguments analogous to those in Lemma 13.

### Cutting redundant information

In this section, we show how to decrease the RHS of the constraint that represents the total work area on the GPU in the big tasks MMCKP, which improves the time complexity of solving it. First, we describe how to modify the knapsack to achieve this, and then analyze the time savings.

► **Definition 20** (Modified Big and Small Tasks MMCKP for CPU/GPU scheduling,  $\beta$ ). *The modified big tasks MMCKP for CPU/GPU scheduling is the unmodified one, except with the following modifications. First, define  $\beta := \lfloor \lambda d/3 \rfloor$ , which is the scaled and rounded work area represented by one size step of  $\varphi$ .*

- *The table is defined for all tuples  $(j, a, b, c)$  such that  $(j, a, b) \in [K] \times [m_c]_0 \times [2m_g]_0$  and  $c \in [2m_g/\delta - b\beta]_0$ .*
- *Instead of Constraint 6 we use the constraint  $\sum_{j \in \mathcal{T}_g} \lfloor \lambda p_j \rfloor - \varphi(j)\beta = c$*

*The modified small tasks MMCKP for CPU/GPU scheduling is modified such that*

- *We choose a triple  $(a, b, c)$  with  $(a, b) \in [m_c]_0 \times [2m_g]_0$  and  $c \in [2m_g/\delta - b\beta]_0$  instead.*
- *We use the constraint  $\lfloor \frac{\mu}{\lambda}(c + \beta b) \rfloor + \sum_{j \in \mathcal{T}_g} \tilde{p}_j \leq n/\delta$  as the modification of Constraint 5.*

This modification decreases the size of the table  $W_{\mathcal{B}}^*$  to almost 2/3 of its original size. While this does not improve the time complexity in  $\mathcal{O}$ -notation, this is a significant speedup for implementations of the algorithm, since the main computational cost of solving the knapsack by dynamic programming lies in computing every entry of the table. In the next lemma, we prove the stated speedup factor.

► **Lemma 21.** *Let there be a CPU/GPU scheduling instance. Let  $W^*$  the table from the big tasks MMCKP, and  $X^*$  from the modified one. Let the mapping  $|\cdot|$  count their number of entries. Then  $\frac{|X^*|}{|W^*|} \leq \frac{2}{3}(1 + \delta)$ .*

**Proof.** (For a more detailed calculation see the Appendix.) Let  $K := |\mathcal{B}|$ . We have  $|W^*| = K(m_c+1)(2m_g+1)(2m_g/\delta+1)$ . Further, we have  $|X^*| = K(m_c+1)(2m_g+1)(2m_g/\delta+1-\beta m_g)$ . Now, with  $\beta = \lfloor 2/(3\delta) \rfloor \geq 2/(3\delta) - 1$  we get

$$(2m_g/\delta + 1 - \beta m_g) \leq (2/3)(2m_g/\delta + 1) + (1/3) + m_g.$$

Then one can show, that  $\frac{|X^*|}{|W^*|} \leq \frac{2}{3} + \frac{4}{3} \cdot \frac{1}{2\delta} = \frac{2}{3}(1 + \delta)$ . ◀

## 3 Complexity

We showed that we can compute a bound  $\omega$  on the optimal makespan  $d^*$  such that such that  $\omega \leq d^* \leq 4\omega$  in time  $\mathcal{O}((m_c n + m_g) \log(m_c n + m_g))$ .

The complexity of the 3/2-dual approximation for both, the unmodified and the modified version, is dominated by the complexity of solving their knapsack problems. Solving the unmodified MMCKP is in  $\mathcal{O}(n^2 m_c m_g)$ , see Remark 12. Solving the modified MMCKP for big tasks is in  $\mathcal{O}(n m_c m_g^2)$ , for small tasks in  $\mathcal{O}(n^2 m_c)$ , so solving both is in  $\mathcal{O}(n m_c \cdot \max(n, m_g^2))$ .

Therefore, with Lemma 2, computing a  $(3/2 + \varepsilon)$ -approximation for the CPU/GPU scheduling is in total

$$\mathcal{O}((m_c n + m_g) \log(m_c n + m_g) + \log(1/\varepsilon) n^2 m_c m_g \subseteq \mathcal{O}(\log(n m_c m_g / \varepsilon) \cdot n^2 m_c m_g)$$

for the unmodified version, and  $\mathcal{O}(\log(m_c n m_g / \varepsilon) \cdot n m_c \cdot \max(n, m_g^2))$  for the modified version. ◀

## 4 Implementation

To complement our theoretical improvements on the current state of the art, we implemented both the naive and improved version of our algorithm. In the following, we describe the implementation and compare these implementations not only to each other but also to the current state of the art. The implementation is in Java and publicly available on GitHub (<https://github.com/Felioh/CPU-GPU-Scheduling>) as well as the test results.

### 4.1 Test Environment

The experiments were run on the High-Performance-Cluster of the University of Kiel, more specifically on a node with 2 AMD Epyc 7313 CPUs (32 Cores) and 512GB main memory. We disabled the Java Garbage Collector, as the default is non-deterministic. It is important to note that obtaining meaningful results can be challenging due to the various parameters involved, including the number of CPU machines, the number of GPU machines, the number of tasks, and the processing time required for each task on both the CPU and GPU machines. We primarily used randomly generated instances to ensure the robustness of the results across a large number of instances.

### 4.2 Results

We conducted two sets of tests: one to demonstrate the dependence on machine numbers and the other to show the dependence on task numbers. For both test runs we set  $\varepsilon = 0.1$  and the processing time of all tasks to be randomly distributed in the interval  $[20, 100]$ . The processing time of GPU tasks on one machine is generated in the same way. The processing time for more machines is generated by respecting the given constraints. Additionally, the number of GPU machines was always set to 20% of the number of CPU machines as we believe this relationship could significantly impact the running time. All test results are available in the corresponding GitHub Repository.

To test the effect of the number of machines on our instances, we created instances with a minimum of 10 and a maximum of 100 CPU machines. To maintain consistency, we restricted the number of tasks to 50. The results of these tests are shown in Figure 2.

To examine how the number of tasks affects the total runtime of our algorithm, we generated instances with 50 CPU machines and the number of tasks varying from 10 to 100. The results of these tests are shown in Figure 3.

The results indicate a greater improvement than expected, likely due to inefficiencies in memory accesses caused by the naive implementation's increased memory requirements.

Table 1 shows the comparison of our implementation and the implementation by Bleuse et al. [1]. To achieve these results we tested both implementations with the same input instances. We can observe that our implementation is significantly slower for small instances. As the instances get bigger our implementation catches up with the trade-off of using more memory. The instances used for these tests and the full results are available on GitHub.

## 5 Conclusion

We presented an efficient algorithm to approximate the maximum makespan of the GPU/CPU scheduling problem with a factor of  $(3/2 + \varepsilon)$ , using the dual approximation framework. We improve the asymptotic running time of the dual approximation from a theorized [1] time of  $\mathcal{O}(n^2 m_c^4 m_g^2)$  to  $\mathcal{O}(n^2 m_c m_g)$ . Impressively, we manage to reduce the dependency on the number of CPU-machines by  $m_c^3$ , while also reducing the dependence on GPU-machines by

$m_g$ . Our implementation utilizes a knapsack formulation to assign tasks. Solving this knapsack dominates the running time. To improve it, we observe that large parts of information are superfluous. This does not strictly improve the asymptotic running time, resulting in a time of  $\mathcal{O}(nm_c \cdot \max(n, m_g^2))$  for the dual approximation. However, we show that these practical improvements have a large impact on the execution time of the algorithm.

In future work, the runtime could be further improved by applying compression techniques from [6], using a more sophisticated algorithm to solve the MMCKP, or using two different algorithms depending on the sizes of  $m_c, m_g$ .

Further, we compare our implementation to another existing state of the art algorithm [1], that makes use of commercial solvers like CPLEX, Gurobi, or GLPK and show that our implementation is faster for big instances while keeping the same makespan guarantee.

---

## References

- 1 Raphaël Bleuse, Sascha Hunold, Safia Kedad-Sidhoum, Florence Monna, Gregory Mounie, and Denis Trystram. Scheduling independent moldable tasks on multi-cores with gpus. *IEEE Trans. Parallel Distributed Syst.*, 28(9):2689–2702, 2017. doi:10.1109/TPDS.2017.2675891.
- 2 Raphaël Bleuse, Safia Kedad-Sidhoum, Florence Monna, Grégory Mounié, and Denis Trystram. Scheduling independent tasks on multi-cores with GPU accelerators. *Concurr. Comput. Pract. Exp.*, 27(6):1625–1638, 2015. doi:10.1002/CPE.3359.
- 3 Maciej Drozdowski. On the complexity of multiprocessor task scheduling. *Bulletin of The Polish Academy of Sciences-technical Sciences*, 43:381–392, 1995.
- 4 M. R. Garey and Ronald L. Graham. Bounds for multiprocessor scheduling with resource constraints. *SIAM J. Comput.*, 4(2):187–200, 1975. doi:10.1137/0204015.
- 5 Bernhard Sebastian Germann, Klaus Jansen, Felix Ohnesorge, and Malte Tutas. Felioh/CPU-GPU-Scheduling. Software, DFG-Project JA 612/25-1, swId: swh:1:dir:6b62fa88e3d3dde792b3c7a9803a04ef24651239 (visited on 2024-07-01). URL: <https://github.com/Felioh/CPU-GPU-Scheduling>.
- 6 Kilian Grage, Klaus Jansen, and Felix Ohnesorge. Improved algorithms for monotone moldable job scheduling using compression and convolution. In José Cano, Marios D. Dikaiakos, George A. Papadopoulos, Miquel Pericàs, and Rizos Sakellariou, editors, *Euro-Par 2023: Parallel Processing - 29th International Conference on Parallel and Distributed Computing, Limassol, Cyprus, August 28 - September 1, 2023, Proceedings*, volume 14100 of *Lecture Notes in Computer Science*, pages 503–517. Springer, 2023. doi:10.1007/978-3-031-39698-4.
- 7 Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal of Applied Mathematics*, 17(2):416–429, 1969.
- 8 Dorit S. Hochbaum and David B. Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *J. ACM*, 34(1):144–162, 1987. doi:10.1145/7531.7535.
- 9 Klaus Jansen. An EPTAS for scheduling jobs on uniform processors: Using an MILP relaxation with a constant number of integral variables. In *ICALP (1)*, volume 5555 of *Lecture Notes in Computer Science*, pages 562–573. Springer, 2009.
- 10 Klaus Jansen and Felix Land. Scheduling monotone moldable jobs in linear time. In *2018 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2018, Vancouver, BC, Canada, May 21-25, 2018*, pages 172–181. IEEE Computer Society, 2018. doi:10.1109/IPDPS.2018.00027.
- 11 Klaus Jansen, Felix Land, and Kati Land. Bounding the running time of algorithms for scheduling and packing problems. *SIAM J. Discret. Math.*, 30(1):343–366, 2016. doi:10.1137/140952636.
- 12 Klaus Jansen and Ralf Thöle. Approximation algorithms for scheduling parallel jobs. *SIAM J. Comput.*, 39(8):3571–3615, 2010.

- 13 Edward G. Coffman Jr., M. R. Garey, and David S. Johnson. An application of bin-packing to multiprocessor scheduling. *SIAM J. Comput.*, 7(1):1–17, 1978.
- 14 Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack problems*. Springer, 2004. doi:10.1007/978-3-540-24777-7.
- 15 Jan Karel Lenstra and David B. Shmoys. Elements of scheduling. *CoRR*, abs/2001.06005, 2020. arXiv:2001.06005.
- 16 Walter Ludwig and Prason Tiwari. Scheduling malleable and nonmalleable parallel tasks. In Daniel Dominic Sleator, editor, *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms. 23-25 January 1994, Arlington, Virginia, USA*, pages 167–176. ACM/SIAM, 1994. URL: <http://dl.acm.org/citation.cfm?id=314464.314491>.
- 17 Gregory Mounie, Christophe Rapine, and Denis Trystram. A 3/2-approximation algorithm for scheduling independent monotonic malleable tasks. *SIAM J. Comput.*, 37(2):401–412, 2007. doi:10.1137/S0097539701385995.
- 18 John Turek, Joel L. Wolf, and Philip S. Yu. Approximate algorithms scheduling parallelizable tasks. In Lawrence Snyder, editor, *Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA '92, San Diego, CA, USA, June 29 - July 1, 1992*, pages 323–332. ACM, 1992. doi:10.1145/140901.141909.
- 19 Fangfang Wu, Xiandong Zhang, and Bo Chen. An improved approximation algorithm for scheduling monotonic moldable tasks. *Eur. J. Oper. Res.*, 306(2):567–578, 2023.

## A

 Proofs

**Proof of Lemma 3.** Let  $t_j : [m_c] \rightarrow \mathbb{Q}_{>0}$  be the processing time function,  $w_j$  the work area function on the CPU, and  $p_j$  the processing time on the GPU for each task  $j \in [n]$ .

Now we define the mapping  $\omega : [m_c]^{[n]} \times \mathcal{P}([n]) \rightarrow \mathbb{Q}_{>0}$  by

$$\omega(\alpha, A) := \max \left\{ \frac{1}{m_c} \sum_{j \in A} w_j(\alpha(j)), \max_{j \in A} \{t_j(\alpha(j))\}, \frac{1}{m_g} \sum_{j \in A^C} p_j, \max_{j \in A^C} \{p_j\} \right\}.$$

There

- $A \subseteq [n]$  represents the tasks scheduled on the CPU and its complement  $A^C = [n]/A$  on the GPU,
- the mapping  $\alpha : A \rightarrow [m_c]$  assigns to each task a number of CPU machines,
- $\frac{1}{m_c} \sum_{j \in A} w_j(\alpha(j))$  is the average work per machine on the CPU,
- $\max_{j \in A} \{t_j(\alpha(j))\}$  is the maximum processing time of any task on the CPU,
- $\frac{1}{m_g} \sum_{j \in A^C} p_j$  is the average work per machine on the GPU, and
- $\max_{j \in A^C} \{p_j\}$  is the maximum processing time of any task on the GPU.

Given an assignment  $\alpha$  and a partition  $A \cup A^C = [n]$ , the 2-approximate algorithm by Garey and Graham yields a CPU/GPU schedule with a makespan that does not exceed  $2\omega(\alpha, A)$  [4]. This suggests the following strategy to find the desired lower bound: Find an assignment and partition such that  $\omega(\alpha, A) \leq d^* \leq 2\omega(\alpha, A)$ .

Now,  $\omega$  defined by

$$\omega := \min_{A \subseteq [n]} \min_{\alpha: [n] \rightarrow [m_c]} \omega(\alpha, A)$$

is a trivial lower bound on the minimum makespan for CPU/GPU scheduling, and, by Garey and Graham's algorithm, satisfies  $\omega \leq d^* \leq 2\omega$ . Next, we will derive an efficient way to compute  $\omega$ .

## 13:16 Scheduling on CPU/GPU

Let  $H := \{p_j : j \in [n]\} \cup \{t_j(p) : j \in [n], p \in [m_c]\}$  be the set of processing times of tasks on the CPU/GPU. Further, let  $h_c(\alpha, A) := \max_{j \in A} t_j(\alpha(j))$ , and  $h_g(A^C) := \max_{j \in A^C} p_j$  be the maximum processing times on CPU and GPU respectively, given an assignment  $\alpha : A \rightarrow [m_c]$  and a subset  $A \subseteq [n]$ . Then we can rewrite  $\omega$  as follows:

$$\begin{aligned}
\omega &= \min_{A \subseteq [n]} \min_{\alpha: A \rightarrow [m_c]} \omega(\alpha, A) \\
&= \min_{A \subseteq [n]} \min_{\alpha: A \rightarrow [m_c]} \max \left\{ \frac{1}{m_c} \sum_{i \in A} w_j(\alpha(j)), \max_{j \in A} \{t_j(\alpha(j))\}, \frac{1}{m_g} \sum_{j \in A^C} p_j, \max_{j \in A^C} \{p_j\} \right\} \\
&= \min_{\tau \in H} \min_{\substack{A \subseteq [n] \\ \text{s.t. } h_g(A^C) \leq \tau}} \min_{\substack{\alpha: A \rightarrow [m_c] \\ \text{s.t. } h_c(\alpha, A) \leq \tau}} \max \left\{ \tau, \frac{1}{m_c} \sum_{j \in A} w_j(\alpha(j)), \frac{1}{m_g} \sum_{j \in A^C} p_j \right\} \\
&= \min_{\tau \in H} \min_{\substack{A \subseteq [n] \\ \text{s.t. } h_g(A^C) \leq \tau}} \max \left\{ \tau, \frac{1}{m_c} \min_{\substack{\alpha: A \rightarrow [m_c] \\ \text{s.t. } h_c(\alpha, A) \leq \tau}} \sum_{j \in A} w_j(\alpha(j)), \frac{1}{m_g} \sum_{j \in A^C} p_j \right\} \\
&= \min_{\tau \in H} \min_{\substack{A \subseteq [n] \\ \text{s.t. } h_g(A^C) \leq \tau}} \max \left\{ \tau, \frac{1}{m_c} \sum_{j \in A} w_j(\gamma_j(\tau)), \frac{1}{m_g} \sum_{j \in A^C} p_j \right\} \\
&= \min_{\tau \in H} \max \left\{ \tau, \min_{\substack{A \subseteq [n] \\ \text{s.t. } h_g(A^C) \leq \tau}} \max \left\{ \sum_{j \in A} \frac{w_j(\gamma_j(\tau))}{m_c}, \sum_{j \in A^C} \frac{p_j}{m_g} \right\} \right\}.
\end{aligned}$$

Here, computing the term

$$W(\tau) := \min_{\substack{A \subseteq [n] \\ \text{s.t. } h_g(A^C) \leq \tau}} \max \left\{ \sum_{j \in A} \frac{w_j(\gamma_j(\tau))}{m_c}, \sum_{j \in A^C} \frac{p_j}{m_g} \right\}$$

for some  $\tau \in H$  can be interpreted as minimizing the maximum makespan of two unrelated machines ( $R2||C_{\max}$ ), where  $\frac{w_j(\gamma_j(\tau))}{m_c}$  is the processing time for task  $j$  on the first machine, and  $\frac{p_j}{m_g}$  for the second machine, with the additional constraint that tasks  $j \in [n]$  with  $p_j > \tau$  must be assigned to the first machine, which can be modeled by setting their processing time for the second machine to  $+\infty$ .

Now note that the identity mapping  $\tau \mapsto \tau$  is increasing, while  $\tau \mapsto W(\tau)$  is non-increasing. So one can find  $\min_{\tau \in H} \max\{\tau, W(\tau)\}$  by binary searching over the sorted processing times in  $H$ , as described by Ludwig and Tiwari in [16].

The problem here is that we can not compute  $W(\tau)$  in polynomial time, since makespan minimization for two identical machines is already NP-complete [15]. But the greedy algorithm, that assigns each task to the machine on which it takes the least time, is a 2-approximate algorithm for  $R2||C_{\max}$  [15].

When using a 2-approximation  $\widetilde{W}(\tau)$  with  $W(\tau) \leq \widetilde{W}(\tau) \leq 2W(\tau)$ , the binary search yields a 2-approximation  $\widetilde{\omega}$  of  $\omega$  with  $\omega \leq \widetilde{\omega} \leq 2\omega$ . Thus

$$\widetilde{\omega}/2 \leq \omega \leq d^* \leq 2\omega \leq 2\widetilde{\omega}.$$

So  $\widetilde{\omega}$  satisfies our criteria.

Finally, we analyze the complexity of computing  $\widetilde{\omega}$ . Sorting  $H$  is in  $\mathcal{O}((m_c n + m_g) \log(m_c n + m_g))$ . The binary search takes  $\mathcal{O}(\log(m_c n + m_g))$  steps, each step costing  $\mathcal{O}(n)$  to solve the unrelated machine scheduling and  $\mathcal{O}(n \log m_c)$  to find  $\gamma_j(\tau)$  for each task  $j$  via binary search. Therefore the overall runtime is dominated by sorting  $H$ .  $\blacktriangleleft$



**Proof of Lemma 18.** The definition of  $\tilde{p}_j$  yields  $\tilde{p}_j/\mu \leq p_j \leq (\tilde{p}_j + 1)/\mu$  for all  $j \in \mathcal{T}_s$ , and analogously we also have  $\lfloor \lambda p_j \rfloor / \lambda \leq p_j \leq (\lfloor \lambda p_j \rfloor + 1) / \lambda$  for all  $j \in \mathcal{T}_b$ . Note, that from Constraint 3 follows, that  $|\mathcal{T}_b| \leq 2m_g$ . Then:

$$\begin{aligned}
\sum_{j \in \mathcal{T}_b \cup \mathcal{T}_s} p_j &= \sum_{j \in \mathcal{T}_b} p_j + \sum_{j \in \mathcal{T}_s} p_j \\
&\leq \left( \frac{1}{\lambda} \sum_{j \in \mathcal{T}_b} \lfloor \lambda p_j \rfloor \right) + \frac{2m_g}{\lambda} + \left( \frac{1}{\mu} \sum_{j \in \mathcal{T}_s} \tilde{p}_j \right) + \frac{n}{\mu} \\
&\leq \frac{1}{\mu} \left( \frac{\mu}{\lambda} \sum_{j \in \mathcal{T}_b} \lfloor \lambda p_j \rfloor + \sum_{j \in \mathcal{T}_s} \tilde{p}_j \right) + \frac{2m_g}{\lambda} + \frac{n}{\mu} \\
&\leq \frac{1}{\mu} \left( 1 + \left\lfloor \frac{\mu}{\lambda} \sum_{j \in \mathcal{T}_b} \lfloor \lambda p_j \rfloor \right\rfloor + \sum_{j \in \mathcal{T}_s} \tilde{p}_j \right) + \frac{2m_g}{\lambda} + \frac{n}{\mu} \\
&\leq \frac{1}{\mu} + \frac{n}{\mu\delta} + \frac{2m_g}{\lambda} + \frac{n}{\mu} \\
&= \frac{m_g\delta d}{n} + \frac{nm_g\delta d}{n\delta} + \frac{2m_g\delta d}{2} + \frac{nm_g\delta d}{n} \\
&= m_g(1 + 2\delta + \frac{\delta}{n})d \quad \blacktriangleleft
\end{aligned}$$

**Proof of Lemma 21.** Let  $K := |\mathcal{B}|$ . We have  $|W^*| = K(m_c + 1)(2m_g + 1)(2m_g/\delta + 1)$ . Further, we have

$$\begin{aligned}
|X^*| &= K(m_c + 1) \sum_{0 \leq b \leq 2m_g} (2m_g/\delta + 1 - b\beta) \\
&= K(m_c + 1) \left( (2m_c + 1)(2m_g/\delta + 1) - \beta \sum_{1 \leq b \leq 2m_g} b \right) \\
&= K(m_c + 1)(2m_g + 1)(2m_g/\delta + 1 - \beta m_g)
\end{aligned}$$

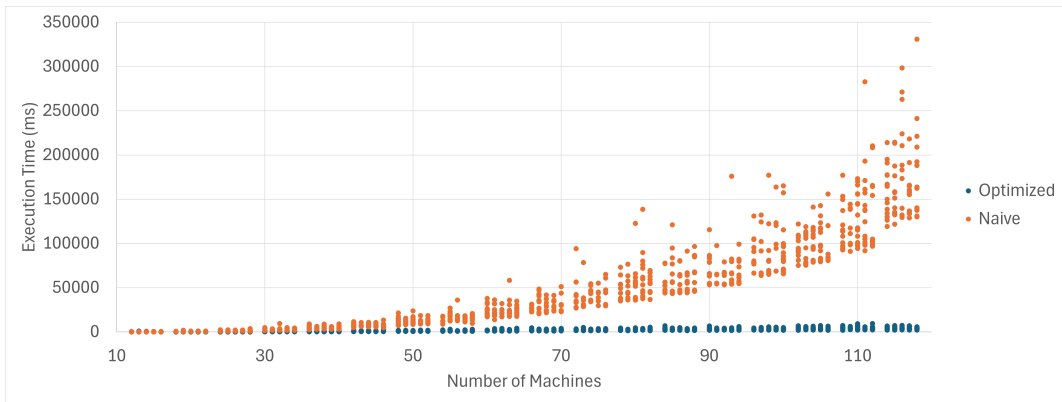
Now, with  $\beta = \lfloor 2/(3\delta) \rfloor \geq 2/(3\delta) - 1$  we get

$$(2m_g/\delta + 1 - \beta m_g) \leq \frac{2}{3}(2m_g/\delta + 1) + \frac{1}{3} + m_g.$$

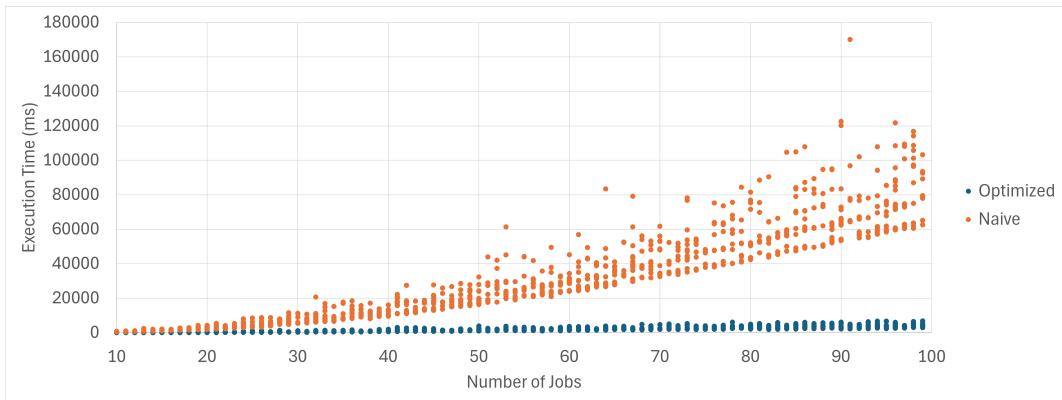
Concluding:

$$\begin{aligned}
\frac{|X^*|}{|W^*|} &= \frac{2m_g/\delta + 1 - \beta m_g}{2m_g/\delta + 1} \\
&\leq \frac{2}{3} + \frac{1}{3} \cdot \frac{1}{2m_g/\delta + 1} + \frac{m_g}{2m_g/\delta + 1} \\
&\leq \frac{2}{3} + \frac{4}{3} \cdot \frac{1}{2/\delta} = \frac{2}{3}(1 + \delta). \quad \blacktriangleleft
\end{aligned}$$

**B** Figures and Tables



■ **Figure 2** Runtime in relation to machine numbers.



■ **Figure 3** Runtime in relation to task numbers.

■ **Table 1** Comparison of makespan and runtime with [1] for  $100 \leq n \leq 400$  and  $\epsilon = 0.1$ .

$m_c$	$m_g$	$n$	our time [s]	time of [1] [s]
100	20	100	19.7	1.0
100	20	150	28.8	2.1
100	20	200	41.7	20.9
100	20	250	67.1	18.2
100	20	300	95.6	32.4
100	20	350	93.6	183.0
100	20	400	127.6	199.8