Online Scheduling of Moldable Task Graphs under Common Speedup Models

Anne Benoit
LIP, ENS Lyon
Lyon, France
anne.benoit@ens-lyon.fr

Lucas Perotin
LIP, ENS Lyon
Lyon, France
lucas.perotin@ens-lyon.fr

Yves Robert
LIP, ENS Lyon, France
& Univ. Tenn. Knoxville, USA
yves.robert@inria.fr

Hongyang Sun
University of Kansas
Lawrence, KS, USA
hongyang.sun@ku.edu

ABSTRACT

The problem of scheduling moldable tasks on multiprocessor systems with the objective of minimizing the overall completion time (or makespan) 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 studies have focused on both (i.e., online scheduling of moldable task graphs). In this paper, we design a new online algorithm and derive constant competitive ratios for this problem under several common yet realistic speedup models (i.e., roofline, communication, Amdahl, and a general combination). We also prove, for each model, a lower bound on the competitiveness of our algorithm, which is very close to the constant competitive ratio. Finally, we provide the first lower bound on the competitive ratio of any deterministic online algorithm for the arbitrary speedup model, which is not constant but depends on the number of tasks in the longest path of the graph.

KEYWORDS

Task graph, moldable task, online scheduling, competitive ratio.

ACM Reference Format:
Anne Benoit, Lucas Perotin, Yves Robert, and Hongyang Sun. 2022. Online Scheduling of Moldable Task Graphs under Common Speedup Models. In *51st International Conference on Parallel Processing (ICPP '22), August 29-September 1, 2022, Bordeaux, France.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3545008.3545049

1 INTRODUCTION

This work investigates the online scheduling of parallel task graphs on a set of identical processors, 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 used to execute it, which is chosen once and for all when the task starts its execution but 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) [8].

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 [9], 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 comes in many flavors:

- **Offline Scheduling vs. Online Scheduling.** In the offline version of the scheduling problem, all tasks are known in advance, before the execution starts. The problem is $\mathcal{NP}$-complete, and the goal is to design good 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 [19] 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,

ICPP '22, August 29-September 1, 2022, Bordeaux, France

ACM ISBN 978-1-4503-9733-9/22/08…$15.00
https://doi.org/10.1145/3545008.3545049

Table 1: Competitive ratios for our online algorithm.

Model	Roofline	Comm.	Amdahl	General
Upper bound	2.62	3.61	4.74	5.72
Lower bound	2.61	3.51	4.73	5.25

with the goal of minimizing the overall completion time, or the makespan. Our main contribution resides in the design of a new online algorithm and 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. For each model, we also prove a lower bound on the competitiveness of our online algorithm, and this lower bound is very close to the constant competitive ratio. All of these new results are summarized in Table 1. Finally, 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 previously known only for task graphs under the roofline model [9], while we derive new results (upper and lower bounds) for 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 a new online algorithm and prove its competitive ratios for different speedup models; we also prove a lower bound for each model. Section 5 is devoted to proving a lower bound of any deterministic online algorithm for the arbitrary speedup model. Finally, Section 6 concludes the paper and suggests 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. [20] 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 [11] 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 $\mathcal{P} = \mathcal{NP}$ [15]. 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 [12] further proposed a polynomial-time approximation scheme (PTAS).

For online scheduling of independent moldable tasks that are released on-the-fly, Ye et al. [23] 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 [7, 10, 17], which studied the communication model.

For offline scheduling of moldable tasks with dependencies, Wang and Cheng [21] showed that the earliest completion time algorithm is a $(3 - 2/P)$-approximation for the roofline model, where P denotes the total number of processors on the platform. For the monotonic model, Lepère et al. [18] proposed an algorithm with approximation ratio $3 + \sqrt{5}$, which was later improved to 4.73 by Jansen and Zhang [14]. Chen and Chu [6] 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.

Table 2: Instances of the scheduling problem.

Problem Instance	Offline	Online
Independent moldable tasks	[11, 12, 20]	[7, 10, 17, 23]
Moldable task graphs	[6, 14, 18, 21]	[9], [This paper]

Feldmann et al. [9] 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 this bound, their algorithm achieves 2.618-competitiveness, even when the task execution times and the DAG structure are unknown. Canon et al. [5] 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.

Benoit et al. [3, 4] recently investigated the problem of scheduling independent moldable tasks subject to failures, 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. We do not consider task failures in this paper, but rather focus on the general online scheduling of moldable task graphs (as in [9]). However, our results can readily carry over to the failure scenario.

Table 2 summarizes the instances of different scheduling problems and the related papers under each instance.

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 $V = \{1, 2, \ldots, n\}$ represents a set of n tasks and $E \subseteq V \times V$ represents a set of precedence constraints (or dependencies) 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. We do not consider costs associated with the data transfers between dependent tasks.

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 [22] (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 [10] (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, and that overhead increases linearly with the number of allocated processors.
- *Amdahl's Model [2] (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, whose dependencies are only revealed upon task completions [1, 5, 9, 16]. Furthermore, when a task j is available, all of its execution time parameters (i.e., $w_j, \bar{p}_j, d_j, c_j$) 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:

$$p_j^{\max} = \min\left(P, \bar{p}_j, \tilde{p}_j\right)\,, \tag{5}$$

where

$$\tilde{p}_j = \begin{cases} \lfloor s_j \rfloor, & \text{if } t_j(\lfloor s_j \rfloor) \le t_j(\lceil s_j \rceil) \\ \lceil s_j \rceil, & \text{otherwise} \end{cases}$$

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 safely assume that the processor allocation of the task should never exceed $p_j^{\max}$ under any reasonable algorithm.

Furthermore, the task is said to satisfy the *monotonic* property [18] if the following two conditions hold:

- 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 $1 \le p < q \le p_j^{\max}$.

Lemma 1. *A task with execution time function given in Equation* (1) *is monotonic if its processor allocation is in the range* $[1, p_j^{\max}]$.

Proof. When the processor allocation is in the range $[1, p_j^{\max}]$, we have $p_j \le p_j^{\max} \le \bar{p}_j$. Thus, the execution time function simplifies to $t_j(p_j) = \frac{w_j}{p_j} + d_j + c_j(p_j - 1)$. This is a convex function whose minimum value is achieved at $\tilde{p}_j$. Since we also have $p_j \le p_j^{\max} \le \tilde{p}_j$, it shows that the execution time is a non-increasing function of p_j in the range $[1, p_j^{\max}]$.

Similarly, when $p_j \le p_j^{\max} \le \bar{p}_j$, the area becomes $a_j(p_j) = p_j \times t_j(p_j) = w_j + d_j p_j + c_j(p_j^2 - p_j)$, which is non-decreasing for any $p_j \ge 1$. □

Thus, based on Lemma 1, the minimum execution time of the task is $t_j^{\min} = t_j(p_j^{\max})$ and the minimum area of the task is $a_j^{\min} = a_j(1)$. Note that the second condition of the monotonic property 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, 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 and critical-path bounds for scheduling any task graph. The minimum choice for both quantities ensures that they can serve as the lower bounds on the optimal makespan. The following lemma states this result.

Lemma 2. $T_{OPT} \ge \max\left(\frac{A_{\min}}{P}, C_{\min}\right)$.

[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 $j_{\pi(i-1)}$.

4 ONLINE ALGORITHM

In this section, we present an online scheduling algorithm and derive its competitive ratio for the general speedup model (Equation (1)) and its three special cases. We also prove lower bounds on the competitiveness of our algorithm under these speedup models.

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, 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. Note that tasks are inserted into the queue without any priority considerations, although in practice certain priority rules may work better.

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 [3, 4]. 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 $\beta_p \leq \frac{1-2\mu}{\mu(1-\mu)}$, where $\mu \leq \frac{3-\sqrt{5}}{2} \approx 0.382$ 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:

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

This step adopts the technique first proposed in [18] and subsequently used in [13, 14]. The purpose is to enable the execution of more tasks at any time, thus potentially increasing the overall resource utilization of the platform and reducing the makespan.

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. The framework combines the analysis shown in [13, 14, 18] for list scheduling as well as the analysis used in [3, 4] for local processor allocation. Together, the result nicely connects the makespan of the online algorithm to the lower bound (Lemma 2), thus proving the competitive ratio.

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 [18], we classify the set of intervals into the following categories.

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

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

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

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

Algorithm 1: Online_Scheduling_Algorithm

```
initialize a waiting queue Q
when at time 0 or a running task completes execution do
    // Processor Allocation
    for each new task j that becomes available do
        Allocate_Processor(j)
        insert task j into the waiting queue Q
    end
    // List Scheduling
    for each task j in the waiting queue Q do
        if there are enough processors to execute the task then
            execute task j now
        end
    end
end
```

Algorithm 2: Allocate_Processor(j)

```
// Step 1: Initial Allocation
Compute p_j^max based on Equation (5)
Compute t_j^min = t_j(p_j^max) and a_j^min = a_j(1)
Find an allocation p_j ∈ [1, p_j^max] from the following optimization problem:
        min_p α_p = a_j(p) / a_j^min
        s.t. β_p = t_j(p) / t_j^min ≤ (1 − 2μ) / (μ(1 − μ))
// Step 2: Allocation Adjustment
if p_j > ⌈μP⌉ then p'_j ← ⌈μP⌉ else p'_j ← p_j
```

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

Lemma 4. *If there exists a constant β such that, for each task j, its initial processor allocation satisfies $t_j(p_j) \leq \beta \times t_j^{\min}$ and $\beta \leq \frac{1}{\mu}$, then we have:*

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

Proof. During T_1 and T_2, the processor utilization is at most $\lceil (1-\mu)P \rceil - 1$, so there are at least $P - (\lceil (1-\mu)P \rceil - 1) \geq \lceil \mu P \rceil$ 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 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 [9, 14, 18].

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 $t_j(p'_j) = t_j(p_j) \leq \beta \times t_j^{\min}$.

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 $t_j(p'_j) \leq \beta \times t_j^{\min} \leq \frac{1}{\mu} \times t_j^{\min}$. Otherwise, if it is reduced, and based on Equation (6), the task execution time should satisfy $\frac{t_j(p'_j)}{t_j^{\min}} = \frac{t_j(\lceil \mu P \rceil)}{t_j(p_j^{\max})} \leq \frac{p_j^{\max}}{\lceil \mu P \rceil} \leq \frac{P}{\mu P} = \frac{1}{\mu}$.

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 $T_1 \leq \beta \times L'_{\min}(f)$ and $T_2 \leq \frac{1}{\mu} \times L''_{\min}(f)$. Thus, we have $\frac{T_1}{\beta} + \mu T_2 \leq L'_{\min}(f) + L''_{\min}(f) \leq L_{\min}(f) \leq C_{\min}$. □

Based on the results of Lemmas 3 and 4, we can now derive a bound on the makespan of the online scheduling algorithm as shown below.

Lemma 5. *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:*

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

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 in the above inequality 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 5 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 Ratios

In this section, we prove competitive ratios for the online algorithm under different speedup models. Based on Lemma 5, the competitive ratio is given by $\frac{\mu\alpha+1-2\mu}{\mu(1-\mu)}$ subject to $\beta \leq \frac{1-2\mu}{\mu(1-\mu)}$. 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 under each considered speedup model. 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 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 6. *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 $\bar{p}$ clearly achieves the minimum execution time $t^{\min} = \frac{w}{\bar{p}}$ for the task. It also achieves the minimum area $a^{\min} = w$, which is not affected by the processor allocation in $[1, \bar{p}]$ due to the task's linear speedup in this range. Thus, this gives $\alpha = \beta = 1$. □

Theorem 1. *Algorithm 1 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. [9][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 $t(p) = \frac{w}{p} + c(p-1)$, with $c > 0$ (if $c = 0$, it simplifies to a special case of the roofline model). For the ease of analysis, we rewrite the execution time function as: $t(p) = c(\frac{w'}{p} + p - 1)$ with $w' = \frac{w}{c}$.

[2] In [9], 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.

Lemma 7. *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 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 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. *Algorithm 1 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 7, 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 $t(p) = \frac{w}{p} + d$, with $d > 0$ (if $d = 0$, it simplifies to a special case of the roofline model). Thus the area function is given by $a(p) = p \times t(p) = w + dp$.

Lemma 8. *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 $\alpha = \frac{a(p)}{a^{\min}} = \frac{w+dp}{w+d} \leq \frac{w+d(x\frac{w}{d}+1)}{w+d} = \frac{w+d+xw}{w+d} = 1 + \frac{xw}{w+d} \leq 1 + x = \alpha_x$. 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. *Algorithm 1 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). For the ease of analysis, we rewrite the execution time function as: $t(p) = c\left(\frac{w'}{\min(p,\bar{p})} + d' + p - 1\right)$ with $w' = \frac{w}{c}$ and $d' = \frac{d}{c}$.

Lemma 9. *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. *Algorithm 1 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 $x^*_\mu \approx 1.972$. □

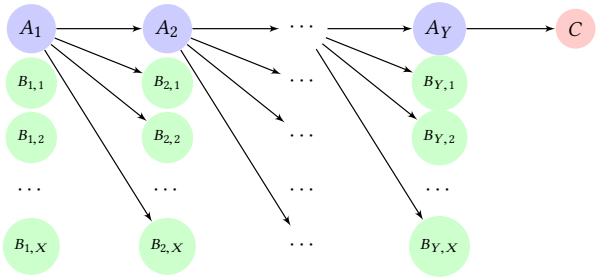

Figure 1: A generic task graph used to prove lower bounds.

4.4 Lower Bounds

In Section 4.3, we have derived the competitive ratios of our online algorithm under several common speedup models. In this section, we show (almost tight) lower bounds on the competitive ratios of our algorithm for these speedup models.

4.4.1 Generic Task Graph and Main Idea. To prove lower bounds, we use a generic task graph as shown in Figure 1. There are $(X + 1)Y + 1$ tasks, partitioned into three different groups: $\mathcal{T}_A, \mathcal{T}_B$ and $\mathcal{T}_C$. Here, X and Y are integers that will be defined differently for each speedup model. Specifically, there are Y identical tasks in $\mathcal{T}_A$, labeled as $(A_i)_{i\in[1,Y]}$, XY identical tasks in $\mathcal{T}_B$, labeled as $(B_{i,j})_{i\in[1,Y],j\in[1,X]}$, and one task in $\mathcal{T}_C$, labeled as C. The tasks are organized in layers and have the following precedence constraints: for any $1 \leq i < Y$, task A_i is the predecessor of task A_{i+1} and of tasks $B_{i+1,j}$ for $1 \leq j \leq X$; and task A_Y is the predecessor of task C. The execution time parameters of tasks will also be defined based on the considered speedup model.

The main idea is to set the parameters in such as way that the schedule of our online algorithm may have the shape as shown in Figure 2(a), with layers scheduled one after another, while the optimal algorithm could deal with the dependencies first to optimize both the critical path and the area, as shown in Figure 2(b).

4.4.2 Roofline Model.

Theorem 5. *Algorithm 1 with* $\mu = \frac{3-\sqrt{5}}{2} \approx 0.382$ *is not better than 2.61-competitive for the roofline model.*

Proof. For this special case, we only need one task, thus $X = Y = 0$. For task C, we set $w = P$ and $\bar{p} = P$, thus $t(p) = \frac{P}{p}$. Let p_C denote the processor allocation chosen by our algorithm for the task. Clearly, our algorithm will choose $p_C = \lceil \mu P \rceil$, and the makespan will satisfy $T = t(p_C) \geq t(\mu P + 1) = \frac{1}{\mu + 1/P}$.

The optimal algorithm could allocate all P processors to the task, resulting in a makespan of $T_{\text{OPT}} = 1$. Therefore, when P gets large enough, we obtain: $\lim_{P\to\infty} \frac{T}{T_{\text{OPT}}} = \frac{1}{\mu} = \frac{3+\sqrt{5}}{2} > 2.61$. □

4.4.3 Communication Model.

Theorem 6. *Algorithm 1 with* $\mu \approx 0.324$ *is not better than 3.51-competitive for the communication model.*

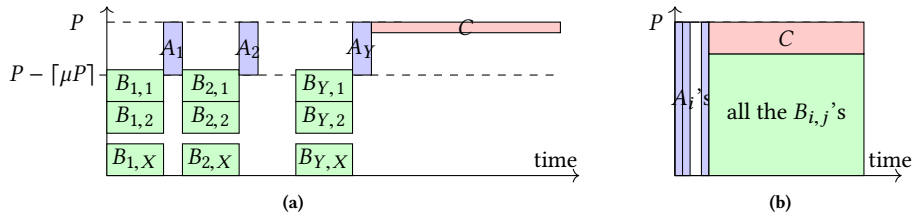

Figure 2: Shapes of our algorithm's schedule (a) and the optimal schedule (b) for the graph of Figure 1.

Proof. Let p_A (resp. p_B, p_C) denote the processor allocation chosen by our algorithm for tasks in $\mathcal{T}_A$ (resp. $\mathcal{T}_B$ and $\mathcal{T}_C$). In this model, we have $\mu \approx 0.324$ and define $\delta = \frac{1-2\mu}{\mu(1-\mu)} \approx 1.61$. We also fix $P > 3$, $X = \left\lfloor \frac{(1-\mu)P}{2} \right\rfloor + 1$ and $Y = P - 3$. Finally, we choose the execution time functions for the tasks in the three groups to be: $t_A(p) = \frac{w_A}{p} + c_A(p-1) = \frac{1}{p}$, $t_B(p) = \frac{w_B}{p} + c_B(p-1) = \frac{\frac{6\delta}{3-\delta}+\frac{1}{P}}{p} + p - 1$, and $t_C(p) = \frac{w_C}{p} + c_C(p-1) = \frac{\delta X w_B}{p} + X w_B \left(\frac{1}{2} - \frac{\delta}{6}\right)(p-1)$.

Processor allocation. For the tasks in $\mathcal{T}_A$, the area is constant, and $\frac{t_A(\mu P)}{t_A^{\min}} = \frac{1}{\mu} > \delta$. Because $t_A(p)$ is decreasing with p, to satisfy the constraint $\beta_p = \frac{t_A(p)}{t_A^{\min}} \leq \delta$, the initial allocation p from Step 1 of Algorithm 2 must be larger than μP. Therefore, the final allocation is exactly $p_A = \lceil \mu P \rceil$ and we get $t_A(p_A) \geq t_A(\mu P + 1) = \frac{1}{\mu P + 1}$.

For the tasks in $\mathcal{T}_B$, we can compute $t_B(1) = \frac{6\delta}{3-\delta} + \frac{1}{P}$, $t_B(2) = \frac{3\delta}{3-\delta} + \frac{1}{2P} + 1$, $t_B(3) = \frac{2\delta}{3-\delta} + \frac{1}{3P} + 2$, and $t_B(4) = \frac{3\delta}{6-2\delta} + \frac{1}{4P} + 3$. We have $t_B(2) - t_B(3) = \frac{\delta}{3-\delta} - 1 + \frac{1}{2P} - \frac{1}{3P} > \frac{\delta}{3-\delta} - 1 > 0$ and $t_B(4) - t_B(3) = 1 - \frac{\delta}{6-2\delta} + \frac{1}{4P} - \frac{1}{3P} > \frac{2}{3} - \frac{\delta}{6-2\delta} > 0$, therefore $t_B^{\min} = t_B(3)$. Furthermore, the initial allocation p must be larger than 1, because:

$$\frac{t_B(1)}{t_B^{\min}} = \frac{\frac{6\delta}{3-\delta} + \frac{1}{P}}{\frac{2\delta}{3-\delta} + \frac{1}{3P} + 2} = \frac{\frac{6\delta}{3-\delta} + \frac{1}{P}}{\frac{6}{3-\delta} + \frac{1}{3P}} = \delta \frac{\frac{6}{3-\delta} + \frac{1}{P\delta}}{\frac{6}{3-\delta} + \frac{1}{3P}} > \delta .$$

The last inequality is because $\delta < 3$. However, $p = 2$ would be an acceptable choice, because:

$$\frac{t_B(2)}{t_B^{\min}} = \frac{\frac{3\delta}{3-\delta} + \frac{1}{2P} + 1}{\frac{2\delta}{3-\delta} + \frac{1}{3P} + 2} = \frac{3 + 2\delta + \frac{3-\delta}{2P}}{6 + \frac{3-\delta}{3P}} < \frac{3 + 2\delta + \frac{3-\delta}{2}}{6} < \delta .$$

The last inequality can be verified by replacing δ with its value. Since $\lceil \mu P \rceil \geq 2$ with $P > 3$, we can conclude that the final allocation is $p_B = 2$, and $t_B(p_B) = \frac{3\delta}{3-\delta} + \frac{1}{2P} + 1 = \frac{w_B}{2} + 1$.

Finally, for task C, we have $t_C(3) = \frac{\delta X w_B}{3} + X w_B - \frac{\delta X w_B}{3} = X w_B$, $t_C(1) = \delta X w_B = \delta t_C(3)$, $t_C(2) = \frac{\delta X w_B}{2} + \frac{X w_B}{2} - \frac{\delta X w_B}{6} = X w_B(\frac{1}{2} + \frac{\delta}{3}) > t_C(3)$ because $\frac{\delta}{3} > \frac{1}{2}$, and $t_C(4) = X w_B \left(\frac{3}{2} - \frac{\delta}{4}\right) > t_C(3)$ because $\frac{\delta}{4} < \frac{1}{2}$. Therefore, $t_C^{\min} = t_C(3)$ and an initial allocation $p = 1$ (thus a final allocation $p_C = 1$) would satisfy the constraint while minimizing the area. We finally have $t_C(p_C) = \delta X w_B$.

Schedule and makespan. By construction, the total number of processors needed to process a layer of tasks in $\mathcal{T}_A$ and $\mathcal{T}_B$ is at least $X p_B + p_A \geq 2(\left\lfloor \frac{(1-\mu)P}{2} \right\rfloor + 1) + \mu P > (1-\mu)P + \mu P = P$. Therefore, a layer cannot be processed in parallel. Assuming in the worst case that our algorithm always prioritizes tasks from $\mathcal{T}_B$ first, it may at best schedule all tasks in $\mathcal{T}_B$ in parallel and then the only task in $\mathcal{T}_A$ for that layer, with a total execution time of $t_B(p_B) + t_A(p_A)$. The same applies to all Y layers, after which the last task C is executed. The makespan of our algorithm is then:

$$\begin{aligned} T &= Y\big(t_A(p_A) + t_B(p_B)\big) + t_C(p_C) \\ &> (P-3)\left(\frac{w_B}{2} + 1\right) + \delta X w_B \\ &= \left(P + \frac{2}{1-\mu}\right)\left(\frac{w_B}{2} + 1\right) + \delta X w_B - \left(3 + \frac{2}{1-\mu}\right)\left(\frac{w_B}{2} + 1\right) \\ &> \frac{2X}{1-\mu}\left(\frac{w_B}{2} + 1\right) + \delta X w_B - 30 = \frac{X(w_B+2)}{1-\mu} + \delta X w_B - 30 . \end{aligned}$$

The second inequality above is due to the following two facts:

- $P + \frac{2}{1-\mu} \geq \frac{2X}{1-\mu} \Leftrightarrow (1-\mu)P + 2 \geq 2X \Leftrightarrow X \leq \frac{(1-\mu)P}{2} + 1$
- $\left(3 + \frac{2}{1-\mu}\right)\left(\frac{w_B}{2} + 1\right) < 30$ (verified by plugging in values)

Lower bound on competitive ratio. We now consider an alternative schedule that works as follows: First, allocate all P processors to each task in $\mathcal{T}_A$ (with execution time $t_A^* = t_A(P) = \frac{1}{P}$), and execute them sequentially. Then, allocate 1 processor to each task in $\mathcal{T}_B$ (with execution time $t_B^* = t_B(1) = w_B$) and 3 processors to task C (with execution time $t_C^* = t_C(3) = X w_B$). It will execute task C whenever the tasks in $\mathcal{T}_A$ are completed, and the remaining processors are used to execute the tasks in $\mathcal{T}_B$ in groups of $P-3$. The optimal makespan would be at least as good as the one produced by this schedule, which is given by:

$$\begin{aligned} T_{\text{OPT}} &\leq Y t_A^* + \max(t_C^*, X t_B^*) \\ &\leq 1 + X w_B . \end{aligned}$$

Therefore, when P gets large enough (thus X gets large enough), we obtain the following lower bound on the competitive ratio:

$$\begin{aligned} \lim_{P\to\infty} \frac{T}{T_{\text{OPT}}} &\geq \lim_{P\to\infty} \frac{\frac{X(w_B+2)}{1-\mu} + \delta X w_B - 30}{1 + X w_B} \\ &= \lim_{X\to\infty} \frac{\frac{X(w_B+2)}{1-\mu} + \delta X w_B}{X w_B} \\ &= \frac{1}{1-\mu} + \frac{2}{(1-\mu)w_B} + \delta \\ &> \frac{1}{\mu} + \frac{\mu}{1-2\mu} - \frac{1}{3(1-\mu)} > 3.51 . \end{aligned}$$

□

4.4.4 Amdahl's Model.

Theorem 7. *Algorithm 1 with $\mu \approx 0.271$ is not better than 4.73-competitive for the Amdahl model.*

Proof. Again, let p_A (resp. p_B, p_C) denote the processor allocation chosen by our algorithm for tasks in $\mathcal{T}_A$ (resp. $\mathcal{T}_B$ and $\mathcal{T}_C$). In this model, we have $\mu \approx 0.271$ and $\delta = \frac{1-2\mu}{\mu(1-\mu)} \approx 2.32$. We fix an integer $K > 3$, $P = K^2$, $X = \left\lfloor \frac{K^2(1-\mu)}{p_B} \right\rfloor + 1$ and $Y = \left\lfloor \frac{K(K-\delta)}{X} \right\rfloor$. We choose the execution time functions for the tasks to be: $t_A(p) = \frac{w_A}{p} + d_A = \frac{K}{p}$, $t_B(p) = \frac{w_B}{p} + d_B = \frac{K}{p} + 1$, and $t_C(p) = \frac{w_C}{p} + d_C = \frac{(\delta-1)K}{p} + K$.

Processor allocation. For the tasks in $\mathcal{T}_A$, the area is constant, and $\frac{t_A(\mu P)}{t_A^{\min}} = \frac{1}{\mu} > \delta$. Because $t_A(p)$ is decreasing with p, to satisfy the constraint $\beta_p = \frac{t_A(p)}{t_A^{\min}} \leq \delta$, the initial allocation p from Step 1 of Algorithm 2 must be larger than μP. Therefore, the final allocation is exactly $p_A = \lceil \mu P \rceil$ and we get $t_A(p_A) \geq t_A(\mu P + 1) = \frac{K}{\mu P + 1}$.

For the tasks in $\mathcal{T}_B$, we first let p^* be the processor allocation if relaxing the constraint that p must be an integer. From the optimization problem in Algorithm 2, as α_p increases with p and β_p decreases with p, to minimize α_p subject to $\beta_p \leq \delta$, we must have:

$$\frac{t_B(p^*)}{t_B^{\min}} = \delta \Leftrightarrow \frac{K}{p^*} + 1 = \delta\left(\frac{K}{K^2} + 1\right)$$
$$\Leftrightarrow p^* = \frac{K}{\delta(\frac{1}{K} + 1) - 1} \Rightarrow \frac{K}{\delta - 1} - 2 \leq p^* \leq \frac{K}{\delta - 1}.$$

The right inequality of the last implication is immediate, whereas the left inequality comes from:

$$\frac{K}{\delta - 1} - 2 \leq \frac{K}{\delta(\frac{1}{K} + 1) - 1} \Leftrightarrow (K - 2\delta + 2)(\frac{\delta}{K} + \delta - 1) \leq \delta K - K$$
$$\Leftrightarrow 5\delta - 2\delta^2 - 2 - \frac{2\delta}{K}(\delta - 1) \leq 0$$
$$\Leftarrow 5\delta - 2\delta^2 - 2 \leq 0$$

The last inequality above is true due to our choice of μ. Now, to respect the constraint $\beta_p \leq \delta$ and to minimize α_p, the initial processor allocation must satisfy $p = \lceil p^* \rceil$, which implies $\frac{K}{\delta-1} - 2 \leq p \leq \frac{K}{\delta-1} + 1$. Finally, we can show that $p \leq \frac{K}{\delta-1} + 1 \leq \mu K^2 \leq \lceil \mu P \rceil$ for any integer $K > 3$. Thus, the final allocation is $p_B = p$, and we have $t_B(p_B) \geq t_B(\frac{K}{\delta-1} + 1) = \frac{K}{\frac{K}{\delta-1}+1} + 1$.

Lastly, for task C, we have $t_C^{\min} \geq d_C = K$ and $t_C(1) = (\delta - 1)K + K = \delta K \leq \delta t_C^{\min}$. Therefore, allocating 1 processor respects the constraint $\beta_p \leq \delta$ and clearly minimizes α_p. Thus, we have $p_C = 1$ and $t_C(p_C) = \delta K$.

Schedule and makespan. By construction, the total number of processors needed to process a layer of tasks in $\mathcal{T}_A$ and $\mathcal{T}_B$ is at least $Xp_B + p_A \geq \left(\left\lfloor \frac{K^2(1-\mu)}{p_B} \right\rfloor + 1\right) p_B + \mu K^2 > \frac{K^2(1-\mu)}{p_B} p_B + \mu K^2 = K^2 = P$. Therefore, a layer cannot be processed in parallel. Assuming in the worst case that our algorithm always prioritizes tasks from $\mathcal{T}_B$ first, it may at best schedule all tasks in $\mathcal{T}_B$ in parallel and then the only task in $\mathcal{T}_A$ for that layer, with a total execution time of $t_B(p_B) + t_A(p_A)$. The same applies to all Y layers, after which the last task C is executed. The makespan of our algorithm is then:

$$\begin{aligned} T &= Y\big(t_A(p_A) + t_B(p_B)\big) + t_C(p_C) \\ &\geq \left(\frac{K(K-\delta)}{X} - 1\right)\left(\frac{K}{\mu K^2 + 1} + \frac{K}{\frac{K}{\delta-1} + 1} + 1\right) + \delta K \\ &> \left(\frac{K(K-\delta)}{\frac{K^2(1-\mu)}{p_B} + 1} - 1\right)\left(\frac{K}{\frac{K}{\delta-1} + 1} + 1\right) + \delta K \\ &\geq \left(\frac{K - \delta}{\frac{1-\mu}{\frac{1}{\delta-1} - \frac{2}{K}} + \frac{1}{K}} - 1\right)\left(\frac{1}{\frac{1}{\delta-1} + \frac{1}{K}} + 1\right) + \delta K\,. \end{aligned}$$

Lower bound on competitive ratio. We now consider an alternative schedule that works as follows: First, allocate all P processors to each task in $\mathcal{T}_A$ (with execution time $t_A^* = t_A(P) = \frac{1}{K}$), and execute them sequentially. Then, allocate 1 processor to each task in $\mathcal{T}_B$ (with execution time $t_B^* = t_B(1) = K + 1$) and $\lceil (\delta - 1)K \rceil < \delta K$ processors to task C (with execution time $t_C^* \leq t_C((\delta-1)K) = 1+K$). The total number of processors required for all tasks in $\mathcal{T}_B$ and task C is less than $XY + \delta K \leq K(K - \delta) + \delta K = K^2 = P$, so they can be executed in parallel. The optimal makespan would be at least as good as the one produced by this schedule, which is given by:

$$\begin{aligned} T_{\text{OPT}} &\leq Yt_A^* + \max(t_B^*, t_C^*) = \frac{Y}{K} + K + 1 < \frac{K}{X} + K + 1 \\ &\leq \frac{p_B}{K(1-\mu)} + K + 1 \leq \frac{2}{1-\mu} + K + 1 < K + 4\,. \end{aligned}$$

The second last inequality above uses $p_B \leq \frac{K}{\delta-1} + 1 \leq 2K$. Therefore, when K gets large enough, we obtain the following lower bound on the competitive ratio:

$$\begin{aligned} \lim_{K\to\infty} \frac{T}{T_{\text{OPT}}} &\geq \lim_{K\to\infty} \frac{\left(\frac{K-\delta}{\frac{1-\mu}{\frac{1}{\delta-1} - \frac{2}{K}} + \frac{1}{K}} - 1\right)\left(\frac{1}{\frac{1}{\delta-1} + \frac{1}{K}} + 1\right) + \delta K}{K + 4} \\ &= \lim_{K\to\infty} \frac{\left(\frac{K}{(\delta-1)(1-\mu)} - 1\right)(\delta - 1 + 1) + \delta K}{K} \\ &= \frac{\delta}{(\delta-1)(1-\mu)} + \delta > 4.73\,. \end{aligned}$$

□

4.4.5 General Model.

Theorem 8. *Algorithm 1 with $\mu \approx 0.211$ is not better than 5.25-competitive for the general model.*

Proof. We can use the exact same instance as for the Amdahl model (Section 4.4.4), but with $\mu \approx 0.211$ and $\delta = \frac{1-2\mu}{\mu(1-\mu)} \approx 3.47$. Indeed, all the derivations still hold, including the condition $5\delta - 2\delta^2 - 2 \leq 0$. We get $\lim_{K\to\infty} \frac{T}{T_{\text{OPT}}} \geq \frac{\delta}{(\delta-1)(1-\mu)} + \delta > 5.25$. □

5 A LOWER BOUND OF ANY DETERMINISTIC ONLINE ALGORITHM FOR ARBITRARY SPEEDUP MODEL

So far, we have focused on the general speedup model of Equation (1) and its special instances. In this section, we show that the competitive ratio of any deterministic online algorithm (including ours) can be unbounded under an arbitrary speedup model[3].

Theorem 9. *Any deterministic online algorithm is at least* $\Omega(\ln(D))$*-competitive for scheduling moldable task graphs under an 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 3 shows such an instance for $\ell = 2, K = 4$ and $n = 15$. All tasks in the graph are identical, with an execution time function $t(p) = \frac{1}{\lg(p)+1}$. Here, lg denotes logarithm to the base 2. We set the total number of processors to be $P = K \times 2^{K-1}$.

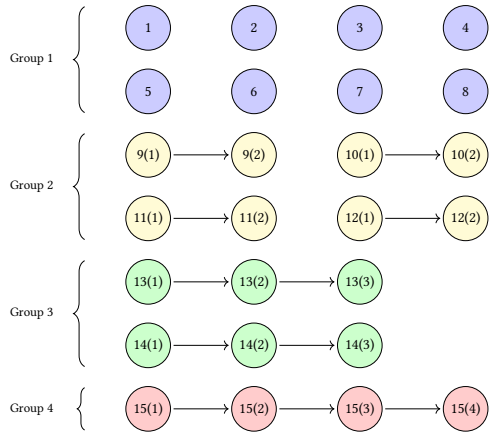

Figure 3: A lower bound instance in Theorem 9 with $\ell = 2$, $K = 4$, and $n = 15$ linear task chains. Each circle represents a task and the number inside each circle indicates the ID of the linear chain the task is in (and the number in the parenthesis indicates the task's position in that linear chain).

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 4(a) illustrates the schedule for this 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 $j \le i$, 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. □

Lemma 10. *Any deterministic online algorithm could produce a schedule that satisfies:* $t_i - t_{i-1} \ge \frac{1}{\ell+i}$, $\forall i \in [1, K]$.

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}.

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 $t(p) = \frac{1}{\lg(p)+1} \le \frac{1}{\ell+i}$, hence their processor allocation must be at least $p \ge 2^{\ell+i-1} = K \times 2^{i-1}$. 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 $2^{K-i} \times a(K \times 2^{i-1}) = \frac{2^{K-i} \times K \times 2^{i-1}}{\lg(K \times 2^{i-1})+1} = \frac{K \times 2^{K-1}}{\ell+i} = \frac{P}{\ell+i}$. 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 4(b) illustrates this strategy for the same instance with $\ell = 2$.

Finally, we can use the result of Lemma 10 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 for all 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:

$$t_K \ge \sum_{i=1}^{K} \frac{1}{\ell+i} > \sum_{i=\ell+1}^{K} \frac{1}{i} = \sum_{i=1}^{K} \frac{1}{i} - \sum_{i=1}^{\ell} \frac{1}{i}$$
$$> (\ln(K) + \gamma) - \left(\ln(\ell) + \gamma + \frac{1}{\ell}\right) = \ln(K) - \ln(\ell) - \frac{1}{\ell}.$$

6 CONCLUSION AND FUTURE WORK

In this paper, we have studied the online scheduling of moldable task graphs to minimize makespan with tasks obeying different speedup models. To the best of our knowledge, no competitive ratio was known under this setting, except for the roofline model [9]. Owing to the design of a new online algorithm, we have extended the result and derived competitive ratios for several other speedup models, including the communication model, Amdahl's model and a general combination. For each of these models, we have also derived

[3]Here, an arbitrary speedup model means that the execution time $t(p)$ of a task can take any arbitrary function of its processor allocation p.

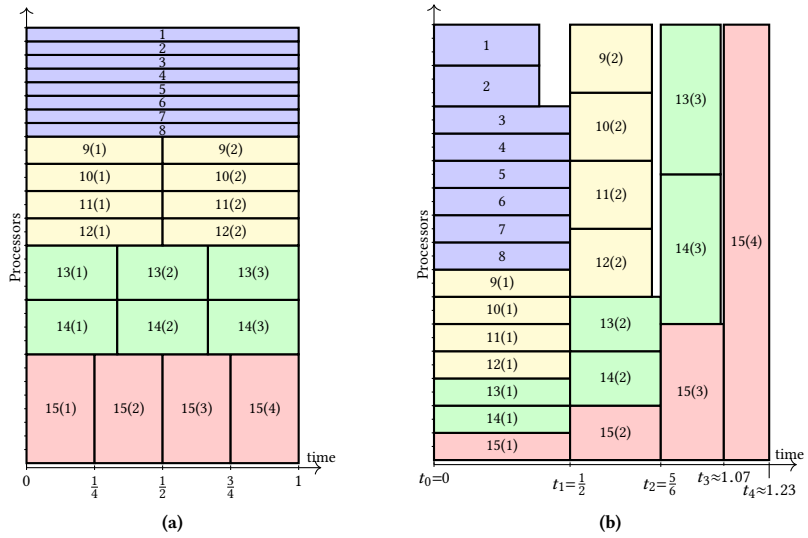

Figure 4: For the lower bound instance of Figure 3: (a) An offline schedule with a makespan of 1; (b) An online algorithm's schedule, allocating (approximately) the same number of processors to all linear chains and producing a makespan of $t_4 \approx 1.23$.

a lower bound on the competitiveness of our online algorithm that is very close to the upper bound. Finally, we have considered the arbitrary speedup model and established a lower bound for any deterministic online algorithm. Altogether, these new results lay the foundations for further study of this important but difficult scheduling problem.

For future work, we will aim at further improving the algorithm and the competitive ratios obtained in this work under the considered speedup models, as well as extending them to other common speedup models. We also plan to extend our algorithm and analysis to other online scheduling settings (e.g., for independent tasks released over time, and for special task graphs such as fork-join graphs or trees). Finally, we will expand this study to a more practical side by experimentally evaluating the performance of our algorithm using realistic workflows. We anticipate that our algorithm will perform much better practically than that predicted by the worst-case competitive ratios.

ACKNOWLEDGMENTS

This research is supported in part by the NSF grant 2135310. We thank the anonymous reviewers for valuable comments, which helped improve the presentation of this paper.

REFERENCES

[1] Kunal Agrawal, Charles E. Leiserson, and Jim Sukha. 2010. Executing task graphs using work-stealing. In *IPDPS*. 1–12.
[2] Gene M. Amdahl. 1967. Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. In *AFIPS'67*. 483–485.
[3] Anne Benoit, Valentin Le Fèvre, Lucas Perotin, Padma Raghavan, Yves Robert, and Hongyang Sun. 2020. Resilient scheduling of moldable jobs on failure-prone platforms. In *IEEE Cluster*.
[4] Anne Benoit, Valentin Le Fèvre, Lucas Perotin, Padma Raghavan, Yves Robert, and Hongyang Sun. 2022. Resilient Scheduling of Moldable Parallel Jobs to Cope with Silent Errors. *IEEE Trans. Comput.* 71, 07 (2022), 1696–1710.
[5] Louis-Claude Canon, Loris Marchal, Bertrand Simon, and Frédéric Vivien. 2020. Online Scheduling of Task Graphs on Heterogeneous Platforms. *IEEE Trans. Parallel Distributed Syst.* 31, 3 (2020), 721–732.
[6] Chi-Yeh Chen and Chih-Ping Chu. 2013. A 3.42-Approximation Algorithm for Scheduling Malleable Tasks under Precedence Constraints. *IEEE Trans. Parallel Distrib. Syst.* 24, 8 (2013), 1479–1488.
[7] Richard A. Dutton and Weizhen Mao. 2007. Online Scheduling of Malleable Parallel Jobs. In *PDCS* (Cambridge, Massachusetts). 136–141.
[8] Dror G. Feitelson and Larry Rudolph. 1996. Toward convergence in job schedulers for parallel supercomputers. In *Job Scheduling Strategies for Parallel Processing*. Springer, 1–26.
[9] Anja Feldmann, Ming-Yang Kao, Jiří Sgall, and Shang-Hua Teng. 1998. Optimal On-Line Scheduling of Parallel Jobs with Dependencies. *Journal of Combinatorial Optimization* 1, 4 (1998), 393–411.
[10] Jessen T. Havill and Weizhen Mao. 2008. Competitive online scheduling of perfectly malleable jobs with setup times. *European Journal of Operational Research* 187 (2008), 1126–1142.
[11] Klaus Jansen. 2012. A $(3/2+\epsilon)$ Approximation Algorithm for Scheduling Moldable and Non-moldable Parallel Tasks. In *SPAA*. 224–235.
[12] K. Jansen and F. Land. 2018. Scheduling Monotone Moldable Jobs in Linear Time. In *IPDPS*. 172–181.
[13] Klaus Jansen and Hu Zhang. 2005. Scheduling Malleable Tasks with Precedence Constraints. In *SPAA*. 86–95.
[14] Klaus Jansen and Hu Zhang. 2006. An Approximation Algorithm for Scheduling Malleable Tasks Under General Precedence Constraints. *ACM Trans. Algorithms* 2, 3 (2006), 416–434.
[15] Berit Johannes. 2006. Scheduling Parallel Jobs to Minimize the Makespan. *J. of Scheduling* 9, 5 (2006), 433–452.
[16] Theodore Johnson, Timothy A. Davis, and Steven M. Hadfield. 1996. A concurrent dynamic task graph. *Parallel Comput.* 22, 2 (1996), 327–333.
[17] Nathaniel Kell and Jessen Havill. 2015. Improved Upper Bounds for Online Malleable Job Scheduling. *J. of Scheduling* 18, 4 (2015), 393–410.
[18] Renaud Lepère, Denis Trystram, and Gerhard J. Woeginger. 2001. Approximation Algorithms for Scheduling Malleable Tasks Under Precedence Constraints. In *ESA*. 146–157.
[19] Daniel D. Sleator and Robert E. Tarjan. 1985. Amortized Efficiency of List Update and Paging Rules. *Commun. ACM* 28, 2 (1985), 202–208.
[20] John Turek, Joel L. Wolf, and Philip S. Yu. 1992. Approximate Algorithms Scheduling Parallelizable Tasks. In *SPAA* (San Diego, California, USA).
[21] Qingzhou Wang and Kam Hoi Cheng. 1992. A Heuristic of Scheduling Parallel Tasks and Its Analysis. *SIAM J. Comput.* 21, 2 (1992), 281–294.
[22] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: An Insightful Visual Performance Model for Multicore Architectures. *Commun. ACM* 52, 4 (2009), 65–76.
[23] Deshi Ye, Danny Z. Chen, and Guochuan Zhang. 2018. Online Scheduling of Moldable Parallel Tasks. *J. of Scheduling* 21, 6 (2018), 647–654.