Online Scheduling of Moldable Task Graphs under Common Speedup Models

Anne Benoit, Lucas Perotin, Yves Robert, Hongyang Sun

Online Scheduling of Moldable Task Graphs under Common Speedup Models

Anne Benoit *Senior Member, IEEE,* Lucas Perotin, Yves Robert *Fellow, IEEE,*
Hongyang Sun *Senior Member, IEEE*

Abstract—The problem of scheduling moldable tasks has been widely studied, in particular when tasks have dependencies (i.e., task graphs), or when tasks are released on-the-fly (i.e., online). However, few study has focused on both (i.e., online scheduling of moldable task graphs). In this paper, we derive constant competitive ratios for this problem under several common yet realistic speedup models for the tasks (roofline, communication, Amdahl, and a combination of them). We also provide the first lower bound on the competitive ratio of any deterministic online algorithm for arbitrary speedup model, which is not constant but depends on the number of tasks in the longest path of the graph.

1 Introduction

This work investigates the online scheduling of parallel task graphs, where each task in the graph is *moldable*. In the scheduling literature, a moldable task (or job) is a parallel task that can be executed on an arbitrary but fixed number of processors. The execution time of the task depends upon the number of processors chosen to execute it. This number of processors is chosen once and for all, when the task starts its execution, and cannot be modified later on during execution. This corresponds to a variable static resource allocation, as opposed to a fixed static allocation (*rigid* tasks) and to a variable dynamic allocation (*malleable* tasks) [7].

Moldable tasks offer a nice trade-off between rigid and and malleable tasks: they easily adapt to the number of available resources, contrarily to rigid tasks, while being easy to design and implement, contrarily to malleable tasks. This explains that many computational kernels in scientific libraries for numerical linear algebra and tensor computations are provided as moldable tasks that can be deployed on a wide range of processor numbers. We assume that the scheduling of each task is non-preemptive and without restarts [8], which is a highly desirable approach to avoid high overheads incurred by checkpointing partial results, context switching, and task migration.

Because of the importance and wide availability of moldable tasks, scheduling algorithms for such tasks have received considerable attention. The scheduling problem, whose objective is to minimize the overall completion time, or makespan, comes in many flavors:

Offline vs. online. In the offline version of the problem, all tasks are known in advance, before the execution starts. The problem is NP-complete, and the goal is to derive lower bounds and approximation algorithms. On the contrary, in the online version of the problem, tasks are released on the fly, and the objective is to derive competitive ratios [17] for the performance of a scheduling algorithm against an optimal offline scheduler, which knows in advance all the tasks and and their dependencies in the graph. The competitive ratio is established against all possible strategies devised by an adversary trying to force the online algorithm to take *bad* decisions.

Independent tasks vs. task graphs. There are two versions of the online problem, with independent tasks or with task graphs. For the version with independent tasks, the tasks are released on the fly and the scheduler discovers their characteristics only upon release. For the version with task graphs, the whole graph is released at the start, but the scheduler discovers a new task and its characteristics only when all of its predecessors have completed execution. In other words, the shape of the graph and the nature of the tasks are not known in advance and are revealed only as the execution progresses.

In this work, we investigate the most difficult instance of the problem, namely, the online scheduling of moldable tasks graphs. Our main contribution resides in several new competitive ratios, which greatly depend upon the speedup model of the tasks. Several common yet realistic speedup models have been introduced and analyzed, including the roofline model, the communication model, the Amdahl's model, and a general combination of them (see Section 3.1 for definitions). We provide a constant competitive ratio for each of these four models. In addition, we derive a new lower bound on the competitiveness of any deterministic online algorithm under the arbitrary speedup model. To the best of our knowledge, a competitive ratio was only known for task graphs under the roofline model [8], and we extend the result to several other speedup models.

The rest of this paper is organized as follows. Section 2 surveys related work. The formal model and problem statement are presented in Section 3. Section 4 is the heart of the paper: we introduce the new online algorithm and prove

• *Anne Benoit, Lucas Perotin and Yves Robert are with the LIP laboratory at Ecole Normale Supérieure de Lyon, France. Yves Robert is also with the University of Tennessee Knoxville, USA. Hongyang Sun is with the University of Kansas, USA. Contact: yves.robert@ens-lyon.fr*

its competitive ratios for the different speedup models. Section 5 is devoted to the lower bound for arbitrary speedup models. Finally, Section 6 concludes the paper and provides hints for future directions.

2 Related Work

Several prior studies have considered offline scheduling of independent moldable tasks, and derived approximation results. While some results depend on specific speedup models for the tasks, other results hold for the arbitrary model. Turek et al. [18] designed a 2-approximation list-based algorithm for the arbitrary model. Furthermore, when each task only admits a subset of all possible processor allocations, Jansen [10] presented a $(1.5+\epsilon)$-approximation algorithm, which is tight since it was also shown that the problem cannot have an approximation ratio better than 1.5 unless $\mathcal{P} = \mathcal{NP}$ [14]. For the monotonic model, where the execution time is non-increasing and the area (processor allocation times execution time) is non-decreasing with the number of processors, Jansen and Land [11] further proposed a polynomial-time approximation scheme (PTAS).

For online scheduling of independent moldable tasks that are released on-the-fly, Ye et al. [21] designed a 16.74-competitive algorithm. They also explained how to transform an algorithm for rigid tasks whose makespan is at most ρ times the lower bound into a 4ρ competitive algorithm for moldable tasks. Further, some algorithms designed in the offline setting will also work online if they make scheduling decisions independently for each task; see for instance [6], [9], [15], which studied the communication model.

For offline scheduling of moldable tasks with dependencies, Wang and Cheng [19] showed that the earliest completion time algorithm is a $(3-2/P)$-approximation for the roofline model. For the monotonic model, Lepère et al. [16] proposed an algorithm with approximation ratio $3+\sqrt{5}$, which was later improved to 4.73 by Jansen and Zhang [13]. Chen and Chu [5] further proposed improved approximations for a more restrictive model, where the area is a concave function and the execution time is strictly decreasing with the number of processors.

Feldmann et al. [8] designed an online algorithm for moldable tasks with dependencies, under the roofline model. By keeping the system utilization above a given bound and by carefully tuning of this bound, their algorithm achieves 2.618-competitiveness, even when the task execution times and the DAG structure are unknown. Canon et al. [4] focused on hybrid platforms with several types of processors (for instance, CPUs and GPUs), and derived competitive ratios depending on the number of such resources, but they did not consider moldable tasks.

We have recently investigated the problem of scheduling independent moldable tasks subject to failures [3], where tasks need to be re-executed after a failure until a successful completion. This corresponds to a semi-online setting, since all tasks are known at the beginning, but failed tasks are only discovered on-the-fly. Although we do not consider task failures in this paper, but rather focus on the general online scheduling of moldable task graphs (as in [8]), the results can readily carry over to the failure scenario.

Table 1 summarizes the instances of different scheduling problems and the related papers under each instance.

Table 1. Instances of the scheduling problem.

Problem Instance	Offline	Online
Independent moldable tasks	[10], [11], [18]	[6], [9], [15], [21]
Moldable task graphs	[5], [13], [16], [19]	[8], [This paper]

3 Problem Statement

In this section, we formally present the online scheduling model and the objective function. We also show a simple lower bound on the optimal makespan, against which the performance of our online algorithms will be measured.

3.1 Model and Objective

We consider the online scheduling of a directed acyclic graph (DAG) of moldable tasks on a platform with P identical processors. Let $G = (V, E)$ denote the task graph, where $V = \{1, 2, \dots, n\}$ represents a set of n tasks and $E \subseteq V \times V$ represents a set of precedence constraints among the tasks. An edge $(i, j) \in E$ indicates that task j depends on task i, and therefore it cannot be executed before task i is completed. Task i is called the *predecessor* of task j, and task j is called the *successor* of task i.

The tasks are assumed to be *moldable*, meaning that the number of processors allocated to a task can be determined by the scheduling algorithm at launch time, but once the task has started executing, its processor allocation cannot be changed. The execution time $t_j(p_j)$ of a task j is a function of the number p_j of processors allocated to it, and we assume that the processor allocation must be an integer between 1 and P. In this paper, we focus on the following execution time function:

$$t_j(p_j) = \frac{w_j}{\min(p_j, \bar{p}_j)} + d_j + c_j(p_j - 1) \ , \tag{1}$$

where w_j denotes the total parallelizable work of the task, $\bar{p}_j$ denotes the maximum degree of parallelism of the task, d_j denotes the sequential work of the task, and c_j denotes the communication overhead when more than one processor is used. The execution time function in Equation (1) generalizes several speedup models commonly observed for parallel applications. In particular, it contains the following well-known models as special cases:

- *Roofline Model [20] (with $d_j = 0$ and $c_j = 0$):*

  $$t_j(p_j) = \frac{w_j}{\min(p_j, \bar{p}_j)} \ . \tag{2}$$

  This model assumes that the task has a linear speedup until a maximum degree of parallelism $\bar{p}_j \le P$.
- *Communication Model [9] (with $\bar{p}_j \ge P$ and $d_j = 0$):*

  $$t_j(p_j) = \frac{w_j}{p_j} + c_j(p_j - 1) \ . \tag{3}$$

  This model assumes that the work of the task can be perfectly parallelized, but there is a communication overhead when more than one processor is allocated to the task, and that overhead increases linearly with the number of allocated processors.
- *Amdahl's Model [1] (with $\bar{p}_j \ge P$ and $c_j = 0$):*

  $$t_j(p_j) = \frac{w_j}{p_j} + d_j \ . \tag{4}$$

This model assumes that the task has a perfectly parallelizable fraction with work w_j and an inherently sequential fraction with work d_j.

From the execution time function of the task j, we can further define the *area* of the task as a function of the processor allocation as follows: $a_j(p_j) = p_j \times t_j(p_j)$. Intuitively, the area represents the total amount of processor resources utilized over the entire period of task execution.

In this work, we consider the *online scheduling* model, where a task becomes available only when all of its predecessors have been completed. This represents a common scheduling model for dynamic task graphs [4], [8]. Furthermore, when a task j is available, all of its execution time parameters (i.e., w_j, $\bar{p}_j$, d_j, c_j) also become known to the scheduling algorithm. The goal is to find a feasible schedule of the task graph that minimizes its overall completion time or *makespan*, denoted by T. The performance of an online scheduling algorithm is measured by its competitive ratio: the algorithm is said to be *c-competitive* if, for any task graph, its makespan T is at most c times the makespan T_{OPT} produced by an optimal offline scheduler, i.e., $T \le c \times T_{\text{OPT}}$. Note that the optimal offline scheduler may know all the tasks and their speedup models, as well as all dependencies in the graph, in advance. The competitive ratio is established against all possible strategies by an adversary trying to force the online algorithm to take *bad* decisions.

3.2 Lower Bound on Optimal Makespan

Given the execution time function in Equation (1), let us define $s_j = \sqrt{w_j/c_j}$. We can then compute the maximum number of processors that should be allocated to the task as

$$p_j^{\max} = \min\left(P, \bar{p}_j, \tilde{p}_j\right) \ , \qquad \text{where } \tilde{p}_j = \begin{cases} \lfloor s_j \rfloor, & \text{if } t_j(\lfloor s_j \rfloor) \le t_j(\lceil s_j \rceil) \\ \lceil s_j \rceil, & \text{otherwise} \end{cases} \tag{5}$$

Indeed, allocating more than $p_j^{\max}$ processors to the task will no longer decrease its execution time while only increasing its area. Thus, we can assume that the processor allocation of the task should never exceed $p_j^{\max}$ under any reasonable algorithm. In addition, we can observe that, when the processor allocation is in the range $[1, p_j^{\max}]$, the task satisfies the following *monotonic* property [16]:

- The execution time is a non-increasing function of the processor allocation, i.e., $t_j(p) \ge t_j(q)$ for all $1 \le p < q \le p_j^{\max}$;
- The area is a non-decreasing function of the processor allocation, i.e., $a_j(p) \le a_j(q)$ for all $1 \le p < q \le p_j^{\max}$.

Thus, the minimum execution time of the task is $t_j^{\min} = t_j(p_j^{\max})$ and the minimum area of the task is $a_j^{\min} = a_j(1)$. We note that the second point above also shows that the task cannot achieve superlinear speedup, i.e.,

$$\frac{t_j(p)}{t_j(q)} \le \frac{q}{p} \text{ for all } 1 \le p < q \le p_j^{\max} \ . \tag{6}$$

We now define two quantities that can be used as a lower bound of the optimal makespan.

Definition 1. *The* minimum total area $A_{\min}$ *of the task graph is the sum of the minimum area of all tasks in the graph, i.e.,* $A_{\min} = \sum_{j=1}^{n} a_j^{\min}$.

Definition 2. *The* minimum length $L_{\min}(f)$ *of a path*[1] f *in the graph is the sum of the minimum execution time of all tasks along that path, i.e.,* $L_{\min}(f) = \sum_{j \in f} t_j^{\min}$. *The* minimum critical path length $C_{\min}$ *of the graph is the longest minimum length of any path in the graph, i.e.,* $C_{\min} = \max_f L_{\min}(f)$.

Clearly, the optimal makespan cannot be smaller than $\frac{A_{\min}}{P}$ and $C_{\min}$. This follows from the well-known area bound and critical-path bound for scheduling any task graph. The choice of minimum value for both quantities ensures that they can serve as the lower bounds on the optimal makespan. The following lemma states this result.

Lemma 1. $T_{\text{OPT}} \ge \max\left(\frac{A_{\min}}{P}, C_{\min}\right)$.

4 Online Algorithm

In this section, we present an online scheduling algorithm and derive its competitive ratio for the considered speedup model (Equation (1)) as well as for its three special cases.

4.1 Algorithm Description

Algorithm 1 presents the pseudocode of the online scheduling algorithm, which at any time maintains the set of available tasks in a waiting queue Q. At time 0 or whenever a running task completes execution and thus releases processors, it checks if new tasks have become available. If so, for each newly available task j, it finds a processor allocation p_j for the task (using Algorithm 2) before inserting it into the queue Q. Then, it applies the well-known list scheduling strategy by scanning through all the available tasks in Q and executing each one right away if there are enough processors.

Algorithm 2 presents the details of the processor allocation strategy for any task j. It consists of two steps. The first step performs an initial allocation for the task, which is inspired by the Local Processor Allocation (LPA) strategy proposed in [2], [3]. Specifically, for each possible allocation $p \in [1, p_j^{\max}]$, we define the ratio between the area of the task and the minimum area to be $\alpha_p = a_j(p)/a_j^{\min}$, and the ratio between the execution time of the task and the minimum execution time to be $\beta_p = t_j(p)/t_j^{\min}$. We then find an allocation that minimizes α_p subject to the constraint $\beta_p \le \frac{1-2\mu}{\mu(1-\mu)}$, where $\mu \le \frac{3-\sqrt{5}}{2} \approx 0.382$ is a constant whose exact value will be determined based upon the speedup model under consideration. The justification for this strategy as well as for the choice of μ will be presented in the next section. Since α_p is non-decreasing with p and β_p is non-increasing with p, the above optimization problem can be efficiently solved in linear time.

In the second step, the algorithm reduces the initial allocation to $\lceil \mu P \rceil$ if it is more than $\lceil \mu P \rceil$; otherwise the allocation will be unchanged. Let p_j denote the initial allocation for the task and p'_j the final allocation. Thus, after the second step, we have:

$$p'_j = \begin{cases} \lceil \mu P \rceil, & \text{if } p_j > \lceil \mu P \rceil \\ p_j, & \text{otherwise} \end{cases} \ . \tag{7}$$

1. A path f consists of a sequence of tasks with linear dependency, i.e., $f = (j_{\pi(1)}, j_{\pi(2)}, \ldots, j_{\pi(v)})$, where the first task $j_{\pi(1)}$ in the sequence has no predecessor in the graph, the last task $j_{\pi(v)}$ has no successor, and, for each $2 \le i \le v$, task $j_{\pi(i)}$ is a successor of task $j_{\pi(i-1)}$.

This step adopts the technique first proposed in [16] and subsequently used in [12], [13]. The purpose is to be able to execute more tasks at any time during the schedule, thus potentially increasing the overall resource utilization of the platform and reducing the makespan.

Algorithm 1: Online_Scheduling_Algorithm

```
1  initialize a waiting queue $Q$
2  when at time 0 or a running task completes execution do
       // Processor Allocation
3      for each new task $j$ that becomes available do
4          Allocate_Processor($j$)
5          insert task $j$ into the waiting queue $Q$
6      end
       // List Scheduling
7      for each task $j$ in the waiting queue $Q$ do
8          if there are enough processors to execute the task then
9              execute task $j$ now
10         end
11     end
12 end
```

Algorithm 2: Allocate_Processor(j)

```
   // Step 1: Initial Allocation
1  Compute $p_j^{\max}$ based on Equation (5)
2  Compute $t_j^{\min} = t_j(p_j^{\max})$ and $a_j^{\min} = a_j(1)$
3  Find an allocation $p_j \in [1, p_j^{\max}]$ by solving the following
     optimization problem:
     $\min_p \ \alpha_p = \frac{\alpha_j(p)}{\alpha_j^{\min}}$ s.t. $\beta_p = \frac{t_j(p)}{t_j^{\min}} \le \frac{1-2\mu}{\mu(1-\mu)}$
   // Step 2: Allocation Adjustment
4  if $p_j > \lceil \mu P \rceil$ then
5      $p'_j \leftarrow \lceil \mu P \rceil$
6  else
7      $p'_j \leftarrow p_j$
8  end
```

4.2 General Analysis Framework

We now outline a general analysis framework, under which the competitive ratio of the proposed online algorithm will be derived for different speedup models.

Recall that T denotes the makespan of the online scheduling algorithm. Since the algorithm allocates and de-allocates processors upon task completions, the schedule can be divided into a set $\mathcal{I} = \{I_1, I_2, \dots\}$ of non-overlapping intervals, where tasks only start (or complete) at the beginning (or end) of an interval, and the number of utilized processors does not change during an interval. For each interval $I \in \mathcal{I}$, let $p(I)$ denote its processor utilization, i.e., the total number of processors used by all tasks running in interval I. Following the analysis of [16], we classify the set of intervals into the following categories.

- $\mathcal{I}_1$: subset of intervals whose processor utilization satisfies $p(I) \in (0, \lceil \mu P \rceil)$;
- $\mathcal{I}_2$: subset of intervals whose processor utilization satisfies $p(I) \in [\lceil \mu P \rceil, \lceil (1-\mu)P \rceil)$;
- $\mathcal{I}_3$: subset of intervals whose processor utilization satisfies $p(I) \in [\lceil (1-\mu)P \rceil, P]$.

Let $|I|$ denote the duration of an interval I, and let $T_1 = \sum_{I \in \mathcal{I}_1} |I|$, $T_2 = \sum_{I \in \mathcal{I}_2} |I|$ and $T_3 = \sum_{I \in \mathcal{I}_3} |I|$ be the total durations of the three categories of intervals, respectively. Since $\mathcal{I}_1$, $\mathcal{I}_2$ and $\mathcal{I}_3$ are obviously disjoint and partition $\mathcal{I}$, we have $T = T_1 + T_2 + T_3$.

The next two lemmas relate these durations to the minimum total area and minimum critical path length of the task graph, given certain conditions on the initial processor allocations of the tasks.

Lemma 2. *If there exists a constant α such that, for each task j, its initial processor allocation satisfies $a_j(p_j) \le \alpha \times a_j^{\min}$, then we have:*

$$\mu T_2 + (1-\mu) T_3 \le \alpha \times \frac{A_{\min}}{P} \ . \tag{8}$$

Proof. As the area of each task j is non-decreasing with its processor allocation and $p'_j \le p_j$, the final area of the task should satisfy $a_j(p'_j) \le a_j(p_j) \le \alpha \times a_j^{\min}$. Thus, the total area A' of all tasks after their final allocations will satisfy $A' = \sum_j a_j(p'_j) \le \alpha \times \sum_j a_j^{\min} = \alpha \times A_{\min}$.

Since at least $\lceil \mu P \rceil \ge \mu P$ processors are utilized during T_2 and at least $\lceil (1-\mu)P \rceil \ge (1-\mu)P$ processors are utilized during T_3, we have $\mu T_2 + (1-\mu)T_3 \le \frac{A'}{P} \le \alpha \times \frac{A_{\min}}{P}$. □

Lemma 3. *If there exists a constant β such that, for each task j, its initial processor allocation satisfies $t_j(p_j) \le \beta \times t_j^{\min}$ and $\beta \le \frac{1}{\mu}$, then we have:*

$$\frac{T_1}{\beta} + \mu T_2 \le C_{\min} \ . \tag{9}$$

Proof. During T_1 and T_2, the processor utilization is at most $\lceil (1-\mu)P \rceil - 1$, so there are at least $P - (\lceil (1-\mu)P \rceil - 1) \ge \lceil \mu P \rceil$ available processors. Based on Algorithm 2, any task is allocated at most $\lceil \mu P \rceil$ processors. Thus, there are enough processors to execute any new task (if one is available). This implies that there is no available task in the queue $\mathcal{Q}$ during T_1 and T_2. When a task graph is scheduled by the list scheduling algorithm, it is well known that there exists a path f in the graph such that some task along that path will be running whenever there is no available task in the queue [8], [13], [16].

For any task j along path f running during T_1, its processor allocation must be less than $\lceil \mu P \rceil$, hence is not reduced by Step 2 of Algorithm 2, i.e., $p'_j = p_j$. Thus, its execution time should satisfy $t_j(p'_j) = t_j(p_j) \le \beta \times t_j^{\min}$.

For any task j along path f running during T_2, its processor allocation may or may not be reduced. If it is not reduced, then similarly we can get $t_j(p'_j) \le \beta \times t_j^{\min} \le \frac{1}{\mu} \times t_j^{\min}$. Otherwise, if it is reduced, and based on Equation (6), the task execution time should satisfy:

$$\frac{t_j(p'_j)}{t_j^{\min}} = \frac{t_j(\lceil \mu P \rceil)}{t_j(p_j^{\max})} \le \frac{p_j^{\max}}{\lceil \mu P \rceil} \le \frac{P}{\mu P} = \frac{1}{\mu} \ .$$

Now, let $L'_{\min}(f)$ (resp. $L''_{\min}(f)$) denote the minimum length for the portion of path f executed during T_1 (resp. T_2). The argument above implies that $T_1 \le \beta \times L'_{\min}(f)$ and $T_2 \le \frac{1}{\mu} \times L''_{\min}(f)$. Thus, we have $\frac{T_1}{\beta} + \mu T_2 \le L'_{\min}(f) + L''_{\min}(f) \le L_{\min}(f) \le C_{\min}$. □

Based on the results of Lemmas 2 and 3, we can now derive a bound on the makespan of the online scheduling algorithm as shown below.

Lemma 4. *If there exist two constants α and β such that, for each task j, its initial processor allocation satisfies $a_j(p_j) \leq \alpha \times a_j^{\min}$ and $t_j(p_j) \leq \beta \times t_j^{\min}$ with $\beta \leq \frac{1-2\mu}{\mu(1-\mu)}$, then we have:*

$$T \leq \frac{\mu\alpha + 1 - 2\mu}{\mu(1-\mu)} \times T_{\text{OPT}} . \qquad (10)$$

Proof. As the makespan is given by $T = T_1 + T_2 + T_3$, we can multiply both sides by $\frac{1-\mu}{\alpha}$ and apply Equation (8) to remove the T_3 term, which gives:

$$\frac{1-\mu}{\alpha} T \leq \frac{1-\mu}{\alpha} T_1 + \frac{1-2\mu}{\alpha} T_2 + T_{\text{OPT}} .$$

We can then multiply both sides of the above inequality by $\frac{\mu\alpha}{1-2\mu}$ and use Equation (9) to remove the T_2 term (since $\beta \leq \frac{1-2\mu}{\mu(1-\mu)} = \frac{1}{\mu} - \frac{1}{1-\mu} \leq \frac{1}{\mu}$). This gives:

$$\frac{\mu(1-\mu)}{1-2\mu} T \leq \left(\frac{\mu(1-\mu)}{1-2\mu} - \frac{1}{\beta}\right) T_1 + \left(\frac{\mu\alpha}{1-2\mu} + 1\right) T_{\text{OPT}} .$$

Finally, if $\beta \leq \frac{1-2\mu}{\mu(1-\mu)}$, the first term above becomes non-positive and hence can be removed without affecting the inequality. By rearranging the factors, we can then obtain the result as shown in Equation (10). □

The result of Lemma 4 shows that the competitive ratio of the online algorithm increases with α, for a given μ. This suggests that the initial processor allocation should try to minimize α subject to the constraint $\beta \leq \frac{1-2\mu}{\mu(1-\mu)}$, which is what is done in Step 1 of Algorithm 2. Since $\beta \geq 1$, the value of μ needs to satisfy $\frac{1-2\mu}{\mu(1-\mu)} \geq 1$, and solving it gives $\mu \leq \frac{3-\sqrt{5}}{2} \approx 0.382$.

4.3 Competitive Ratio

In this section, we prove the competitive ratio of the online algorithm, which is given by $\frac{\mu\alpha+1-2\mu}{\mu(1-\mu)}$ subject to $\beta \leq \frac{1-2\mu}{\mu(1-\mu)}$, based on Lemma 4. We will show that there exists a processor allocation parameterized by a parameter x and that achieves specific values of α and β for any task that follows the considered speedup model. Then, by carefully choosing the values of x and μ, we can minimize the ratio while satisfying the constraint.

In the following, we first consider the three special speedup models (i.e., roofline, communication and Amdahl) before tackling the general model. As the analysis focuses on bounding the ratios α and β for each individual task, we drop the task index j for simplicity.

4.3.1 Roofline Model

Recall that a task follows the roofline speedup model if its execution time satisfies $t(p) = \frac{w}{\min(p,\bar{p})}$ for some $\bar{p} \leq P$.

Lemma 5. *For any task that follows the roofline speedup model, there exists a processor allocation that achieves $\alpha = 1$ and $\beta = 1$.*

Proof. Setting the processor allocation to $\bar{p}$ achieves both the minimum execution time and the minimum area for the task, thus giving $\alpha = \beta = 1$. □

Theorem 1. *The online algorithm is 2.62-competitive for any graph of tasks that follow the roofline speedup model. This is achieved with $\mu = \frac{3-\sqrt{5}}{2} \approx 0.382$.*

Proof. With $\beta = 1$, the condition $\frac{1-2\mu}{\mu(1-\mu)} \geq \beta = 1$ can be satisfied with $\mu \leq \frac{3-\sqrt{5}}{2}$. Since $\alpha = 1$, the competitive ratio is given by $\frac{\mu+1-2\mu}{\mu(1-\mu)} = \frac{1}{\mu}$. By setting $\mu = \frac{3-\sqrt{5}}{2} \approx 0.382$, the ratio is minimized at $\frac{1}{\mu} = \frac{3+\sqrt{5}}{2} < 2.62$. □

The above ratio retains the same result by Feldmann et al. [8][2]. They also proved a matching lower bound for any online deterministic algorithm under the "non-clairvoyant" setting, where the work w of a task is also unknown to the scheduler.

4.3.2 Communication Model

Recall that a task follows the communication model if its execution time satisfies $t(p) = \frac{w}{p} + c(p-1)$. For the ease of analysis, we rewrite the execution time function as: $t(p) = c(\frac{w'}{p} + p - 1)$ with $w' = \frac{w}{c}$.

Lemma 6. *For any task that follows the communication model and for any $x \in [\frac{\sqrt{13}-1}{6}, \frac{1}{2}]$, there exists a processor allocation that achieves $\alpha_x = 1 + x^2 + \frac{x}{3}$ and $\beta_x = \frac{3}{5x} + \frac{3x}{5}$.*

Proof. Recall that $p^{\max}$ denotes the number of processors that minimizes the execution time function $t(p)$, i.e., $t(p^{\max}) = t^{\min}$. Clearly, we have either $p^{\max} = P$ or $\lfloor\sqrt{w'}\rfloor \leq p^{\max} \leq \lceil\sqrt{w'}\rceil$. Also, the area function is given by $a(p) = p \times t(p) = c(w' + p(p-1))$, and the minimum area is obtained with one processor, i.e., $a^{\min} = a(1) = cw'$. We consider two cases.

Case 1: $w' \leq 9$. In this case, we must have $p^{\max} \leq 3$. We further divide this case into three subcases and, for each subcase, we will show that there always exists a processor allocation p that achieves $\alpha \leq \frac{4}{3}$ and $\beta \leq \frac{3}{2}$.

- If $p^{\max} = 1$, we can set $p = 1$ and get $\alpha = \beta = 1$.
- If $p^{\max} = 2$, we can set $p = 1$ and get $\alpha = 1$. Moreover, $t(2) \leq t(3) \Rightarrow \frac{w'}{2} + 1 \leq \frac{w'}{3} + 2 \Rightarrow w' \leq 6$. We then have $\beta = \frac{t(1)}{t^{\min}} = \frac{w'}{\frac{w'}{2}+1} \leq \frac{w'}{\frac{w'}{2}+\frac{w'}{6}} = \frac{3}{2}$.
- If $p^{\max} = 3$, we can set $p = 2$. In this case, $t(2) \geq t(3) \Rightarrow w' \geq 6$, and we also supposed $w' \leq 9$. Thus, $\alpha = \frac{a(2)}{a^{\min}} = \frac{w'+2}{w'}$, which is decreasing with w', and plugging in $w' \geq 6$, we get $\alpha \leq \frac{4}{3}$. Furthermore, $\beta = \frac{t(2)}{t^{\min}} = \frac{\frac{w'}{2}+1}{\frac{w'}{3}+2} = \frac{3w'+6}{2w'+12}$, which is increasing with w', and plugging in $w' \leq 9$, we get $\beta \leq \frac{11}{10}$.

Case 2: $w' > 9$. In this case, for any $x \in [\frac{\sqrt{13}-1}{6}, \frac{1}{2}]$, we can set $p = \min(\lceil x\sqrt{w'}\rceil, P)$. First, if we allow the processor allocation to take non-integer values, the execution time function $t(p)$ would be minimized at $p^* = \sqrt{w'}$. Thus, the minimum execution time should satisfy $t^{\min} \geq t(p^*) = c(2\sqrt{w'} - 1)$. We further consider two subcases.

- If $p = \lceil x\sqrt{w'}\rceil$, we apply $x\sqrt{w'} \leq p \leq x\sqrt{w'} + 1$ to get $\alpha = \frac{a(p)}{a^{\min}} = \frac{w'+p(p-1)}{w'} \leq 1+x^2+\frac{x}{\sqrt{w'}} \leq 1+x^2+\frac{x}{3} = \alpha_x$, and $\beta = \frac{t(p)}{t^{\min}} \leq \frac{\frac{\sqrt{w'}}{x}+x\sqrt{w'}}{2\sqrt{w'}-1} = \frac{1/x+x}{2-1/\sqrt{w'}} \leq \frac{1/x+x}{2-1/3} = \frac{3}{5}(\frac{1}{x} + x) = \beta_x$.
- If $p = P < \lceil x\sqrt{w'}\rceil$, then as $x \leq \frac{1}{2}$, we must have $\sqrt{w'} > P$ and thus $\tilde{p} = p = P$. In this case, we clearly

2. In [8], each task has a parallelism p, and can be virtualized if $p' \leq p$ processors are used for execution, with a linear slowdown. This is equivalent to the roofline model.

have $\beta = 1 \leq \beta_x$. Moreover, we still have $p \leq x\sqrt{w'}+1$, thus $\alpha = \frac{a(p)}{a^{\min}} \leq 1 + x^2 + \frac{x}{3} = \alpha_x$ holds.

Lastly, we need to make sure that $\alpha_x \geq \frac{4}{3}$ and $\beta_x \geq \frac{3}{2}$, because the ratios must hold for Case 1 as well. We can easily check that $x \leq \frac{1}{2} \Rightarrow \beta_x \geq \frac{3}{2}$ and $x \geq \frac{\sqrt{13}-1}{6} \Rightarrow \alpha_x \geq \frac{4}{3}$, thus the result holds for any $x \in [\frac{\sqrt{13}-1}{6}, \frac{1}{2}]$. □

Theorem 2. *The online algorithm is 3.61-competitive for any graph of tasks that follow the communication model. This is achieved with* $\mu \approx 0.324$.

Proof. From result of Lemma 6, we aim to minimize $\alpha_x = 1 + x^2 + \frac{x}{3}$ while satisfying the constraint $\beta_x = \frac{3}{5x} + \frac{3x}{5} \leq \frac{1-2\mu}{\mu(1-\mu)}$. For a fixed μ, multiplying both sides of the constraint by x and rearranging terms, we get a second-degree inequality: $\frac{3}{5}x^2 - \frac{1-2\mu}{\mu(1-\mu)}x + \frac{3}{5} \leq 0$. The smallest x satisfying this inequality can be computed to be $x^*_\mu = \frac{5}{6}\left(\frac{1-2\mu}{\mu(1-\mu)} - \sqrt{\left(\frac{1-2\mu}{\mu(1-\mu)}\right)^2 - \frac{36}{25}}\right)$.

Now, plugging the above expression of x^*_μ into $\alpha_x = 1 + x^2 + \frac{x}{3}$ and plugging the result into the competitive ratio $\frac{\mu\alpha_x+1-2\mu}{\mu(1-\mu)}$, we get a function with only a single variable μ. Minimizing this function numerically for $\mu \in (0, \frac{3-\sqrt{5}}{2}]$, we can get the optimal competitive ratio to be at most 3.61, which is obtained at $\mu^* \approx 0.324$. This results in the value $x^*_\mu \approx 0.446$, which is indeed in $[\frac{\sqrt{13}-1}{6}, \frac{1}{2}]$, thus is a valid choice. □

4.3.3 Amdahl's Model

Recall that a task follows the Amdahl's model if its execution time function is $t(p) = \frac{w}{p} + d$, so the area function is given by $a(p) = p \times t(p) = w + dp$.

Lemma 7. *For any task that follows the Amdahl's model and for any* $x > 0$, *there exists a processor allocation that achieves* $\alpha_x = 1 + x$ *and* $\beta_x = 1 + \frac{1}{x}$.

Proof. The minimum execution time of the task is obtained by allocating all P processors, i.e., $t^{\min} = t(P) = \frac{w}{P} + d$, and the minimum area is obtained with one processor, i.e., $a^{\min} = a(1) = w + d$.

For any $x > 0$, we can set $p = \min(\lceil x\frac{w}{d}\rceil, P)$. This implies $p \leq \lceil x\frac{w}{d}\rceil \leq x\frac{w}{d} + 1$. Thus, we have $\alpha = \frac{a(p)}{a^{\min}} = \frac{w+dp}{w+d} \leq \frac{w+d\left(x\frac{w}{d}+1\right)}{w+d} = \frac{w+d+xw}{w+d} = 1 + \frac{xw}{w+d} \leq 1 + x = \alpha_x$. Furthermore, if $p = \lceil x\frac{w}{d}\rceil \geq x\frac{w}{d}$, we have $\beta = \frac{t(p)}{t^{\min}} \leq \frac{\frac{w}{x\frac{w}{d}}+d}{\frac{w}{P}+d} \leq \frac{\frac{d}{x}+d}{d} = 1 + \frac{1}{x} = \beta_x$. Otherwise, if $p = P$, we get $t(p) = t^{\min}$ and thus $\beta = 1 < \beta_x$. □

Theorem 3. *The online algorithm is 4.74-competitive for any graph of tasks that follow the Amdahl's model. This is achieved with* $\mu \approx 0.271$.

Proof. Again, we need to minimize $\alpha_x = 1 + x$ subject to the constraint $\beta_x = 1 + \frac{1}{x} \leq \frac{1-2\mu}{\mu(1-\mu)}$. For a fixed μ, the smallest x satisfying the above inequality can be computed as: $x^*_\mu = \frac{\mu(1-\mu)}{\mu^2-3\mu+1}$.

Plugging x^*_μ into $\alpha_x = 1 + x$, and then plugging the result into the competitive ratio $\frac{\mu\alpha_x+1-2\mu}{\mu(1-\mu)}$ and simplifying, we can get the following function:

$$f(\mu) = \frac{-2\mu^3 + 5\mu^2 - 4\mu + 1}{-\mu^4 + 4\mu^3 - 4\mu^2 + \mu} .$$

Minimizing this function numerically for $\mu \in (0, \frac{3-\sqrt{5}}{2}]$, we can get the optimal competitive ratio to be at most 4.74, which is obtained at $\mu^* \approx 0.271$ (thus $x^*_\mu \approx 0.759$). □

4.3.4 General Model

We finally consider the general speedup model as given in Equation (1). Again, for the ease of analysis, we rewrite the execution time function as: $t(p) = c(\frac{w'}{\min(p,\bar{p})} + d' + p - 1)$ with $w' = \frac{w}{c}$ and $d' = \frac{d}{c}$.

Lemma 8. *For any task that follows the general model and for any* $x > 1$, *there exists a processor allocation that achieves* $\alpha_x = 1 + \frac{1}{x} + \frac{1}{x^2}$ *and* $\beta_x = x + 1 + \frac{1}{x}$.

Proof. If we allow the processor allocation to take non-integer values and assuming unbounded $\bar{p}$, the execution time function $t(p)$ would be minimized at $p^* = \sqrt{w'}$. Thus, the minimum execution time should satisfy $t^{\min} \geq c(2\sqrt{w'} + d' - 1)$. Note that this bound will hold true regardless of the value of $\bar{p}$: it is obviously true if $\bar{p} \geq p^*$, otherwise $t^{\min}$ is achieved at $\bar{p}$, with a value also higher than $c(2\sqrt{w'} + d' - 1)$. Furthermore, the minimum area is obtained with one processor, i.e., $a^{\min} = a(1) = c(w' + d')$.

Recall that $p^{\max}$ denotes the number of processors that minimizes the execution time, i.e., $t(p^{\max}) = t^{\min}$. Clearly, we have either $p^{\max} = P$, or $\lfloor\sqrt{w'}\rfloor \leq p^{\max} \leq \lceil\sqrt{w'}\rceil$, or $p^{\max} = \bar{p}$. We consider two cases.

Case 1: $w' \leq 1$. In this case, it must be that $p^{\max} = 1$. We can then set the processor allocation to be $p = 1$ and have $\alpha = \beta = 1$.

Case 2: $w' > 1$. In this case, for any $x > 1$, we can set $p = \min(\lceil\frac{w'+d'}{x(\sqrt{w'}+d')}\rceil, \bar{p}, P)$, thus have $a(p) = c(w' + p(d' + p - 1))$. Since $p \leq \lceil\frac{w'+d'}{x(\sqrt{w'}+d')}\rceil \leq \frac{w'+d'}{x(\sqrt{w'}+d')} + 1$, we obtain:

$$\begin{aligned}
\alpha = \frac{a(p)}{a^{\min}} &= \frac{w' + p(d' + p - 1)}{w' + d'} \\
&\leq \frac{w' + \left(\frac{w'+d'}{x(\sqrt{w'}+d')} + 1\right)\left(d' + \frac{w'+d'}{x(\sqrt{w'}+d')}\right)}{w' + d'} \\
&= \frac{w' + d'\frac{w'+d'}{x(\sqrt{w'}+d')} + \left(\frac{w'+d'}{x(\sqrt{w'}+d')}\right)^2 + d' + \frac{w'+d'}{x(\sqrt{w'}+d')}}{w' + d'} \\
&= 1 + \frac{d' + 1}{x(\sqrt{w'} + d')} + \frac{w' + d'}{x^2(\sqrt{w'} + d')^2} \\
&\leq 1 + \frac{1}{x} + \frac{1}{x^2} = \alpha_x .
\end{aligned}$$

The last inequality above comes from $w' > 1$ and $d' > 0$.

Since $w' > 1$, we get $t^{\min} > c(\sqrt{w'} + d')$. To derive β, we further consider two subcases.

- If $p = \lceil\frac{w'+d'}{x(\sqrt{w'}+d')}\rceil$, then $p \times t^{\min} > c\frac{w'+d'}{x} = \frac{a^{\min}}{x}$. We can then get $\beta = \frac{t(p)}{t^{\min}} < x\frac{t(p)p}{a^{\min}} = x\frac{a(p)}{a^{\min}} \leq x\alpha_x = x + 1 + \frac{1}{x} = \beta_x$.
- If $p < \lceil\frac{w'+d'}{x(\sqrt{w'}+d')}\rceil$, then we must have $p = \min(\bar{p}, P) < \lceil\frac{w'+d'}{x(\sqrt{w'}+d')}\rceil$. Since p is an integer, it is necessarily the case that $p < \frac{w'+d'}{x(\sqrt{w'}+d')} \leq \frac{w'+d'}{\sqrt{w'}+d'} \leq \sqrt{w'}$ (because $w' > 1$). Therefore, we should also have $p^{\max} = \min(\bar{p}, P) = p$, and thus $\beta = 1$. □

Theorem 4. *The online algorithm is 5.72-competitive for any graph of tasks that follow the general speedup model given in Equation* (1). *This is achieved with $\mu \approx 0.211$.*

Proof. Once again, we aim at minimizing $\alpha_x = 1 + \frac{1}{x} + \frac{1}{x^2}$ subject to $\beta_x = x + 1 + \frac{1}{x} \leq \frac{1-2\mu}{\mu(1-\mu)}$. For a fixed μ, the constraint above corresponds to a second-degree inequality: $x^2 - \frac{\mu^2-3\mu+1}{\mu(1-\mu)}x + 1 \leq 0$. The largest x satisfying this inequality can be computed as $x^*_\mu = \frac{1}{2}\left(\frac{\mu^2-3\mu+1}{\mu(1-\mu)} + \sqrt{\left(\frac{\mu^2-3\mu+1}{\mu(1-\mu)}\right)^2 - 4}\right)$.

Plugging the above x^*_μ into $\alpha_x = 1 + \frac{1}{x} + \frac{1}{x^2}$ and then plugging the result into the competitive ratio $\frac{\mu\alpha_x+1-2\mu}{\mu(1-\mu)}$, we get a function with only a single variable μ. Minimizing this function numerically for $\mu \in (0, \frac{3-\sqrt{5}}{2}]$, we obtain that the optimal competitive ratio is at most 5.72, obtained at $\mu^* \approx 0.211$. This results in the value $x^*_\mu \approx 1.972$. □

5 A Lower Bound for Arbitrary Speedup Model

In the previous section, we have proven constant competitive ratios of our online algorithm for task graphs under several common speedup models. In this section, we show that the competitive ratio of any deterministic online algorithm can be unbounded for the arbitrary speedup model.

Theorem 5. *Any deterministic online algorithm is at least $\Omega(\ln(D))$-competitive for scheduling moldable task graphs under the arbitrary speedup model, where D denotes the number of tasks along the longest (critical) path of the graph.*

Proof. We fix an arbitrary integer $\ell > 1$ and set $K = 2^\ell$. The instance consists of $n = 2^K - 1$ independent linear task chains organized in groups. Specifically, for any $i \in [1, K]$, group i contains 2^{K-i} linear chains, each with exactly i tasks. Thus, the number of tasks along the longest path of the graph is given by $D = K$. Figure 1 shows such an instance for $\ell = 2, K = 4$ and $n = 15$. All tasks in the graph are identical, with an execution time function $t(p) = \frac{1}{\lg(p)+1}$. We set the total number of processors to be $P = K \times 2^{K-1}$.

We show that the optimal offline algorithm completes the above instance with a makespan at most 1, whereas any deterministic online algorithm may produce a makespan at least $\ln(K) - \ln(\ell) - \frac{1}{\ell}$, thus showing the result.

First, the optimal offline algorithm could schedule the tasks as follows: for any group $i \in [1, K]$, it allocates 2^{i-1} processors to each linear chain in the group. The total number of required processors is then $\sum_{i=1}^{K} 2^{i-1} \times 2^{K-i} = K \times 2^{K-1} = P$. Thus, all linear chains could be executed in parallel. Furthermore, they will all be completed at time 1, since each linear chain in group i has i tasks, and each task has an execution time $t(2^{i-1}) = \frac{1}{\lg(2^{i-1})+1} = \frac{1}{i}$. Figure 2 illustrates the schedule for our instance with $\ell = 2$.

Now, we establish a lower bound on the makespan of any deterministic online algorithm. For any $i \in [1, K-1]$, let L_i denote the set of linear chains in all groups $j \leq i$, and let L'_i denote the set of linear chains in all groups $j > i$. Let us define t_i to be the first time a linear chain in L'_i completes i tasks. We further define $t_0 = 0$ and let t_K denote the makespan of the online algorithm.

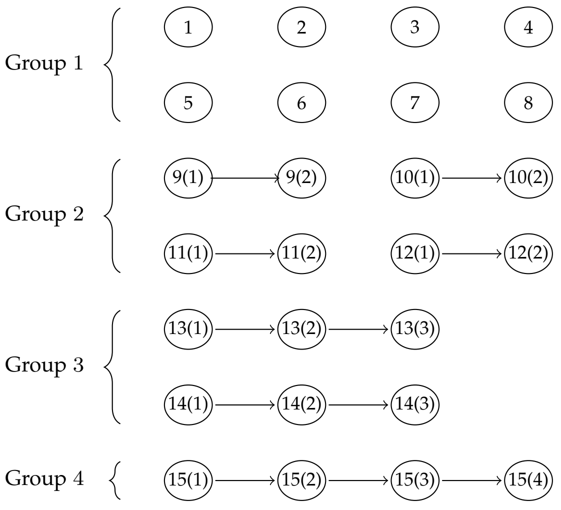

Fig. 1. Lower bound instance for $\ell = 2$, $K = 4$, and $n = 15$ linear task chains. Each circle represents a task and the number inside each circle indicates the ID of the linear chain the task is in (and the number in the parenthesis indicates the task's position in that linear chain).

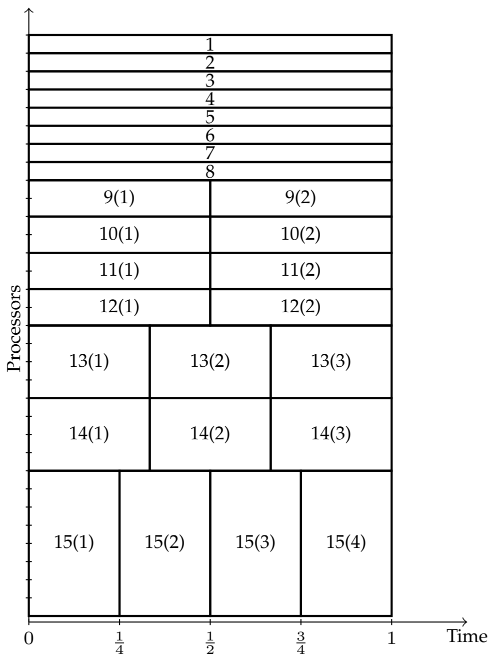

Fig. 2. An offline schedule for the lower bound instance with $\ell = 2$, $K = 4$, and $n = 15$ linear chains, producing a makespan of 1.

Lemma 9. *In the worst case, a schedule produced by any deterministic online algorithm could satisfy:*

$$t_i - t_{i-1} \geq \frac{1}{\ell + i}, \ \forall i \in [1, K].$$

Proof. Since all tasks are identical, an online algorithm cannot distinguish the linear chains. Thus, for any $i \in [1, K]$, an adversary could make all linear chains that first complete i tasks by the online algorithm be chains from L_i. Therefore, at time t_i, all linear chains containing exactly i tasks (i.e., the ones from group i) are already completed, and at time t_{i-1}, no linear chain has started its i-th task by definition (this also holds for t_0 and t_K). Hence, all tasks in the i-th position of the linear chains in group i must be entirely processed between t_i and t_{i-1}, and the number of such tasks is 2^{K-i}.

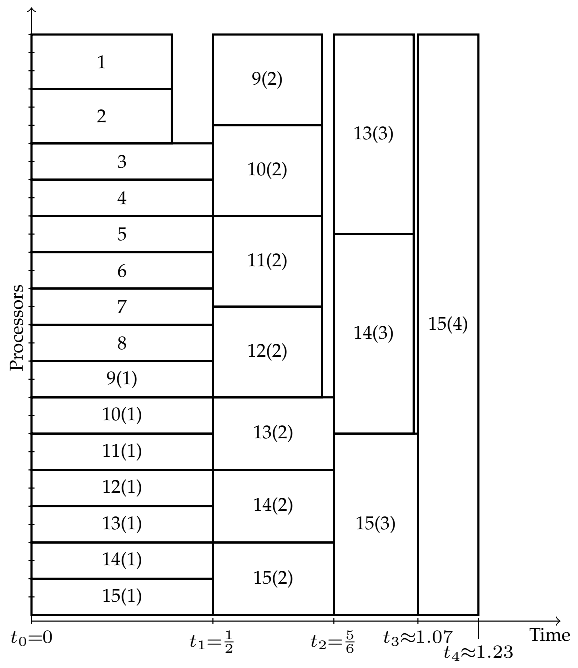

Fig. 3. The schedule of an online algorithm for the lower bound instance with $\ell = 2$, $K = 4$, and $n = 15$ linear chains. The algorithm allocates (approximately) the same number of processors to all linear chains, producing a makespan $t_4 \approx 1.23$.

For the sake of contradiction, suppose we have $t_i - t_{i-1} < \frac{1}{\ell+i}$. Thus, the execution time of these tasks must satisfy $t(p) = \frac{1}{\lg(p)+1} \leq \frac{1}{\ell+i}$, hence their processor allocation must be at least $p \geq 2^{\ell+i-1} = K \times 2^{i-1}$. As the area of the task $a(p) = p \times t(p) = \frac{p}{\lg(p)+1}$ is increasing with the number of processors, the total area of all tasks that needs to be processed between t_i and t_{i-1} is at least $2^{K-i} \times a(K \times 2^{i-1}) = \frac{2^{K-i} \times K \times 2^{i-1}}{\log(K \times 2^{i-1})+1} = \frac{K \times 2^{K-1}}{\ell+i} = \frac{P}{\ell+i}$. Since we have P processors, the total time required to process this area is at least $\frac{1}{\ell+i}$ which contradicts $t_i - t_{i-1} < \frac{1}{\ell+i}$. □

One strategy to cope with the worst-case scenario above is to allocate the same number of processors to each linear chain (or more precisely allocate one more processor to some linear chains in order to utilize all the processors). Figure 3 illustrates this strategy for the same instance with $\ell = 2$.

Finally, we can use the result of Lemma 9 to lower bound the makespan of an online algorithm, which is given by $t_K = \sum_{i=1}^{K}(t_i - t_{i-1})$. Since $\forall j, \ln(j) + \gamma < \sum_{i=1}^{j} \frac{1}{i} < \ln(j) + \gamma + \frac{1}{j}$ where γ is the Euler constant, we obtain:

$$
\begin{aligned}
t_K &\geq \sum_{i=1}^{K} \frac{1}{\ell+i} > \sum_{i=\ell+1}^{K} \frac{1}{i} > \sum_{i=1}^{K} \frac{1}{i} - \sum_{i=1}^{\ell} \frac{1}{i} \\
&> (\ln(K) + \gamma) - \left(\ln(\ell) + \gamma + \frac{1}{\ell}\right) = \ln(K) - \ln(\ell) - \frac{1}{\ell} .
\end{aligned}
$$

This completes the proof of Theorem 5. □

6 Conclusion and Future Work

This paper studies the online scheduling of moldable task graphs whose tasks obey different speedup models. To the best of our knowledge, no competitive ratio was known under this setting, except for the roofline model [8]. We have extended the result and derived competitive ratios for several other speedup models, including the communication model, the Amdahl's model and a general combination. We have also established a lower bound for the arbitrary speedup model. Altogether, these new results lay the foundations for further study of this important but difficult scheduling problem. Future work will aim at assessing the tightness of the competitive ratios obtained in this work.

References

[1] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *AFIPS'67*, pages 483–485, 1967.

[2] A. Benoit, V. Le Fèvre, L. Perotin, P. Raghavan, Y. Robert, and H. Sun. Resilient scheduling of moldable jobs on failure-prone platforms. In *IEEE Cluster*, 2020.

[3] A. Benoit, V. Le Fèvre, L. Perotin, P. Raghavan, Y. Robert, and H. Sun. Resilient scheduling of moldable parallel jobs to cope with silent errors. *IEEE Transactions on Computers*, 2021.

[4] L. Canon, L. Marchal, B. Simon, and F. Vivien. Online scheduling of task graphs on heterogeneous platforms. *IEEE Trans. Parallel Distributed Syst.*, 31(3):721–732, 2020.

[5] C.-Y. Chen and C.-P. Chu. A 3.42-approximation algorithm for scheduling malleable tasks under precedence constraints. *IEEE Trans. Parallel Distrib. Syst.*, 24(8):1479–1488, 2013.

[6] R. A. Dutton and W. Mao. Online scheduling of malleable parallel jobs. In *PDCS*, pages 136–141, 2007.

[7] D. G. Feitelson and L. Rudolph. Toward convergence in job schedulers for parallel supercomputers. In *Job Scheduling Strategies for Parallel Processing*, pages 1–26. Springer, 1996.

[8] A. Feldmann, M.-Y. Kao, J. Sgall, and S.-H. Teng. Optimal online scheduling of parallel jobs with dependencies. *Journal of Combinatorial Optimization*, 1(4):393–411, 1998.

[9] J. T. Havill and W. Mao. Competitive online scheduling of perfectly malleable jobs with setup times. *European Journal of Operational Research*, 187:1126–1142, 2008.

[10] K. Jansen. A $(3/2 + \epsilon)$ approximation algorithm for scheduling moldable and non-moldable parallel tasks. In *SPAA*, pages 224–235, 2012.

[11] K. Jansen and F. Land. Scheduling monotone moldable jobs in linear time. In *IPDPS*, pages 172–181, 2018.

[12] K. Jansen and H. Zhang. Scheduling malleable tasks with precedence constraints. In *SPAA*, page 86–95, 2005.

[13] K. Jansen and H. Zhang. An approximation algorithm for scheduling malleable tasks under general precedence constraints. *ACM Trans. Algorithms*, 2(3):416–434, 2006.

[14] B. Johannes. Scheduling parallel jobs to minimize the makespan. *J. of Scheduling*, 9(5):433–452, 2006.

[15] N. Kell and J. Havill. Improved upper bounds for online malleable job scheduling. *J. of Scheduling*, 18(4):393–410, 2015.

[16] R. Lepère, D. Trystram, and G. J. Woeginger. Approximation algorithms for scheduling malleable tasks under precedence constraints. In *ESA*, pages 146–157, 2001.

[17] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.

[18] J. Turek, J. L. Wolf, and P. S. Yu. Approximate algorithms scheduling parallelizable tasks. In *SPAA*, 1992.

[19] Q. Wang and K. H. Cheng. A heuristic of scheduling parallel tasks and its analysis. *SIAM J. Comput.*, 21(2):281–294, 1992.

[20] S. Williams, A. Waterman, and D. Patterson. Roofline: An insightful visual performance model for multicore architectures. *Commun. ACM*, 52(4):65–76, 2009.

[21] D. Ye, D. Z. Chen, and G. Zhang. Online scheduling of moldable parallel tasks. *J. of Scheduling*, 21(6):647–654, 2018.