Improved Online Scheduling of Moldable Task Graphs under Common Speedup Models

LUCAS PEROTIN, LIP, ENS Lyon, France
HONGYANG SUN, University of Kansas, USA

We consider the online scheduling problem of moldable task graphs on multiprocessor systems for minimizing the overall completion time (or makespan). Moldable job scheduling has been widely studied in the literature, 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 article, we design a new online scheduling algorithm for this problem and derive constant competitive ratios under several common yet realistic speedup models (i.e., roofline, communication, Amdahl, and a general combination). These results improve the ones we have shown in the preliminary version of the article. We also prove, for each speedup model, a lower bound on the competitiveness of any online list scheduling algorithm that allocates processors to a task based only on the task's parameters and not on its position in the graph. This lower bound matches exactly the competitive ratio of our algorithm for the roofline, communication, and Amdahl's model, and is close to the ratio for the general model. Finally, we provide a 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.

CCS Concepts: • **Theory of computation → Online algorithms**; **Scheduling algorithms;**

Additional Key Words and Phrases: Task graph, moldable task, online scheduling, competitive ratio

ACM Reference format:
Lucas Perotin and Hongyang Sun. 2024. Improved Online Scheduling of Moldable Task Graphs under Common Speedup Models. *ACM Trans. Parallel Comput.* 11, 1, Article 2 (March 2024), 31 pages.
https://doi.org/10.1145/3630052

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) [12].

A preliminary version [7] of this article appeared in the 51st International Conference on Parallel Processing (ICPP'22).
This work is supported in part by the US National Science Foundation grant #2135310.
Authors' addresses: L. Perotin, Laboratoire LIP, ENS Lyon, 46 Allée d'Italie, 69364 Lyon Cedex 07, France; e-mail: lucas.perotin@ens-lyon.fr; H. Sun, Department of EECS, University of Kansas, 1520 W 15th St Lawrence, KS 66045, USA; e-mail: hongyang.sun@ku.edu.

2329-4949/2024/03-ART2 $15.00
https://doi.org/10.1145/3630052

Moldable tasks offer a nice tradeoff 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. Thus, many computational kernels in scientific libraries (e.g., for numerical linear algebra and tensor computations) can be deployed as moldable tasks to efficiently run on a wide range of processors. Because of the importance and wide availability of moldable tasks, scheduling algorithms for such tasks have received considerable attention in the literature (see Section 2 for details). Like many other scheduling problems, scheduling moldable tasks comes in different flavors, and the following provides a brief taxonomy:

- **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 for both independent and dependent tasks [30], and the goal is to design scheduling algorithms with good *approximation ratios*, which measure the worst-case performance of an algorithm against an optimal scheduler for all possible input instances. On the contrary, in the online version of the scheduling problem, tasks are released on-the-fly, and the objective is to design online algorithms with good *competitive ratios* [28], against an optimal offline scheduler that knows in advance all the tasks 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.** In a scheduling problem, different tasks can be either independent of each other or dependent forming a task graph. If tasks are independent, then they are either all known to the scheduling algorithm initially (in the offline version) or released on-the-fly and the scheduler only discovers their characteristics upon release (in the online version). For task graphs, either the entire graph is known at the start (in the offline version) or each new task along with its characteristics is only released when all of its predecessors have completed execution (in the online version). For the latter case, the shape of the graph as well as the nature of the tasks are not known in advance, and they are revealed only as the execution progresses.

In this article, we investigate arguably the most difficult version of the moldable task scheduling problem, namely, the *online scheduling of tasks graphs*, which is the version that has received the least research attention so far. The objective is to minimize the overall completion time of the task graph, or the *makespan*. We assume that the scheduling of each task is non-preemptive and without restarts [13], which is highly desirable to avoid large overheads incurred by checkpointing partial results, context switching, task migration, and so on.

The performance of a scheduling algorithm greatly depends upon the speedup model, which specifies the speedup (or equivalently the execution time) of a task as a function of its processor allocation. We consider several common yet realistic speedup models, including the roofline model, the communication model, the Amdahl's model, and a general combination of them (see Section 3.1 for their precise definitions). These models have been widely assumed and studied in the literature for modeling the scaling behavior of parallel applications. We extend and improve upon our preliminary work [7] by designing a new online scheduling algorithm that achieves better constant competitive ratios for these models. This is done through a novel analysis framework, which provides a tighter and more coupled analysis between a local processor allocation algorithm and an online list scheduling algorithm. To the best of our knowledge, a competitive ratio was previously known only for task graphs under the roofline model [13], while our work offers the first competitive results for several other speedup models. Besides these common models, we also consider the arbitrary speedup model, which allows the speedup (or execution time) of a task to take any arbitrary function of its processor allocation. We show the difficulty of scheduling

Table 1. Competitive Ratios of Our Algorithm in This Article and the Algorithm in [7] for Four Common Speedup Models and a Lower Bound for the Arbitrary Speedup Model

Speedup Model	Roofline	Comm.	Amdahl	General	Arbitrary (Lower Bound)
Results of this article	≈ 2.62	≈ 3.39	≈ 4.55	≈ 4.63	$\Omega(\ln(D))$
Results of [7]	≈ 2.62	≈ 3.61	≈ 4.74	≈ 5.72	

under such a model by proving a lower bound in this case. Altogether, these results lay the theoretical foundations for this difficult but practical scheduling problem.

Our main contributions are summarized as follows:

— We present a new online algorithm and prove its constant competitive ratio for the four speedup models. In particular, the results for the communication model, the Amdahl's model, and the general model improve upon those previously obtained in [7]. Table 1 lists our results in comparison with the ones proven in [7].
— For each speedup model, we prove a lower bound on the competitiveness of any online list scheduling algorithm whose processor allocation is *local and deterministic*, i.e., the decision depends only on the total number of processors and a task's parameters but not on its position in the graph. The results show that our algorithm achieves the best possible competitive ratios for the roofline, communication, and Amdhal's models for this class of algorithms.
— We derive a lower bound on the competitiveness of any deterministic online algorithm for the arbitrary speedup model and show that it is not constant but grows logarithmically with the number of tasks D in the longest path of the graph. Table 1 also includes this lower bound result.

The rest of this article is organized as follows: Section 2 surveys some related works on moldable task scheduling. The formal model and problem statement are presented in Section 3. Section 4 introduces the new online algorithm and proves its competitive ratios for different speedup models. Section 5 presents, for each model, a lower bound of any online list scheduling algorithm with deterministic local processor allocation. Our algorithm belongs to this class and has the best possible competitive ratio for the roofline, communication, and Amdahl's models. Section 6 is devoted to proving a lower bound of any deterministic online algorithm for the arbitrary speedup model. Finally, Section 7 concludes the article and provides hints for future directions.

2 RELATED WORKS

In this section, we discuss some related works on moldable task scheduling. Following the taxonomy of the previous section, we consider four versions of the problem combining offline vs. online scheduling and independent tasks vs. task graphs scheduling. We mainly focus on works that have derived approximation or competitive ratios. While some of these results depend on specific speedup models, others hold for a more general class of models.

Offline Scheduling of Independent Tasks. Belkhale and Banerjee [3] considered moldable tasks that follow the monotonic model, where the execution time of a task is non-increasing and the area (processor allocation times execution time) is non-decreasing with the number of processors. They presented a $\frac{2}{1+1/P}$-approximation algorithm by iteratively updating the processor allocations. For the same model, Błażewicz et al. [8] also presented a 2-approximation algorithm while relying on an optimal continuous schedule. Mounié et al. [26] presented a $(\sqrt{3}+\epsilon)$-approximation algorithm using dual approximation. They later improved the ratio to $1.5+\epsilon$ [27]. Finally, Jansen and Land [17] proposed a **Polynomial-Time Approximation Scheme (PTAS)** for the problem.

If the execution time of a task can be an arbitrary function of the processor allocation (i.e., the arbitrary model), Turek et al. [29] designed a 2-approximation list-based algorithm and a

3-approximation shelf-based algorithm. Ludwig and Tiwari [25] showed the same result but with lower computational complexity. When each task only admits a subset of all possible processor allocations, Jansen [16] presented a $(1.5 + \epsilon)$-approximation algorithm, which is tight, since the problem cannot have an approximation ratio better than 1.5 unless $\mathcal{P} = \mathcal{NP}$ [21]. When the number of processors is a constant or polynomially bounded by the number of tasks, Jansen et al. [18] showed that a PTAS exists.

Online Scheduling of Independent Tasks. For scheduling independent moldable tasks that arrive online over time, Havill and Mao [15] presented a 4-competitive algorithm for the communication model by minimizing the execution time of each task. For the same model, Dutton and Mao [11] and Kell and Havill [23] further presented improved competitive results when the total number P of processors is bounded by a constant.

Under an alternative online setting, where independent moldable tasks with the same arrival time are released one-by-one to the scheduler, Ye et al. [33] designed a 16.74-competitive algorithm for the arbitrary speedup model. They also showed how to transform an online algorithm for rigid tasks whose makespan is at most ρ times the lower bound into a 4ρ-competitive algorithm for moldable tasks.

Offline Scheduling of Task Graphs. For offline scheduling of moldable task graphs, Wang and Cheng [31] showed that the earliest completion time algorithm is a $(3 - \frac{2}{P})$-approximation for the roofline model. Since the processor allocation is done independently for each task, their algorithm and corresponding ratio can also be applied to the online setting as discussed below.

For the monotonic model, Belkhale and Banerjee [4] presented a 2.618-approximation algorithm while assuming the availability of an optimal processor allocation. Lepère et al. [24] proposed an algorithm with an approximation ratio of 5.236. They also showed that the optimal allocation can be achieved in pseudo-polynomial time for some special graphs, such as series-parallel graphs and trees, thus leading to a 2.618-approximation for these graphs. Jansen and Zhang [20] later improved the approximation ratio for general graphs to around 4.73. When assuming that the area of a job is a concave function of the number of processors, Jansen and Zhang [19] proposed a 3.29-approximation algorithm. Chen and Chu [10] improved the ratio to around 2.95 by further assuming that the execution time of a job is strictly decreasing in the number of allocated processors.

Online Scheduling of Task Graphs. Feldmann et al. [13] designed an online algorithm to schedule moldable task graphs under the roofline model. They showed that their algorithm achieves a competitive ratio of 2.618, thus improving the previous result by Wang and Cheng [31]. Furthermore, their algorithm works in the *non-clairvoyant* setting, where the task execution time is also unknown to the scheduler.

Benoit et al. [5, 6] recently investigated the problem of scheduling moldable tasks subject to failures, where a task needs to be re-executed after a failure until it is successfully completed. This corresponds to a special task graph consisting of multiple linear chains (one per task), where the length of each chain corresponds to the total number of executions of a task. The problem is semi-online, since all the tasks are known at the beginning, but the task failures and hence their re-executions are only discovered on-the-fly. They considered several common speedup models (as in this article) and presented a scheduling algorithm that achieves constant competitive ratios for these models. In this article, we extend our preliminary work [7] and study the general online scheduling of moldable task graphs (as in Reference [13]). We present improved competitive ratios as well as lower bounds for the common speedup models. We do not consider task failures as in References [5, 6], but our results can readily carry over to the failure scenario.

Table 2. Difference Versions of Moldable Task Scheduling Problem and Related Papers in each Version

	Offline Scheduling	Online Scheduling
Independent Tasks	*Monotonic model*: [3, 8, 17, 26, 27] *Arbitrary model*: [16, 17, 25, 29]	*Comm. model (over time)* [11, 15, 23] *Arbitrary model (one-by-one)*: [33]
Task Graphs	*Roofline model*: [31] *Monotonic model*: [4, 10, 19, 20, 24]	*Roofline model*: [13, 31] *Common models*: [5–7], [this article]

Table 2 summarizes different versions of the moldable task scheduling problem together with the related papers within each version.

3 PROBLEM STATEMENT

In this section, we formally present the online scheduling model and objective function. We also show a simple lower bound on the optimal makespan, against which the performance of our online algorithms will be measured. We finally derive the minimum execution time and minimum area of any task, which will be used in our subsequent analysis.

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. In this work, we do not consider the 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 article, we mainly 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*** [32] (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 $\bar{p}_j$.

— ***Communication Model*** [15] (with $\bar{p}_j \geq P$ 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, which increases linearly with the number of allocated processors.

— ***Amdahl's Model*** [2] (with $\bar{p}_j \geq P$ 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: $a_j(p_j) = p_j t_j(p_j)$. 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, 9, 13, 22]. 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 as well. 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., $\frac{T}{T^{\text{opt}}} \leq c$. Note that the optimal offline scheduler knows in advance all the tasks and their speedup models, as well as all dependencies in the graph. 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

We now state a simple and well-known lower bound on the optimal makespan. Let t_j^{opt} and a_j^{opt} denote, respectively, the execution time and the area of task j under the processor allocation of an optimal schedule. The two quantities defined below can be used as a lower bound of the optimal makespan.

Definition 1. Given the processor allocations of all the tasks in an optimal schedule,

— the ***total area*** A^{opt} of the task graph is the sum of the areas of all the tasks in the graph, i.e., $A^{\text{opt}} = \sum_{j=1}^{n} a_j^{\text{opt}}$.

— the length $L^{\text{opt}}(f)$ of a path[1] f in the graph is the sum of the execution times of all the tasks along that path, i.e., $L^{\text{opt}}(f) = \sum_{j \in f} t_j^{\text{opt}}$. The ***critical path length*** C^{opt} of the graph is the length of the longest path in the graph, i.e., $C^{\text{opt}} = \max_f L^{\text{opt}}(f)$.

Clearly, the optimal makespan cannot be smaller than $\frac{A^{\text{opt}}}{P}$ and C^{opt}. This follows from the well-known area and critical-path bounds for scheduling any task graph [14]. The following lemma states this result:

Lemma 1. $T^{opt} \geq \max(\frac{A^{opt}}{P}, C^{opt})$.

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

3.3 Minimum Execution Time and Area of a Task

This section derives the minimum execution time and minimum area of any task j. Given the execution time function in Equation (1), we first define a ***critical*** processor count for task j as follows:

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

where $s_j = \sqrt{\frac{w_j}{c_j}}$. Here, $\tilde{p}_j$ represents a processor count beyond which the task's execution time can no longer decrease (the same is also true for $\bar{p}_j$).

Lemma 2. *The execution time of task j is non-decreasing for any $p_j > \min(\bar{p}_j, \tilde{p}_j)$.*

Proof. For $p_j > \bar{p}_j$, the execution time function simplifies to $t_j(p_j) = \frac{w_j}{\bar{p}_j} + d_j + c_j(p_j - 1)$, which is obviously an increasing function of p_j. For $p_j \leq \bar{p}_j$, the execution time function becomes $t_j(p_j) = \frac{w_j}{p_j} + d_j + c_j(p_j - 1)$, which is a convex function of p_j whose minimum value is achieved at $s_j = \sqrt{\frac{w_j}{c_j}}$. Since the processor allocation must be an integer, the minimum execution time is then achieved at $\tilde{p}_j$. Thus, the execution time is non-decreasing for any $p_j > \tilde{p}_j$. □

We now define the maximum number of processors that should be allocated to task j as follows:

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

Indeed, allocating more than $p_j^{\max}$ processors to the task either is impossible (if $p_j^{\max} = P$) or will no longer decrease its execution time while only increasing its area (based on Lemma 2). Thus, we assume that the processor allocation of the task never exceeds $p_j^{\max}$ by any reasonable algorithm.

Furthermore, we say that a task j is ***monotonic*** if it satisfies the following two monotonic properties [24]:

- The execution time is a ***non-increasing*** function of the processor allocation, i.e., $t_j(p) \geq t_j(q)$ for all $1 \leq p < q \leq P$;
- The area is a ***non-decreasing*** function of the processor allocation, i.e., $a_j(p) \leq a_j(q)$ for all $1 \leq p < q \leq P$.

Note that the second condition above also suggests that the task cannot achieve superlinear speedup, i.e.,

$$\frac{t_j(p)}{t_j(q)} \leq \frac{q}{p} \text{ for all } 1 \leq p < q \leq P \,. \tag{6}$$

Lemma 3. *A task j 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 $p_j \leq p_j^{\max} \leq \bar{p}_j$. 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 integer solution is achieved at $\tilde{p}_j$. Since we also have $p_j \leq p_j^{\max} \leq \tilde{p}_j$, it shows that the execution time is a non-increasing function of p_j in the range $[1, p_j^{\max}]$.

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

Based on Lemmas 2 and 3, the minimum execution time of the task is achieved as $t_j^{\min} = t_j(p_j^{\max})$ and the minimum area of the task is achieved as $a_j^{\min} = a_j(1)$.

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
```

4 ONLINE ALGORITHM AND ITS ANALYSIS

In this section, we present a new online scheduling algorithm and derive its competitive ratio for the general speedup model (Equation (1)) and 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, it checks if new tasks have become available. If so, then 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 [14] 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*** strategy proposed in References [5, 6]. Specifically, for each possible allocation $p \in [1, p_j^{\max}]$, we define the following:

- $g_j(p) \triangleq \frac{a_j(p)}{a_j^{\min}}$: ratio between the area of the task and its minimum area;
- $f_j(p) \triangleq \frac{t_j(p)}{t_j^{\min}}$: ratio between the execution time of the task and its minimum execution time.

We then find an allocation p that minimizes $f_j(p)$ subject to the constraint $g_j(p) \leq \alpha^M$, where $\alpha^M \geq 1$ is a constant whose exact value will be determined based upon the specific speedup model M under consideration. As we will show in our later analysis (Section 4.4), this algorithm can achieve certain *local properties* for any task j under a speedup model M by guaranteeing constant upper bounds for the two ratios (i.e., $g_j(p)$ and $f_j(p)$). Such local properties can then be used globally for proving the competitive ratios of the algorithm. Since $g_j(p)$ is non-decreasing with p and $f_j(p)$ is non-increasing with p, the above optimization problem can be efficiently solved using binary search in $O(\log P)$ time.

In the second step, the algorithm reduces the initial allocation to $\lceil \mu^M P \rceil$ if it is more than $\lceil \mu^M P \rceil$; otherwise, the allocation will be unchanged. Here, $\mu^M \leq 0.5$ is also a constant whose value will be determined by the speedup model M. Let p_j denote the initial allocation for the task and p'_j the

ALGORITHM 2: Allocate_Processor(j)

Input: Task j and its speedup model M; parameters α^M, μ^M
Output: Processor allocation p'_j for the task

// 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 $p_j \in [1, p_j^{\max}]$ from the following optimization problem:

$$\min_p f_j(p) \triangleq \frac{t_j(p)}{t_j^{\min}}$$
$$\text{s.t. } g_j(p) \triangleq \frac{a_j(p)}{a_j^{\min}} \leq \alpha^M$$

// Step 2: Allocation Adjustment

if $p_j > \lceil \mu^M P \rceil$ **then** $p'_j \leftarrow \lceil \mu^M P \rceil$ **else** $p'_j \leftarrow p_j$

final allocation. Thus, after the second step, we have:

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

This step adopts the technique first proposed in Reference [24] and subsequently also used in References [19, 20]. The intuition behind this step is to impose a cap on the number of processors allocated for any task to enable the execution of more tasks at any time, thus potentially increasing the overall resource utilization of the platform and reducing the makespan.

The exact values of the two parameters α^M and μ^M for each speedup model M will be presented in Section 4.4 when analyzing the above algorithm.

4.2 Analysis Overview

Before delving into the details of the analysis for the proposed online algorithm, we first provide an overview in this section that outlines the key ideas of our analysis. On a high level, the analysis consists of two parts: a global analysis framework and a local analysis of the tasks' properties. Combining the two parts leads to the algorithm's competitive ratios.

First, in the global analysis framework (Section 4.3), we focus on the online list scheduling algorithm and divide its entire execution of a task graph into intervals of different categories. We relate subsets of these intervals to the total area A^{opt} and critical path length C^{opt} under an optimal schedule (Lemmas 6 and 7), while assuming certain local properties of the tasks. As A^{opt} and C^{opt} serve as two lower bounds on the optimal makespan (Lemma 1), it allows us to derive an explicit expression for the algorithm's competitive ratio subject to a constraint and the tasks' local properties (Lemma 8). We note that this analysis framework is generically applicable to any monotonic speedup model, and it is inspired by the analysis shown in References [19, 20, 24]. Compared to the original analysis, we refine the intervals to include more categories. This allows for a more coupled analysis with the tasks' local properties, which were not considered in References [19, 20, 24].

Later, in the local analysis (Section 4.4), we focus on the processor allocation algorithm (i.e., Algorithm 2) and show that it achieves certain local properties for any task under a given speedup model M. Specifically, we show that the area and execution time ratios of a task (as defined in Algorithm 2) can be upper-bounded by two constants α^M and β^M regardless of the task's parameters (Lemmas 9 to 12). Then, plugging α^M and β^M into the global analysis result while verifying that the required constraint is satisfied gives rise to the competitive ratio for each considered speedup

model (Theorems 1 to 4). We point out that, compared to the analysis in our preliminary work [7], the local bounds α^M and β^M derived in this article are tighter, which along with the refined global analysis leads to the improved competitive ratios.

4.3 Global Analysis Framework

In this section, we present the global analysis framework that derives the competitive ratio of our online algorithm. Since the framework applies to any speedup model M, for simplicity, we will drop the superscript M for α^M, β^M, and μ^M, and re-introduce it later in Section 4.4 when deriving the local properties for each considered speedup model.

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. We first classify the set of all intervals into the following two categories:

— $\mathcal{I}_0$: subset of intervals that satisfy $p(I) \in (0, \lceil(1-\mu)P\rceil)$;
— $\mathcal{I}_3$: subset of intervals that satisfy $p(I) \in [\lceil(1-\mu)P\rceil, P]$.

We will use the following well-known result from References [13, 14] regarding a property of list scheduling.

Lemma 4 ([13, 14]). *When a task graph is scheduled by list scheduling, there exists a path f in the graph such that some task along that path is running whenever there is no available task in the queue.*

We now apply the above lemma to show a property for the subset of intervals in $\mathcal{I}_0$.

Lemma 5. *There exists a path f in the graph such that in any interval of $\mathcal{I}_0$ some task along f is running.*

Proof. During $\mathcal{I}_0$, 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 during $\mathcal{I}_0$. The result directly follows by applying Lemma 4. □

Using the path f stated in Lemma 5, we further split $\mathcal{I}_0$ into the following two sub-categories:

— $\mathcal{I}_1$: subset of $\mathcal{I}_0$ where the processor allocation for the currently running task in f was not reduced (by the second step of Algorithm 2);
— $\mathcal{I}_2$: subset of $\mathcal{I}_0$ where the processor allocation for the currently running task in f was reduced (i.e., the task is running on $\lceil\mu P\rceil$ processors).

Finally, given the processor allocation of an optimal schedule, we further split $\mathcal{I}_2$ into the following two sub-categories:

— $\mathcal{I}_{2'}$: subset of $\mathcal{I}_2$ where the currently running task in f was allocated with strictly fewer processors than in the optimal schedule;
— $\mathcal{I}_{2''}$: subset of $\mathcal{I}_2$ where the currently running task in f was allocated with equal or more processors than in the optimal schedule.

Let $|I|$ denote the duration of an interval I, and let $T_1 = \sum_{I\in\mathcal{I}_1} |I|$, $T_2 = \sum_{I\in\mathcal{I}_2} |I|$, $T_{2'} = \sum_{I\in\mathcal{I}_{2'}} |I|$, $T_{2''} = \sum_{I\in\mathcal{I}_{2''}} |I|$, and $T_3 = \sum_{I\in\mathcal{I}_3} |I|$ denote the total durations of the different categories of intervals,

respectively. Since $\mathcal{I}_1$, $\mathcal{I}_2$, and $\mathcal{I}_3$ are obviously disjoint and partition $\mathcal{I}$, we have $T = T_1 + T_2 + T_3$. Finally, we define $z \in [0, 1]$ such that $T_{2'} = zT_2$ and $T_{2''} = (1 - z)T_2$.

The next two lemmas relate these durations to the total area A^{opt} and critical path length C^{opt} of the task graph under an optimal schedule, given certain conditions on the initial processor allocations of the tasks under our algorithm.

Lemma 6. *If there exists a constant α such that, for each task j, its initial processor allocation satisfies $a_j(p_j) \le \alpha a_j^{\min}$, then we have:*

$$\mu \left(z + \frac{1 - z}{\alpha} \right) T_2 + \frac{(1 - \mu)}{\alpha} T_3 \le T^{opt} . \tag{8}$$

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 $a_j(p'_j) \le a_j(p_j) \le \alpha a_j^{\min} \le \alpha a_j^{\text{opt}}$. Furthermore, during $\mathcal{I}_{2'}$, any running task j from path f satisfies $a_j(p'_j) \le a_j(p_j^{\text{opt}}) = a_j^{\text{opt}}$. We let $A_{2'|f}$ (respectively, $A_{2''|f}$) denote the total area of the fraction of tasks from f running in $\mathcal{I}_{2'}$ (respectively, $\mathcal{I}_{2''}$), and $A^{\text{opt}}_{2'|f}$ (respectively, $A^{\text{opt}}_{2''|f}$) the corresponding fraction of area in an optimal schedule. We have $A_{2'|f} \le A^{\text{opt}}_{2'|f}$ and $A_{2''|f} \le \alpha A^{\text{opt}}_{2''|f}$. Since $\lceil \mu P \rceil \ge \mu P$ processors are used to run tasks from f in $\mathcal{I}_{2'} \cup \mathcal{I}_{2''}$, we have $\mu T_{2'} \le \frac{A_{2'|f}}{P} \le \frac{A^{\text{opt}}_{2'|f}}{P}$ and $\mu T_{2''} \le \frac{A_{2''|f}}{P} \le \frac{\alpha A^{\text{opt}}_{2''|f}}{P}$.

Finally, let A_3 denote the total area of the fraction of tasks running in $\mathcal{I}_3$ and A_3^{opt} the corresponding fraction of area in an optimal schedule. Since at least $\lceil (1 - \mu)P \rceil \ge (1 - \mu)P$ processors are utilized during $\mathcal{I}_3$, we have $(1 - \mu)T_3 \le \frac{A_3}{P} \le \frac{\alpha A_3^{\text{opt}}}{P}$.

Thus, altogether, we can derive:

$$\begin{aligned}
&\mu \left(z + \frac{1 - z}{\alpha} \right) T_2 + \frac{(1 - \mu)}{\alpha} T_3 \\
&= \mu T_{2'} + \frac{\mu T_{2''}}{\alpha} + \frac{(1 - \mu)}{\alpha} T_3 \\
&\le \frac{A^{\text{opt}}_{2'|f}}{P} + \frac{A^{\text{opt}}_{2''|f}}{P} + \frac{A_3^{\text{opt}}}{P} \\
&\le \frac{A^{\text{opt}}}{P} \le T^{\text{opt}} .
\end{aligned}$$

The last inequality is due to the makespan lower bound shown in Lemma 1. □

Lemma 7. *If there exists a constant β such that, for each task j, its initial processor allocation satisfies $t_j(p_j) \le \beta t_j^{\min}$, then we have:*

$$\frac{T_1}{\beta} + (\mu z + 1 - z) T_2 \le T^{opt} . \tag{9}$$

Proof. For any task j from path f running during $\mathcal{I}_1$, its processor allocation was not reduced by the second step of Algorithm 2, thus, we must have $p'_j = p_j \le \lceil \mu P \rceil$. Therefore, its execution time should satisfy $t_j(p'_j) = t_j(p_j) \le \beta t_j^{\min} \le \beta t_j^{\text{opt}}$.

For any task j from path f running during $\mathcal{I}_{2'}$, its processor allocation has been reduced, i.e., $p'_j = \lceil \mu P \rceil$. Based on Equation (6), the task's execution time should satisfy $\frac{t_j(p'_j)}{t_j^{\min}} = \frac{t_j(\lceil \mu P \rceil)}{t_j(p_j^{\max})} \le \frac{p_j^{\max}}{\lceil \mu P \rceil} \le \frac{P}{\mu P} = \frac{1}{\mu}$. Thus, we have $t_j(p'_j) \le \frac{1}{\mu} t_j^{\min} \le \frac{1}{\mu} t_j^{\text{opt}}$.

Finally, for any task j from path f running during $\mathcal{I}_{2''}$, its processor allocation is higher than that of an optimal schedule. Therefore, its execution time should satisfy: $t_j(p'_j) \leq t_j(p_j^{\text{opt}}) = t_j^{\text{opt}}$.

Now, let $L_{1|f}^{\text{opt}}$ (respectively, $L_{2'|f}^{\text{opt}}$ and $L_{2''|f}^{\text{opt}}$) denote the length for the portion of path f executed during $\mathcal{I}_1$ (respectively, $\mathcal{I}_{2'}$ and $\mathcal{I}_{2''}$) under an optimal schedule. The argument above implies that $T_1 \leq \beta L_{1|f}^{\text{opt}}$, $T_{2'} \leq \frac{1}{\mu} L_{2'|f}^{\text{opt}}$ and $T_{2''} \leq L_{2''|f}^{\text{opt}}$. Thus, we can derive:

$$\begin{aligned}
&\frac{T_1}{\beta} + (\mu z + 1 - z) T_2 \\
&= \frac{T_1}{\beta} + \mu T_{2'} + T_{2''} \\
&\leq L_{1|f}^{\text{opt}} + L_{2'|f}^{\text{opt}} + L_{2''|f}^{\text{opt}} \\
&\leq L^{\text{opt}}(f) \leq C^{\text{opt}} \leq T^{\text{opt}} .
\end{aligned}$$

The last inequality is again due to the makespan lower bound shown in Lemma 1. □

Based on the results of Lemmas 6 and 7, we can now derive an upper bound on the makespan of the online scheduling algorithm as shown below.

Lemma 8. *If there exist two constants α and β such that, for each task j, then its initial processor allocation satisfies:*

$$g_j(p_j) \triangleq \frac{a_j(p_j)}{a_j^{\min}} \leq \alpha , \tag{10}$$

$$f_j(p_j) \triangleq \frac{t_j(p_j)}{t_j^{\min}} \leq \beta , \tag{11}$$

then by setting μ such that $\beta + \frac{\alpha}{1-\mu} = \frac{1}{\mu}$, i.e., $\mu = \frac{\alpha+\beta+1-\sqrt{(\alpha+\beta+1)^2-4\beta}}{2\beta}$, and under the condition $\beta \geq \frac{\mu(\alpha-1)}{(1-\mu)^2}$, we get:

$$\frac{T}{T^{opt}} \leq \frac{1}{\mu} = \frac{2\beta}{\alpha + \beta + 1 - \sqrt{(\alpha+\beta+1)^2 - 4\beta}} . \tag{12}$$

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 Inequality (8) to remove the T_3 term, which gives:

$$\frac{1-\mu}{\alpha} T \leq \frac{1-\mu}{\alpha} T_1 + \left(\frac{1 - \mu - z\alpha\mu - (1-z)\mu}{\alpha} \right) T_2 + T^{\text{opt}} .$$

We can then multiply both sides of the above inequality by $\frac{\alpha}{(1-\mu)\beta}$ and apply Inequality (9) to remove the T_1 term. This gives:

$$\frac{T}{\beta} \leq \left(\frac{1 - \mu - z\alpha\mu - (1-z)\mu}{(1-\mu)\beta} - \mu z + z - 1 \right) T_2 + \left(1 + \frac{\alpha}{(1-\mu)\beta} \right) T^{\text{opt}} .$$

Now, if $f(z) = \frac{1-\mu-z\alpha\mu-(1-z)\mu}{(1-\mu)\beta} - \mu z + z - 1 \leq 0$ is true for all $z \in [0, 1]$, then we can omit the T_2 term in the above inequality and get:

$$T \leq \left(\beta + \frac{\alpha}{(1-\mu)} \right) T^{\text{opt}} = \frac{1}{\mu} T^{\text{opt}} .$$

We have $f'(z) = \frac{\mu(1-\alpha)}{(1-\mu)\beta} + (1-\mu) \geq 0$ under the condition $\beta \geq \frac{\mu(\alpha-1)}{(1-\mu)^2}$, which makes $f(z)$ an increasing function of z. Thus, we simply need to ensure that $f(1) = \frac{1-\mu-\alpha\mu}{(1-\mu)\beta} - \mu \leq 0$, which is true if $\beta + \frac{\alpha}{1-\mu} = \frac{1}{\mu}$. One can then solve for μ from the above second-degree equation, and get $\mu = \frac{\alpha+\beta+1-\sqrt{(\alpha+\beta+1)^2-4\beta}}{2\beta}$.

Finally, we show that the value of μ above is well-defined and is a valid choice satisfying $\mu \in (0, 0.5]$.

- First, we can derive that $\Delta = (\alpha+\beta+1)^2 - 4\beta > (\beta+1)^2 - 4\beta = (\beta-1)^2 \geq 0$. Thus, the value of μ is well-defined.
- We have $\mu > \frac{\alpha+\beta+1-\sqrt{(\alpha+\beta+1)^2}}{2\beta} = 0$, since $\alpha, \beta > 0$.
- We can show $\mu = \frac{\alpha+\beta+1-\sqrt{(\alpha+\beta+1)^2-4\beta}}{2\beta} \leq 0.5$, which after some manipulations is equivalent to showing $0 \leq \beta^2 + 2\beta(\alpha-1)$. The latter inequality is always true, since $\alpha \geq 1$. □

Remarks. We point out that the two bounds shown in Lemma 8, i.e., Inequalities (10) and (11), must be satisfied by all tasks in a task graph for the derived competitive ratio shown in Equation (12) to hold. In Section 4.4, we will prove that, for a given speedup model and regardless of the task parameters, there always exists a processor allocation that satisfies the two bounds with a particular (α, β) choice that achieves the minimum (or close to minimum) competitive ratio. We then show in Section 5 that the obtained competitive ratios are tight (or almost tight) by proving matching (or near-matching) lower bounds for different speedup models. Thus, given an achievable (α, β) pair for a speedup model, the processor allocation algorithm should always be able to find an allocation for each task to satisfy the two bounds. In case of multiple allocations that all satisfy the bounds, Algorithm 2 will find one that, subject to the α bound, minimizes the execution time of the task, thus satisfying the β bound as well. Intuitively, this is a good practical choice and it also helps to simplify the analysis, which we will present in Section 4.4.

4.4 Local Analysis and Competitive Ratios

In this section, we analyze the local properties of a task and prove the competitive ratios of the online algorithm under different speedup models. Based on Lemma 8, the competitive ratio is given by: $\frac{1}{\mu} = \frac{2\beta}{\alpha+\beta+1-\sqrt{(\alpha+\beta+1)^2-4\beta}}$ subject to the constraint $\beta \geq \frac{\alpha-1}{(1-\mu)^2}$.

For a given speedup model with parameters $\mathcal{P} \subseteq \{w, d, c, \bar{p}\}$, a generic approach for minimizing the ratio above can be outlined as follows: First, compute $\beta(\alpha)$ as small as possible based on the model parameters $\mathcal{P}$ for any fixed $\alpha \geq 1$. To do that, since the area of a task is non-decreasing with the processor allocation and the time non-increasing in the range $[1, p^{\max}]$, we can find the largest processor allocation $p^*(\mathcal{P}) \in [1, p^{\max}]$ that satisfies $\frac{a(p^*(\mathcal{P}))}{a^{\min}} \leq \alpha$ and then compute $\beta(\alpha) = sup_{\mathcal{P}}(\frac{t(p^*(\mathcal{P}))}{t^{\min}})$. Finally, plug $\beta(\alpha)$ into the expression of the competitive ratio and find the α that minimizes it while satisfying the constraint.

Although the technique outlined above is a good generic approach, the computations involved are often too complicated for some speedup models and solving it will rely on numerical tools. Therefore, to derive the competitive ratios analytically, we will simply find a valid pair (α, β) below for each considered speedup model while verifying that the constraint is satisfied. We will then show the tightness (or near tightness) of the obtained competitive ratios by computing the lower bounds in Section 5.

In the following, we will first consider the three special speedup models (i.e., roofline, communication, and Amdahl) before tackling the general model. For clarity, we now re-introduce superscript $M \in \{\text{Roo}, \text{Com}, \text{Amd}, \text{Gen}\}$ to the notations α^M, β^M, and μ^M in the lemmas and theorems below. Given a speedup model M, the analysis focuses on finding α^M and β^M for each individual task, thus, we will drop the task index j for simplicity.

4.4.1 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 9. *For any task that follows the roofline speedup model, there exists a processor allocation that achieves $\alpha^{Roo} = \beta^{Roo} = 1$.*

Proof. For any task with $\bar{p}$, setting the processor allocation to $p = \bar{p}$ 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 $\alpha^{\text{Roo}} = \beta^{\text{Roo}} = 1$. □

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

Proof. With $\alpha = \beta = 1$, the constraint $\beta \geq \frac{\mu(\alpha-1)}{(1-\mu)^2} = 0$ is obviously satisfied. Thus, we get $\mu = \frac{\alpha+\beta+1-\sqrt{(\alpha+\beta+1)^2-4\beta}}{2\beta} = \frac{3-\sqrt{5}}{2}$, and the competitive ratio is given by $\frac{1}{\mu} = \frac{2}{3-\sqrt{5}} < 2.62$. □

Remarks. The above ratio retains the same result by Feldmann et al. [13],[2] who also proved a matching lower bound for any online deterministic algorithm under the "non-clairvoyant" setting, where the work w of a task is unknown to the scheduler. In Section 5, we will prove the same lower bound, but without the non-clairvoyant setting for a class of list scheduling algorithms with deterministic local decisions for processor allocation.

4.4.2 Communication Model. Recall that a task follows the communication model if its execution time satisfies $t(p) = \frac{w}{p} + c(p-1)$, with $c \geq 0$. If $c = 0$, then it simplifies to a special case of the roofline model and we can reach $\alpha = \beta = 1$. Thus, we assume $c > 0$ and rewrite the execution time function as: $t(p) = c(\frac{w'}{p} + p - 1)$ with $w' = \frac{w}{c}$. The area function is then given by $a(p) = c(w' + p(p-1))$.

Lemma 10. *For any task that follows the communication model, there exists a processor allocation that achieves $\alpha^{Com} = \frac{4}{3}$ and $\beta^{Com} = \frac{3}{2}$.*

Proof. Recall that $p^{\max}$ denotes the number of processors that minimizes the task's execution time $t(p)$, 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 minimum area of the task is obtained with one processor, i.e., $a^{\min} = a(1) = cw'$.

Furthermore, for a given choice of p, we can show that $f(w', p) \triangleq \frac{t(p)}{t^{\min}} = \frac{\frac{w'}{p}+p-1}{\frac{w'}{p^{\max}}+p^{\max}-1}$ is a non-decreasing function of w' in the interval $[p^2, \infty)$. To see that, we can compute the partial

[2]In Reference [13], 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.

derivative:

$$\frac{\partial f(w',p)}{\partial w'} = \frac{\frac{1}{p}\left(\frac{w'}{p^{\max}} + p^{\max} - 1\right) - \frac{1}{p^{\max}}\left(\frac{w'}{p} + p - 1\right)}{\left(\frac{w'}{p^{\max}} + p^{\max} - 1\right)^2} = \frac{\frac{p^{\max}-1}{p} - \frac{p-1}{p^{\max}}}{\left(\frac{w'}{p^{\max}} + p^{\max} - 1\right)^2},$$

which is defined everywhere except at the points where $p^{\max}$ changes (due to changes of w'), and has the same sign as $\frac{p^{\max}-1}{p} - \frac{p-1}{p^{\max}} \geq 0$. The last inequality is because if $p = p^{\max}$, then it is equal to 0, otherwise, $\frac{p^{\max}-1}{p} \geq 1$ and $\frac{p-1}{p^{\max}} < 1$. This remains true if $p \leq p^{\max}$, which is satisfied when $w' \geq p^2$.

We now consider three cases:

Case 1: $w' \leq 6$. In this case, we set $p = 1$, which gives the minimum area, i.e., $\frac{a(p)}{a^{\min}} = 1$. When $w' \leq 1$, setting $p = 1$ also gives the minimum execution time, i.e., $\frac{t(p)}{t^{\min}} = 1$. Otherwise, we have $\frac{t(p)}{t^{\min}} = f(w',1) \leq f(6,1) = \frac{6}{\min(\frac{6}{2}+1, \frac{6}{3}+2)} = \frac{3}{2}$, since $p^{\max} = 2$ or $p^{\max} = 3$ when $w' = 6$.

Case 2: $6 < w' \leq 25$. In this case, we set $p = 2$ and get $\frac{a(p)}{a^{\min}} = \frac{c(w'+2)}{cw'} = 1 + \frac{2}{w'} < \frac{4}{3}$. As $w' > p^2 = 4$, we can also get $\frac{t(p)}{t^{\min}} = f(w',2) \leq f(25,2) = \frac{\frac{25}{2}+1}{\frac{25}{5}+4} = \frac{27}{18} = \frac{3}{2}$, since $p^{\max} = 5$ when $w' = 25$.

Case 3: $w' > 25$. In this case, we use $t^{\min} \geq c(2\sqrt{w'} - 1)$, which is the minimum possible execution time if the processor allocation could be non-integers. We set $p = \lfloor \sqrt{\frac{w'}{3}} + \frac{1}{2} \rfloor$ and obtain $\frac{a(p)}{a^{\min}} = \frac{c(w'+p(p-1))}{cw'} \leq 1 + \frac{1}{w'}(\sqrt{\frac{w'}{3}} + \frac{1}{2})(\sqrt{\frac{w'}{3}} - \frac{1}{2}) \leq 1 + \frac{1}{w'}\frac{w'}{3} = \frac{4}{3}$. Finally, $\frac{t(p)}{t^{\min}} \leq \frac{c(\frac{w'}{\sqrt{\frac{w'}{3}}-\frac{1}{2}} + \sqrt{\frac{w'}{3}})}{c(2\sqrt{w'}-1)} = \frac{1}{2-\frac{1}{\sqrt{w'}}}(\frac{1}{\frac{1}{\sqrt{3}}-\frac{1}{2\sqrt{w'}}} + \frac{1}{\sqrt{3}})$. This function is clearly decreasing with w', and using $w' > 25$, we get $\frac{t(p)}{t^{\min}} \leq \frac{5}{9}(\frac{10\sqrt{3}}{10-\sqrt{3}} + \frac{1}{\sqrt{3}}) \approx 1.48 < \frac{3}{2}$. □

Remarks. Using the generic approach outlined at the beginning of this section, we could actually compute the best possible $\beta(\alpha)$ for any $\alpha > 1$. The analysis, however, is very technical, and finding the optimal α then requires numerical analysis tools. It turns out that the best (α, β) pair is $(\frac{4}{3}, \frac{3}{2})$, and we will show that this pair is optimal in Section 5.

Theorem 2. *Algorithm 1 is* $\frac{18}{23-\sqrt{313}} < 3.391$*-competitive for any graph of tasks that follow the communication model. This is achieved with* $\mu^{\text{Com}} = \frac{23-\sqrt{313}}{18} \approx 0.295$.

Proof. With $\alpha = \frac{4}{3}$ and $\beta = \frac{3}{2}$, we get $\mu = \frac{\alpha+\beta+1-\sqrt{(\alpha+\beta+1)^2-4\beta}}{2\beta} = \frac{23-\sqrt{313}}{18}$, and the constraint $\beta \geq \frac{\mu(\alpha-1)}{(1-\mu)^2} \approx 0.2$ is satisfied. Thus, the competitive ratio is given by $\frac{1}{\mu} = \frac{18}{23-\sqrt{313}} < 3.391$. □

4.4.3 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 $d \geq 0$, thus the area function is given by $a(p) = pt(p) = w + dp$.

Lemma 11. *For any task that follows the Amdahl's model, there exists a processor allocation that achieves* $\alpha^{\text{Amd}} = \frac{\sqrt{2}+1+\sqrt{2\sqrt{2}-1}}{2} \approx 1.883$ *and* $\beta^{\text{Amd}} = \frac{1+\sqrt{4\sqrt{2}+5}}{2} \approx 2.132$.

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

We first show that, for any $\alpha > 1$, there exists a processor allocation p that satisfies the α bound, i.e., $\frac{a(p)}{a^{\min}} \leq \alpha$ and at the same time achieves $\beta(\alpha) = \frac{\alpha}{\alpha-1}$, i.e., $\frac{t(p)}{t^{\min}} \leq \beta(\alpha) = \frac{\alpha}{\alpha-1}$. To show that, for any given $\alpha > 1$, we let $x = \alpha - 1$, and 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 $\frac{a(p)}{a^{\min}} = \frac{w+dp}{w+d} \leq \frac{w+d\left(x\frac{w}{d}+1\right)}{w+d} = \frac{w+d+xw}{w+d} = 1 + \frac{xw}{w+d} \leq 1 + x = \alpha$. Furthermore, if $p = \lceil x\frac{w}{d} \rceil \geq x\frac{w}{d}$, then we have $\frac{t(p)}{t^{\min}} \leq \frac{\frac{w}{x\frac{w}{d}}+d}{\frac{w}{P}+d} \leq \frac{\frac{d}{x}+d}{d} = \frac{1}{x} + 1 = \frac{1}{\alpha-1} + 1 = \frac{\alpha}{\alpha-1} = \beta(\alpha)$. Otherwise, if $p = P$, then we get $t(p) = t^{\min}$ and thus $\frac{t(p)}{t^{\min}} = 1 < \frac{\alpha}{\alpha-1} = \beta(\alpha)$.

We can now substitute $\beta(\alpha)$ into the expression of the competitive ratio and get:

$$\frac{1}{\mu} = \frac{2\frac{\alpha}{\alpha-1}}{\alpha + \frac{\alpha}{\alpha-1} + 1 - \sqrt{\left(\alpha + \frac{\alpha}{\alpha-1} + 1\right)^2 - 4\frac{\alpha}{\alpha-1}}}.$$

To minimize the ratio above, one can use the standard technique of differentiating and setting the derivative to zero. The expression is quite long, and the full analysis is omitted. It turns out that $\alpha^{\text{Amd}} = \frac{\sqrt{2}+1+\sqrt{2\sqrt{2}-1}}{2}$ minimizes the ratio. Plugging it back into $\beta(\alpha) = \frac{\alpha}{\alpha-1}$ and simplifying, we can get $\beta^{\text{Amd}} = \frac{1+\sqrt{4\sqrt{2}+5}}{2}$. □

Theorem 3. *Algorithm 1 is* $\frac{2}{1-\sqrt{8\sqrt{2}-11}} < 4.55$*-competitive for any graph of tasks that follow the Amdahl's model. This is achieved with* $\mu^{\text{Amd}} = \frac{1-\sqrt{8\sqrt{2}-11}}{2} \approx 0.22$.

Proof. By substituting $\alpha = \frac{\sqrt{2}+1+\sqrt{2\sqrt{2}-1}}{2}$ and $\beta = \frac{1+\sqrt{4\sqrt{2}+5}}{2}$ into the expression of μ and simpifying (hard!), we can get $\mu = \frac{\alpha+\beta+1-\sqrt{(\alpha+\beta+1)^2-4\beta}}{2\beta} = \frac{1-\sqrt{8\sqrt{2}-11}}{2}$. We can also check that the constraint $\beta > \frac{\mu(\alpha-1)}{(1-\mu)^2} \approx 0.32$ is satisfied. Thus, the competitive ratio is given by $\frac{1}{\mu} = \frac{2}{1-\sqrt{8\sqrt{2}-11}} < 4.55$. □

4.4.4 General Model. We finally consider the general speedup model as given in Equation (1). Without loss of generality, we assume $w > 0$ (otherwise, we get $\alpha = \beta = 1$ using one processor), and $c, d > 0$ (otherwise, the model reduces to the communication or the Amdahl's model and also results in smaller α and β). 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 $d' = \frac{d}{c}$. We further assume $\bar{p} \leq P$ (otherwise, changing $\bar{p}$ to P does not affect the execution time of the task for any feasible processor allocation). Finally, any reasonable scheduling algorithm will not allocate more than $\bar{p}$ processors to the task, since it would increase both execution time and area. Thus, assuming $p \leq \bar{p}$, we can simplify the execution time function as $t(p) = c(\frac{w'}{p} + d' + p - 1)$ and the area function is given by $a(p) = c(w' + d'p + p(p-1))$.

Lemma 12. *For any task that follows the general model, there exists a processor allocation that achieves* $\alpha^{\text{Gen}} = 2$ *and* $\beta^{\text{Gen}} = \frac{27}{13}$.

Proof. If we allow the processor allocation to take non-integer values and assuming unbounded $\bar{p}$, then 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 $p^{\max} = \bar{p}$ or $\lfloor \sqrt{w'} \rfloor \le p^{\max} \le \lceil \sqrt{w'} \rceil$.

We consider three cases.

Case 1: $w' \le 4$ or $\bar{p} = 1$. In this case, it must be that $p^{\max} \le 2$. We can then set $p = 1$, and get $\frac{a(p)}{a^{\min}} = 1$ and $\frac{t(p)}{t^{\min}} \le 2$.

Case 2: $4 < w' \le 49$ and $\bar{p} \ge 2$. In this case, we set $p = 2$ and get $\frac{a(p)}{a^{\min}} \le \frac{w'+2d'+2}{w'+d'} \le 2$. Similarly to the proof of Lemma 10 (for the communication model), we can show that $f(w', p) \triangleq \frac{t(p)}{t^{\min}} = \frac{\frac{w'}{p}+d'+p-1}{\frac{w'}{p^{\max}}+d'+p^{\max}-1}$ is increasing with w' if $w' \ge p^2$. Therefore, we can get $\frac{t(p)}{t^{\min}} \le f(49, 2) \le \frac{\frac{49}{2}+d'+1}{2\sqrt{49}+d'-1} = \frac{51+2d'}{26+2d'} \le 2$.

Case 3: $w' > 49$ and $\bar{p} \ge 2$. In this case, we will set $p = \min(\lfloor \frac{w'+d'}{\sqrt{w'}+d'} + \frac{1}{2} \rfloor, \bar{p})$ and get:

$$\begin{aligned}
\frac{a(p)}{a^{\min}} &= \frac{w' + p(d' + p - 1)}{w' + d'} \\
&\le \frac{w' + \left(\frac{w'+d'}{\sqrt{w'}+d'} + \frac{1}{2}\right)\left(d' + \frac{w'+d'}{\sqrt{w'}+d'} - \frac{1}{2}\right)}{w' + d'} \\
&= \frac{w' + \frac{d'}{2} - \frac{1}{4} + \frac{w'+d'}{\sqrt{w'}+d'}\left(d' + \frac{w'+d'}{\sqrt{w'}+d'}\right)}{w' + d'} \\
&\le \frac{w' + d' + \frac{w'+d'}{\sqrt{w'}+d'}\left(d' + \frac{w'+d'}{\sqrt{w'}+d'}\right)}{w' + d'} \\
&= 1 + \frac{d'(\sqrt{w'} + d') + w' + d'}{(\sqrt{w'} + d')^2} \\
&= 1 + \frac{d'^2 + d'\sqrt{w'} + d' + w'}{d'^2 + 2d'\sqrt{w'} + w'} \\
&\le 2 .
\end{aligned}$$

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

Since $w' > 1$, we get $t^{\min} \ge c(2\sqrt{w'} + d' - 1) > c(\sqrt{w'} + d')$. To derive the execution time ratio, we further consider two subcases.

— If $p = \lfloor \frac{w'+d'}{\sqrt{w'}+d'} + \frac{1}{2} \rfloor$, then $p \ge \frac{w'+d'}{\sqrt{w'}+d'} - \frac{1}{2} \ge \frac{w'-\frac{1}{2}\sqrt{w'}}{\sqrt{w'}+d'}$. We can then get:

$$\begin{aligned}
\frac{t(p)}{t^{\min}} &\le \frac{\frac{w'}{p} + d' + p - 1}{\sqrt{w'} + d'} \\
&\le \frac{\frac{w'(\sqrt{w'}+d')}{w'-\frac{1}{2}\sqrt{w'}}}{\sqrt{w'} + d'} + \frac{d' + \frac{w'+d'}{\sqrt{w'}+d'}}{\sqrt{w'} + d'} \\
&\le \frac{1}{1 - \frac{1}{2\sqrt{w'}}} + \frac{d'(\sqrt{w'} + d') + w' + d'}{(\sqrt{w'} + d')^2} \\
&\le \frac{1}{1 - \frac{1}{2\sqrt{w'}}} + 1.
\end{aligned}$$

Table 3. Summary of Parameters and Competitive Ratios for Different Speedup Models

Model M	μ^M	α^M	β^M	**Comp. Ratio**
Roofline (Roo)	$\frac{3-\sqrt{5}}{2} \approx 0.382$	1	1	$\frac{2}{3-\sqrt{5}} \approx 2.62$
Comm. (Com)	$\frac{23-\sqrt{313}}{18} \approx 0.295$	$\frac{4}{3}$	$\frac{3}{2}$	$\frac{18}{23-\sqrt{313}} \approx 3.39$
Amdahl (Amd)	$\frac{1-\sqrt{8\sqrt{2}-11}}{2} \approx 0.22$	$\frac{\sqrt{2}+1+\sqrt{2\sqrt{2}-1}}{2} \approx 1.88$	$\frac{1+\sqrt{4\sqrt{2}+5}}{2} \approx 2.13$	$\frac{2}{1-\sqrt{8\sqrt{2}-11}} \approx 4.55$
General (Gen)	$\frac{33-\sqrt{738}}{27} \approx 0.216$	2	$\frac{27}{13}$	$\frac{27}{33-\sqrt{738}} \approx 4.63$

For the last inequality, we recognize the same term we had when bounding the area ratio, which is at most 1. Finally, the last expression above decreases with w', so using $w' > 49$, we get $\frac{t(p)}{t^{\min}} \leq \frac{1}{1-\frac{1}{14}} + 1 = \frac{27}{13}$.

— If $p = \bar{p} < \lfloor \frac{w'+d'}{\sqrt{w'}+d'} + \frac{1}{2} \rfloor$, and, since $\bar{p}$ is an integer, then it is necessarily the case that $\bar{p} \leq \lfloor \frac{w'+d'}{\sqrt{w'}+d'} + \frac{1}{2} \rfloor - 1 \leq \frac{w'+d'}{\sqrt{w'}+d'} \leq \sqrt{w'}$ (because $w' > 1$). Therefore, we should also have $p^{\max} = \bar{p} = p$, and thus $\frac{t(p)}{t^{\min}} = 1$. □

Theorem 4. *Algorithm 1 is* $\frac{27}{33-\sqrt{738}} < 4.63$*-competitive for any graph of tasks that follow the general speedup model given in Equation* (1)*. This is achieved with* $\mu^{Gen} = \frac{33-\sqrt{738}}{27} \approx 0.216$.

Proof. With $\alpha = 2$ and $\beta = \frac{27}{13}$, we get $\mu = \frac{\alpha+\beta+1-\sqrt{(\alpha+\beta+1)^2-4\beta}}{2\beta} = \frac{33-\sqrt{738}}{27}$ and the constraint $\beta \geq \frac{\mu(\alpha-1)}{(1-\mu)^2} \approx 0.35$ is satisfied. Thus, the competitive ratio is given by $\frac{1}{\mu} = \frac{27}{33-\sqrt{738}} < 4.63$. □

Finally, Table 3 summarizes the parameters and competitive ratios derived for all the considered speedup models.

5 LOWER BOUNDS FOR ONLINE LIST SCHEDULING ALGORITHMS WITH DETERMINISTIC LOCAL DECISIONS

In Section 4.4, we derived the competitive ratios of our online algorithm under several common speedup models. In this section, we will show corresponding lower bounds on the competitive ratios. We point out that, in contrast to the lower bounds proven in our preliminary work [7], which apply only to the presented algorithm, the lower bounds shown in this section are stronger, as they apply to *any* online list scheduling algorithm with deterministic local decisions for processor allocation.

Definition 2. An online algorithm is said to make **deterministic local decisions** if it allocates processors by considering only the total number of processors (i.e., P) and the parameters of a task's speedup function (i.e., w, $\bar{p}$, d, c). Thus, two identical tasks will receive exactly the same allocation regardless of their relative positions in the task graph as well as the graph structure.

Ultimately, we will show that our algorithm has the optimal competitive ratios over all algorithms in this class for the roofline, communication, and Amdahl's models. The result also indicates that our algorithms's competitive ratio for the general model is close to optimal using the lower bound of the Amdahl's model.

5.1 Analysis Overview

Under any model M, we have shown in Section 4 that, for any task, our online algorithm achieves:

$$\frac{a}{a^{\min}} \leq \alpha^M \text{ and } \frac{t}{t^{\min}} \leq \beta^M \,, \tag{13}$$

where $\frac{\alpha^M}{1-\mu^M} + \beta^M = \frac{1}{\mu^M}$. In particular, for any possible instance consisting of a set $\mathcal{T}$ of tasks, if our algorithm achieves a makespan of T and the optimal makespan is T^{opt}, then we have shown that:

$$\frac{T}{T^{\text{opt}}} \leq \max_{j\in\mathcal{T}} \frac{a_j}{a_j^{\min}(1-\mu^M)} + \max_{j\in\mathcal{T}} \frac{t_j}{t_j^{\min}} \leq \frac{\alpha^M}{1-\mu^M} + \beta^M = \frac{1}{\mu^M} \,. \tag{14}$$

To prove the lower bounds, we will use a *contradiction* argument. Suppose an online list scheduling algorithm $\mathcal{A}$ respecting Definition 2 and having a competitive ratio strictly less than $\frac{1}{\mu^M}$ exists. Specifically, we will assume that $\mathcal{A}$'s competitive ratio is $\frac{1}{\mu^M} - 4\epsilon$ for some $0 < \epsilon < 1$. We will then derive a contradiction thus proving the result.

Notations. We will use four different tasks A, B, C, D to construct the lower bound instances. For algorithm $\mathcal{A}$, we let p_A (respectively, p_B, p_C, p_D) denote its processor allocation for task A (respectively, B, C, D), let t_A (respectively, t_B, t_C, t_D) denote the resulting execution time of the tasks, and let $a_A = t_A p_A$ (respectively, a_B, a_C, a_D) denote the resulting area of the tasks. Similarly, we use $p_A^*, p_B^*, p_C^*, p_D^*, t_A^*, t_B^*, t_C^*, t_D^*, a_A^*, a_B^*, a_C^*, a_D^*$ to denote the corresponding values for another algorithm $\mathcal{A}^*$ on this instance.

Two-step Approach. We will proceed in two steps corresponding to proving the tightness of the two inequalities (13, local) and (14, global), respectively.

In the first step (Section 5.2), we will show the existence of two tasks A and B and an algorithm $\mathcal{A}^*$ such that:

$$\frac{a_B}{a_B^*(1-\mu^M)} + \frac{t_A}{t_A^*} \geq \frac{1}{\mu^M} - \epsilon \,, \tag{15}$$

thus showing the tightness of our local analysis.

In the second step (Section 5.3), we will build an instance using these two tasks (as well as two other tasks C and D) such that, by using list scheduling, and with best decision-making for $\mathcal{A}^*$, algorithm $\mathcal{A}$ will always have a makespan that satisfies:

$$\frac{T}{T^*} \geq \frac{a_B}{a_B^*(1-\mu^M)} + \frac{t_A}{t_A^*} - 2\epsilon \geq \frac{1}{\mu^M} - 3\epsilon \,, \tag{16}$$

where T and T^* denote the makespans of $\mathcal{A}$ and $\mathcal{A}^*$ for this instance, respectively. This contradicts the assumed competitive ratio of $\frac{1}{\mu^M} - 4\epsilon$ for algorithm $\mathcal{A}$, thus showing the tightness of our global analysis and hence the non-existence of $\mathcal{A}$.

Constraints. The lower bound instances need to respect a set of constraints (or rules) for the tasks and for the task graph, which are required to show the global results. This will allow us to prove the lower bound regardless of the model, as long as these constraints are satisfied. For convenience, Table 4 lists and labels all the required constraints (R's) along with some definitions (F's). We will refer to them according to their labels and use (R : ✓) to denote that a constraint R is satisfied in the subsequent analysis.

Table 4. List of Constraints (R's) and Definitions (F's) for Constructing Lower Bound Instances

For tasks		**For task graph**	
$t_C^* \leq \frac{\epsilon}{121P^2}$	(R_1)	$P \geq \left(\frac{120900}{\epsilon}\right)^4$	(F_1)
$p_C \geq \mu^M P$	(R_2)	$X = \left\lceil \frac{P - p_C + 1}{p_B} \right\rceil$	(F_2)
$t_A \leq 5t_A^*$	(R_3)	$K = \left\lceil \frac{5t_A^*}{\epsilon X t_B^*} \right\rceil$	(F_3)
$t_B^* = a_B^* = t_B(1)$	(R_4)	$Y = \left\lfloor \frac{XKt_B^*}{t_A^*} \right\rfloor$	(F_4)
$a_B \leq 5a_B^*$	(R_5)	$Z = K(P - p_A^*)$	(F_5)
$0.1 \leq t_B^* \leq 100$	(R_6)		
$p_B \leq P^{3/4}$	(R_7)	$1 \leq X \leq P$	(R_{13})
$p_A^* \leq P^{3/4}$	(R_8)	$XKt_B^*\left(1 - \frac{\epsilon}{5}\right) \leq Yt_A^* \leq XKt_B^*$	(R_{14})
$t_A^* \leq 24t_B^*$	(R_9)	$K(P - P^{3/4}) \leq Z \leq \frac{121P}{\epsilon}$	(R_{15})
$t_D \leq t_B$	(R_{10})		
$t_D^* \leq \frac{\epsilon}{121P^2}$	(R_{11})		
$p_D \leq 4$	(R_{12})		

5.2 Step 1: Local Analysis

In this section, we will show that, for a given model M and any online list scheduling algorithm $\mathcal{A}$ respecting Definition 2, there exist tasks A and B as well as another algorithm $\mathcal{A}^*$ such that $\frac{a_B}{a_B^*(1-\mu^M)} + \frac{t_A}{t_A^*} \geq \frac{1}{\mu^M} - \epsilon$. We start with the following theorem and will prove it separately for each considered model:

Theorem 5. *Given a model $M \in \{\text{Roo}, \text{Com}, \text{Amd}\}$, let $\mathcal{A}$ be an online list scheduling algorithm respecting Definition 2 with a competitive ratio of $\frac{1}{\mu^M} - 4\epsilon$ for some $0 < \epsilon < 1$, and let $P \geq (\frac{120900}{\epsilon})^4$. Then, there exist four tasks A, B, C, and D satisfying the constraints on tasks in Table 4 (R_1 to R_{12}) and another algorithm $\mathcal{A}^*$ such that:*

$$\frac{a_B}{a_B^*(1-\mu^M)} + \frac{t_A}{t_A^*} \geq \frac{1}{\mu^M} - \epsilon. \tag{17}$$

Proof. First, we set $t_C(p) = \frac{\epsilon}{121P^2 \cdot p}$. Indeed, this execution time function belongs to all speedup models,[3] and clearly, we have $(R_1 : \checkmark)$, i.e., constraint (R_1) is satisfied. Further, if we had $p_C < \mu^M P$, then we would have $\frac{t_C}{t_C^*} > \frac{\frac{\epsilon}{121P^2 \cdot \mu^M P}}{\frac{\epsilon}{121P^2 \cdot P}} = \frac{1}{\mu^M}$, which contradicts the competitive ratio of $\mathcal{A}$ on an instance consisting of only one task C $(R_2 : \checkmark)$. Similarly, for task A, we must have $t_A \leq 5t_A^*$ to respect the competitive ratio of $\mathcal{A}$ on an instance consisting of a single such task, as $\frac{1}{\mu^M} < 5$ for all models $(R_3 : \checkmark)$. For task B, we will set $p_B^* = 1$ for all models $(R_4 : \checkmark)$. Also, if we had $a_B > 5a_B^*$, then an instance consisting of P independent such tasks would result in a makespan at least $\frac{Pa_B}{P} = a_B$

[3]For all models, we have $w = \frac{\epsilon}{121P^2}$. Additionally, for the roofline model, $\bar{p} = \infty$; for the communication model, $c = 0$; and for the Amdhal's model, $d = 0$.

for $\mathcal{A}$, since Pa_B is the total area to be completed on P processors, while $\mathcal{A}^*$ can execute all tasks simultaneously in parallel with a resulting makespan of a_B^*, which also contradicts the competitive ratio of $\mathcal{A}$ ($R_5 : \checkmark$).

The following three lemmas will conclude the proof of the theorem by considering each of the three models separately. Note that we only need to define tasks A, B, and D and verify the constraints (R_6) to (R_{12}). □

Lemma 13. *Theorem 5 is true for the roofline model.*

Proof. For the roofline model, we will only use sequential tasks with $\bar{p} = 1$, and set $t_A(p) = t_B(p) = 1$ and $t_D(p) = \frac{\epsilon}{121P^2}$ for all p. Thus, we have ($R_6, R_9, R_{10}, R_{11} : \checkmark$). Clearly, if $\mathcal{A}$ allocates 3 or more processors to task B or D, then running P independent such tasks would result in a makespan at least three times that of the optimal using a single processor, contradicting the competitive ratio of $\mathcal{A}$. Thus, we can assume that $p_B \leq 2 \leq P^{3/4}$ ($R_7 : \checkmark$) and $p_D \leq 2 \leq 4$ ($R_{12} : \checkmark$). We further set $p_A^* = p_B^* = 1$ ($R_8 : \checkmark$). These give $\frac{a_B}{a_B^*} \geq 1$ and $\frac{t_A}{t_A^*} = 1$. With $\mu^{\text{Roo}} = \frac{3+\sqrt{5}}{2}$, we obtain:

$$\frac{a_B}{a_B^*(1-\mu^{\text{Roo}})} + \frac{t_A}{t_A^*} \geq \frac{1}{1-\mu^{\text{Roo}}} + 1 = \frac{1}{\mu^{\text{Roo}}} .$$ □

Lemma 14. *Theorem 5 is true for the communication model.*

Proof. Given algorithm $\mathcal{A}$, we define $\mathcal{U}$ to be the subset of $\{x \in \mathbb{R}^+\}$ such that $\mathcal{A}$ allocates one processor to a task whose execution time function has the form: $t(p) = \frac{x}{p} + p - 1$. By definition, $\mathcal{A}$ always has the same allocation for identical tasks, so $\mathcal{U}$ is well-defined and must satisfy:

- $[0, 0.1] \subseteq \mathcal{U}$, otherwise, there would exist an $x \leq 0.1$ such that $\mathcal{A}$ allocates at least two processors for $t(p) = \frac{x}{p} + p - 1$, and we would have $\frac{a}{a^{\min}} \geq \frac{a(2)}{a(1)} = \frac{x+2}{x} = 1 + \frac{2}{x} \geq 21$, which contradicts the competitive ratio of $\mathcal{A}$.
- $\mathcal{U} \subseteq [0, 64]$, otherwise, there would exist an $x > 64$ such that $\mathcal{A}$ allocates one processor for $t(p) = \frac{x}{p} + p - 1$, and we would have $\frac{t}{t^{\min}} \geq \frac{x}{x/8+7} = \frac{8x+448}{x+56} - \frac{448}{x+56} \geq 8 - \frac{448}{120} > 4$, which also contradicts the competitive ratio of $\mathcal{A}$.

Based on the previous analysis, we now consider $s = sup(\mathcal{U}) \in [0.1, 64]$. By definition, there exists a $\delta \in [0, \frac{1}{P})$ such that $(s - \delta) \in \mathcal{U}$ and $(s - \delta + \frac{1}{P}) \notin \mathcal{U}$. We choose such δ, and set $\bar{w} = s - \delta > 0.09$, $t_A(p) = \frac{\bar{w}}{p} + p - 1$ and $t_B(p) = \frac{\bar{w}+\frac{1}{P}}{p} + p - 1$ with $p_A = 1$ and $p_B > 1$. Thus, we get $t_B^* = \bar{w} + \frac{1}{P}$ ($R_6 : \checkmark$).

We also have $\frac{a_B}{a_B^{\min}} = \frac{\bar{w}+\frac{1}{P}+p_B(p_B-1)}{\bar{w}+\frac{1}{P}} \geq 1 + \frac{(p_B-1)^2}{65}$, from which we can assume $p_B \leq 15$, otherwise, $\frac{a_B}{a_B^{\min}} > 4$, again contradicting $\mathcal{A}$'s competitive ratio. As $P > 81$, it leads to $p_B \leq P^{3/4}$ ($R_7 : \checkmark$). We now show that Inequality (17) is true:

$$\begin{aligned}
\frac{a_B}{a_B^*(1-\mu^{\text{Com}})} + \frac{t_A}{t_A^*} &\geq \frac{\bar{w}+2}{(\bar{w}+\frac{1}{P})(1-\mu^{\text{Com}})} + \frac{\bar{w}}{\frac{\bar{w}}{p_A^*} + p_A^* - 1} \\
&= \frac{\bar{w}+2}{\bar{w}(1-\mu^{\text{Com}})} \cdot \frac{1}{1+\frac{1}{P\bar{w}}} + \frac{\bar{w}}{\frac{\bar{w}}{p_A^*} + p_A^* - 1} \\
&\geq \frac{\bar{w}+2}{\bar{w}(1-\mu^{\text{Com}})} \cdot \left(1 - \frac{1}{P\bar{w}}\right) + \frac{\bar{w}}{\frac{\bar{w}}{p_A^*} + p_A^* - 1}
\end{aligned}$$

$$
\begin{aligned}
&= \frac{\bar{w}+2}{\bar{w}(1-\mu^{\text{Com}})} - \frac{(\bar{w}+2)}{P\bar{w}^2(1-\mu^{\text{Com}})} + \frac{\bar{w}}{\frac{\bar{w}}{p_A^*}+p_A^*-1} \\
&\geq \frac{\bar{w}+2}{\bar{w}(1-\mu^{\text{Com}})} - \frac{2(\bar{w}+2)}{P\bar{w}^2} + \frac{\bar{w}}{\frac{\bar{w}}{p_A^*}+p_A^*-1} \\
&\geq \frac{\bar{w}+2}{\bar{w}(1-\mu^{\text{Com}})} - \frac{16297}{P} + \frac{\bar{w}}{\frac{\bar{w}}{p_A^*}+p_A^*-1} \\
&\geq \frac{1+\frac{2}{\bar{w}}}{1-\mu^{\text{Com}}} + \frac{1}{\frac{1}{p_A^*}+\frac{p_A^*-1}{\bar{w}}} - \epsilon \ .
\end{aligned}
$$

We further consider two cases:

Case 1: $\bar{w} \leq 6$. In this case, we set $p_A^* = 2$ and obtain:

$$
\frac{a_B}{a_B^*(1-\mu^{\text{Com}})} + \frac{t_A}{t_A^*} \geq \frac{1+\frac{2}{\bar{w}}}{1-\mu^{\text{Com}}} + \frac{1}{\frac{1}{2}+\frac{1}{\bar{w}}} - \epsilon \ .
$$

We define $f(\bar{w}) \triangleq \frac{1+\frac{2}{\bar{w}}}{1-\mu^{\text{Com}}} + \frac{1}{\frac{1}{2}+\frac{1}{\bar{w}}}$, and get $f'(\bar{w}) = \frac{4}{(\bar{w}+2)^2} - \frac{2}{(1-\mu^{\text{Com}})\bar{w}^2}$, which is negative in $[0, w^0)$, where w^0 is the smallest positive $\bar{w}$ such that $f'(\bar{w}) = 0$. Solving the equation above, we find $w^0 = \frac{2+2\sqrt{2-2\mu^{\text{Com}}}}{1-2\mu^{\text{Com}}} > 6$. Therefore, we can replace $\bar{w}$ by 6 to obtain:

$$
\begin{aligned}
\frac{a_B}{a_B^*(1-\mu^{\text{Com}})} + \frac{t_A}{t_A^*} &\geq \frac{4}{3(1-\mu^{\text{Com}})} + \frac{3}{2} - \epsilon \\
&= \frac{\alpha^{\text{Com}}}{1-\mu^{\text{Com}}} + \beta^{\text{Com}} - \epsilon \\
&= \frac{1}{\mu^{\text{Com}}} - \epsilon \ .
\end{aligned}
$$

Case 2: $\bar{w} > 6$. In this case, we set $p_A^* = 3$ and obtain:

$$
\frac{a_B}{a_B^*(1-\mu^{\text{Com}})} + \frac{t_A}{t_A^*} \geq \frac{1+\frac{2}{\bar{w}}}{1-\mu^{\text{Com}}} + \frac{1}{\frac{1}{3}+\frac{2}{\bar{w}}} - \epsilon \ .
$$

We again define $f(\bar{w}) \triangleq \frac{1+\frac{2}{\bar{w}}}{1-\mu^{\text{Com}}} + \frac{1}{\frac{1}{3}+\frac{2}{\bar{w}}}$, and get $f'(\bar{w}) = \frac{18}{(\bar{w}+6)^2} - \frac{2}{(1-\mu^{\text{Com}})\bar{w}^2}$, which is positive in (w^0, ∞), where w^0 is the largest positive $\bar{w}$ such that $f'(\bar{w}) = 0$. Solving the equation above, we find $w^0 = \frac{6+18\sqrt{1-\mu^{\text{Com}}}}{8-9\mu^{\text{Com}}} < 6$. Therefore, we can replace $\bar{w}$ by 6 to obtain:

$$
\begin{aligned}
\frac{a_B}{a_B^*(1-\mu^{\text{Com}})} + \frac{t_A}{t_A^*} &\geq \frac{4}{3(1-\mu^{\text{Com}})} + \frac{3}{2} - \epsilon \\
&= \frac{\alpha^{\text{Com}}}{1-\mu^{\text{Com}}} + \beta^{\text{Com}} - \epsilon \\
&= \frac{1}{\mu^{\text{Com}}} - \epsilon \ .
\end{aligned}
$$

In both cases, we get the desired result with ($R_8 : \checkmark$). Further, because $p_A^* \leq 3$ and $\bar{w} > 0.09$, we get $t_A^* \leq \bar{w} + 2 \leq 24\bar{w} \leq 24 t_B^*$ ($R_9 : \checkmark$).

Finally, for task D, we set $t_D(p) = \frac{\epsilon}{121P^2 \cdot p} + p - 1$ and $p_D^* = 1$. Thus, $\mathcal{A}$ must also allocate one processor to the task (i.e., $P_D = 1$), otherwise, $\frac{a_D}{a_D^{\min}} \geq \frac{\frac{\epsilon}{121P^2}+2}{\frac{\epsilon}{121P^2}} > 5$, which contradicts its competitive ratio. Therefore, we have $(R_{10}, R_{11}, R_{12} : \checkmark)$. □

Lemma 15. *Theorem 5 is true for the Amdahl's model.*

Proof. Given algorithm $\mathcal{A}$, we define $\mathcal{U}$ to be the subset of $\{x \in \mathbb{R}^+\}$ such that $\mathcal{A}$ allocates strictly less than $\sqrt{P}$ processors to a task whose execution time function has the form: $t(p) = \frac{x}{p} + \frac{1}{\sqrt{P}}$. By definition, $\mathcal{A}$ always has the same allocation for identical tasks, so $\mathcal{U}$ is well-defined and must satisfy:

- $[0, 0.1] \subseteq \mathcal{U}$, otherwise, there would exist an $x \leq 0.1$ such that $\mathcal{A}$ allocates at least $\sqrt{P}$ processors for $t(p) = \frac{x}{p} + \frac{1}{\sqrt{P}}$, and we would have $\frac{a}{a^{\min}} \geq \frac{a(\sqrt{P})}{a(1)} = \frac{x+1}{x+\frac{1}{\sqrt{P}}} = 1 + \frac{1-\frac{1}{\sqrt{P}}}{x+\frac{1}{\sqrt{P}}} \geq 1 + \frac{0.99}{0.11} = 10$, which contradicts the competitive ratio of $\mathcal{A}$.
- $\mathcal{U} \subseteq [0, 10]$, otherwise, there would exist an $x > 10$ such that $\mathcal{A}$ allocates less than $\sqrt{P}$ processors for $t(p) = \frac{x}{p} + \frac{1}{\sqrt{P}}$, and we would have $\frac{t}{t^{\min}} \geq \frac{t(\sqrt{P})}{t(P)} = \frac{\frac{x}{\sqrt{P}}+\frac{1}{\sqrt{P}}}{\frac{x}{P}+\frac{1}{\sqrt{P}}} > \frac{x}{\frac{x}{\sqrt{P}}+1} = \frac{1}{\frac{1}{\sqrt{P}}+\frac{1}{x}} > \frac{1}{0.11} > 9$, which also contradicts the competitive ratio $\mathcal{A}$.

Based on the previous analysis, we now consider $s = sup(\mathcal{U}) \in [0.1, 10]$. By definition, there exists a $\delta \in [0, \frac{1}{\sqrt{P}})$ such that $(s - \delta) \in \mathcal{U}$ and $(s - \delta + \frac{1}{\sqrt{P}}) \notin \mathcal{U}$. We choose such δ, and set $\bar{w} = s - \delta > 0.09$, $t_A(p) = \frac{\bar{w}}{p} + \frac{1}{\sqrt{P}}$ and $t_B(p) = \frac{\bar{w}+\frac{1}{\sqrt{P}}}{p} + \frac{1}{\sqrt{P}}$ with $p_A < \sqrt{P}$ and $p_B \geq \sqrt{P}$. Thus, we get $t_A^* \leq \bar{w} + \frac{1}{\sqrt{P}}$ and $t_B^* = \bar{w} + \frac{2}{\sqrt{P}}$ $(R_6, R_9 : \checkmark)$. We can further assume $p_B \leq P^{3/4}$, otherwise, we would have $\frac{a_B}{a_B^{\min}} \geq \frac{\bar{w}+\frac{P^{3/4}}{\sqrt{P}}}{\bar{w}+\frac{2}{\sqrt{P}}} \geq \frac{0.09+P^{1/4}}{11} > 5$, which contradicts the competitive ratio of $\mathcal{A}$ $(R_7 : \checkmark)$. Finally, we set $p_A^* = \lfloor P^{3/4} \rfloor$ $(R_8 : \checkmark)$ and get:

$$
\begin{aligned}
\frac{a_B}{a_B^*(1-\mu^{\text{AMD}})} + \frac{t_A}{t_A^*} &\geq \frac{\bar{w}+1}{(\bar{w}+\frac{2}{\sqrt{P}})(1-\mu^{\text{AMD}})} + \frac{\frac{\bar{w}}{\sqrt{P}}+\frac{1}{\sqrt{P}}}{\frac{\bar{w}}{P^{3/4}-1}+\frac{1}{\sqrt{P}}} \\
&= \frac{\bar{w}+1}{\bar{w}(1-\mu^{\text{AMD}})} \cdot \frac{1}{1+\frac{2}{\bar{w}\sqrt{P}}} + \frac{\bar{w}+1}{1+\frac{\bar{w}}{P^{1/4}-\frac{1}{\sqrt{P}}}} \\
&\geq \frac{\bar{w}+1}{\bar{w}(1-\mu^{\text{AMD}})}\left(1 - \frac{2}{\bar{w}\sqrt{P}}\right) + (\bar{w}+1)\left(1 - \frac{\bar{w}}{P^{1/4}-\frac{1}{\sqrt{P}}}\right) \\
&\geq \frac{\bar{w}+1}{\bar{w}(1-\mu^{\text{AMD}})} + \bar{w} + 1 - \frac{22}{0.09^2 \times 0.5\sqrt{P}} - \frac{110}{P^{1/4}-\frac{1}{\sqrt{P}}} \\
&\geq \frac{\bar{w}+1}{\bar{w}(1-\mu^{\text{AMD}})} + \bar{w} + 1 - \epsilon \ .
\end{aligned}
$$

We now set $x = \frac{\bar{w}+1}{\bar{w}} > 1$, so $\frac{x}{x-1} = \bar{w} + 1$. We can finally conclude that:

$$
\begin{aligned}
\frac{a_B}{a_B^*(1-\mu^{\text{AMD}})} + \frac{t_A}{t_A^*} &\geq \frac{x}{1-\mu^{\text{AMD}}} + \frac{x}{x-1} - \epsilon \\
&\geq \min_{x'>1}\left(\frac{x'}{1-\mu^{\text{AMD}}} + \frac{x'}{x'-1}\right) - \epsilon
\end{aligned}
$$

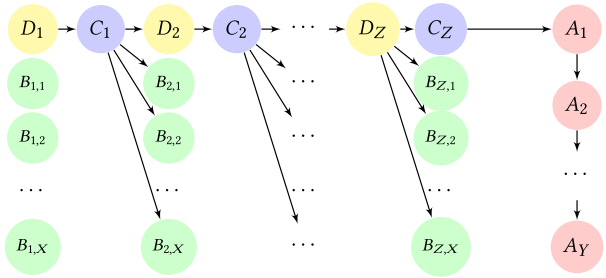

Fig. 1. A task graph for proving lower bounds.

$$= \frac{\alpha^{\text{AMD}}}{1-\mu^{\text{AMD}}} + \beta^{\text{AMD}} - \epsilon$$
$$= \frac{2}{1-\sqrt{8\sqrt{2}-11}} - \epsilon\ .$$

To show the third step above, we can take the derivative of $\frac{x'}{1-\mu^{\text{AMD}}} + \frac{x'}{x'-1}$ and show that its minimum is achieved when x' satisfies $(x'-1)^2 = 1-\mu^{\text{AMD}}$. This is equivalent to $x' = \alpha^{\text{AMD}}$, because $(\alpha^{\text{AMD}}-1)^2 = \frac{(\sqrt{2}-1+\sqrt{2\sqrt{2}-1})^2}{4} = \frac{1+(\sqrt{2}-1)\sqrt{2\sqrt{2}-1}}{2} = \frac{1+\sqrt{8\sqrt{2}-11}}{2} = 1-\mu^{\text{AMD}} = (x'-1)^2$. As $x' > 1$, the only solution is $x' = \alpha^{\text{AMD}}$. From the proof of Lemma 11, we also get that $\beta^{\text{AMD}} = \frac{\alpha^{\text{AMD}}}{\alpha^{\text{AMD}}-1} = \frac{x'}{x'-1}$.

Finally, for task D, we set $t_D(p) = \frac{\epsilon}{121P^2}$ and $p_D^* = 1$. Thus, we must have $p_D \leq 4$, otherwise, $\frac{a_D}{a_D^{\min}} > 4$, which contradicts the competitive ratio of $\mathcal{A}$ (R_{10}, R_{11}, R_{12} : ✓). □

5.3 Step 2: Global Analysis

In this section, we assume that algorithm $\mathcal{A}$ and model M are fixed, while the tasks A, B, C, D, and algorithm $\mathcal{A}^*$ are chosen such that the conditions of Theorem 5 hold. We construct a task graph (as shown in Figure 1), based on which we will show that $\frac{T}{T^*} \geq \frac{a_B}{a_B^*(1-\mu^M)} + \frac{t_A}{t_A^*} - 2\epsilon \geq \frac{1}{\mu^M} - 3\epsilon$.

In our constructed task graph, the tasks are partitioned into four different groups: $\mathcal{T}_A$, $\mathcal{T}_B$, $\mathcal{T}_C$, and $\mathcal{T}_D$. Specifically,

- $\mathcal{T}_A$ has Y tasks identical to A, labeled as $(A_i)_{i\in[1,Y]}$;
- $\mathcal{T}_B$ has XZ tasks identical to B, labeled as $(B_{i,j})_{i\in[1,Z],j\in[1,X]}$;
- $\mathcal{T}_C$ has Z tasks identical to C, labeled as $(C_i)_{i\in[1,Z]}$;
- $\mathcal{T}_D$ has Z tasks identical to D, labeled as $(D_i)_{i\in[1,Z]}$,

where $X = \lceil \frac{P-p_C+1}{p_B} \rceil$, and using $K = \lceil \frac{5t_A^*}{\epsilon X t_B^*} \rceil$, we set $Y = \lfloor \frac{XKt_B^*}{t_A^*} \rfloor$ and $Z = K(P-p_A^*)$. These parameters are specified as definitions (F's) in Table 4.

The tasks are organized in layers and have the following precedence constraints:

- task C_i is the predecessor of task D_{i+1} for $1 \leq i < Z$, and of tasks $B_{i+1,j}$ for $1 \leq i < Z$ and $1 \leq j \leq X$;
- task D_i is the predecessor of task C_i for $1 \leq i \leq Z$;
- task C_Z is the predecessor of task A_1;
- task A_i is the predecessor of task A_{i+1} for $1 \leq i < Y$.

To prove the lower bound, we will first show that the constraints (R_{13}) to (R_{15}) in Table 4 pertaining to the parameters of the task graph are also respected (Lemma 16). We will then show

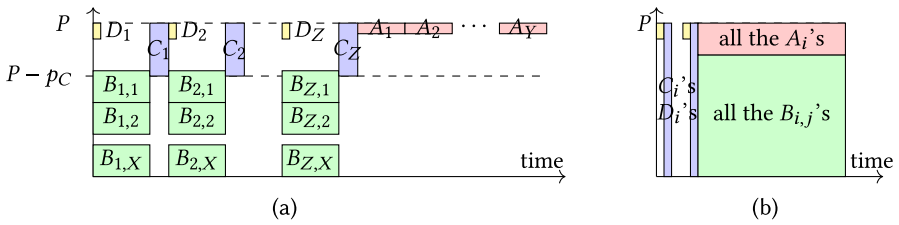

Fig. 2. Shapes of algorithm $\mathcal{A}$'s schedule (a) and algorithm $\mathcal{A}^*$'s schedule (b) for the task graph of Figure 1.

that algorithm $\mathcal{A}$'s schedule must follow the shape as shown in Figure 2(a) (Lemma 17), whereas algorithm $\mathcal{A}^*$ could wait until tasks in $\mathcal{T}_C$ and $\mathcal{T}_D$ are finished before launching tasks in $\mathcal{T}_A$ and $\mathcal{T}_B$, resulting in a better schedule as shown in Figure 2(b). This last result together with Theorem 5 will lead to a contradiction, hence proving the lower bound (Theorem 6). In the following analysis, we will provide a reference to a constraint or a definition whenever it is used.

Lemma 16. *Given the setting above, constraints* (R_{13}), (R_{14}), *and* (R_{15}) *in Table 4 are satisfied.*

Proof. Constraint (R_{13}) can be obtained directly from the definition of X:

$$1 \le X = \left\lceil \frac{P - p_C + 1}{p_B} \right\rceil \le P \,. \qquad (F_2)$$

Constraint (R_{14}) can be derived from the definitions of Y and K:

$$\begin{aligned} XKt_B^* \ge Yt_A^* &\ge XKt_B^* - t_A^* && (F_4)\\ &= XKt_B^* \left(1 - \frac{t_A^*}{XKt_B^*}\right) \\ &\ge XKt_B^* \left(1 - \frac{\epsilon}{5}\right) \,. && (F_3) \end{aligned}$$

Finally, constraint (R_{15}) can be obtained with:

$$\begin{aligned} K(P - P^{3/4}) \le Z &\le KP && (F_5, R_8)\\ &\le \left(\frac{5t_A^*}{\epsilon X t_B^*} + 1\right) P && (F_3)\\ &\le \left(\frac{120}{\epsilon} + 1\right) P && (R_9, R_{13})\\ &\le \frac{121P}{\epsilon} \,. && (\epsilon < 1) \end{aligned}$$

□

Lemma 17. *For a given task* $\mathcal{T}$, *let* $s(\mathcal{T})$ *denote its starting time in algorithm* $\mathcal{A}$*'s schedule and* $e(\mathcal{T})$ *its ending time. If all constraints in Table 4 are satisfied, then algorithm* $\mathcal{A}$*'s schedule must follow the shape as shown in Figure 2(a), i.e.:*

- $s(D_i) = s(B_{i,1}) = \cdots = s(B_{i,X}), \forall i \in [1, Z]$;
- $s(C_i) = e(B_{i,1}) = \cdots = e(B_{i,X}), \forall i \in [1, Z]$;
- $s(D_i) = e(C_{i-1}), \forall i \in [2, Z]$;
- $s(A_1) = e(C_Z)$;
- $s(A_i) = e(A_{i-1}), \forall i \in [2, Y]$.

As a result, the makespan of $\mathcal{A}$ *must satisfy:* $T \ge Zt_B + Yt_A$.

Proof. It is possible to simultaneously run all the tasks $B_{i,j}$'s and D_i in layer i, as the total number of processors required is:

$$\begin{aligned} Xp_B + p_D &\leq \left(\frac{P - p_C + 1}{p_B} + 1\right) P_B + 4 && (F_2, R_{12}) \\ &= P - P_C + P_B + 5 \\ &\leq (1 - \mu^M)P + P^{3/4} + 5 && (R_2, R_7) \\ &< 0.8P + 0.01P + 5 < P\,. && (F_1, \mu_M > 0.2) \end{aligned}$$

However, it is not possible to run all the $B_{i,j}$'s and C_i in parallel, as the number of processors required would be:

$$\begin{aligned} Xp_B + p_C &\geq \frac{P - p_C + 1}{p_B} p_B + p_C && (F_2) \\ &= P + 1\,. \end{aligned}$$

Therefore, given that algorithm $\mathcal{A}$ uses list scheduling to schedule the tasks, we get $s(D_1) = s(B_{1,1}) = \cdots = s(B_{1,X})$. Furthermore, since $t_D \leq t_B$ (R_{10}), C_1 becomes available before the first layer of tasks in $\mathcal{T}_B$ finishes and will be launched as soon as the layer is done, which gives $s(C_1) = e(B_{1,1}) = \cdots = e(B_{1,X})$. A direct induction shows that, using list scheduling, the same scenario would happen for all the Z layers. Finally, the tasks in $\mathcal{T}_A$ are executed one after another after the completion of C_Z, so the schedule corresponds to the one shown in Figure 2(a). Since there are Z layers of tasks in $\mathcal{T}_B$ and Y layers of tasks in $\mathcal{T}_A$, the makespan of algorithm $\mathcal{A}$ must satisfy $T \geq Zt_B + Yt_A$. □

Theorem 6. *Given a model $M \in \{\textsc{Roo}, \textsc{Com}, \textsc{Amd}\}$, there is no online list scheduling algorithm respecting Definition 2 with a competitive ratio strictly less than $\frac{1}{\mu^M}$.*

Proof. We prove the theorem by contradiction. Specifically, we assume that there exists such an algorithm $\mathcal{A}$ with a competitive ratio of $\frac{1}{\mu^M} - 4\epsilon$ for some $0 < \epsilon < 1$, as also assumed in Theorem 5. We will then show, using the constructed task graph, that the makespan of $\mathcal{A}$ satisfies $\frac{T}{T^*} \geq \frac{1}{\mu^M} - 3\epsilon$, which leads to a contradiction, hence implying that no such algorithm exists.

We first bound the makespan T^* of algorithm $\mathcal{A}^*$, assuming that it follows the schedule of Figure 2(b) by first running all the C_i's and D_i's before running the A_i's sequentially while at the same time using one processor to run each of the $B_{i,j}$'s. As there are XZ tasks of $B_{i,j}$'s, executing them all on $P - p_A^*$ processors takes time $\lceil \frac{XZ}{P-p_A^*} \rceil t_B^* = \lceil \frac{XK(P-p_A^*)}{P-p_A^*} \rceil t_B^* = XKt_B^*$ (F_5), whereas executing all the A_i's takes time $Yt_A^* \leq XKt_B^*$ (R_{14}). Therefore, T^* should satisfy:

$$\begin{aligned} T^* &\leq Z(t_C^* + t_D^*) + XKt_B^* \\ &\leq \frac{121P}{\epsilon}\left(\frac{\epsilon}{121P^2} + \frac{\epsilon}{121P^2}\right) + XKt_B^* && (R_1, R_{11}, R_{15}) \\ &\leq \frac{2}{P} + XKt_B^*\,. \end{aligned}$$

Now, using the result of Lemma 17, we get:

$$\begin{aligned} \frac{T}{T^*} &\geq \frac{Zt_B + Yt_A}{\frac{2}{P} + XKt_B^*} \\ &\geq \frac{K(P - P^{3/4})t_B}{2 + XKt_B^*} + \frac{Yt_A}{\frac{2}{P} + Yt_A^* / \left(1 - \frac{\epsilon}{5}\right)} && (R_{14}, R_{15}) \end{aligned}$$

$$
\begin{aligned}
&= \frac{(P - P^{3/4})t_B}{\frac{2}{K} + Xt_B^*} + \frac{t_A}{t_A^* / \left(1 - \frac{\epsilon}{5}\right) + \frac{2}{YP}} \\
&\geq \frac{(P - P^{3/4})t_B}{2 + t_B^* \left(\frac{P(1-\mu^M)+1}{p_B} + 1\right)} + \frac{t_A}{t_A^*} \cdot \frac{1 - \frac{\epsilon}{5}}{1 + \frac{2}{PXKt_B^*}(1 - \frac{\epsilon}{5})} \qquad (F_2, R_2, F_4) \\
&\geq \frac{(P - P^{3/4})t_B}{2 + 2t_B^* + \frac{Pt_B^*(1-\mu^M)}{p_B}} + \frac{t_A}{t_A^*} \cdot \frac{1 - \frac{\epsilon}{5}}{1 + \frac{2}{Pt_B^*}} \\
&\geq \frac{t_B p_B}{\frac{p_B(2+2t_B^*)}{P - P^{3/4}} + \frac{Pt_B^*(1-\mu^M)}{P - P^{3/4}}} + \frac{t_A}{t_A^*} \cdot \frac{1 - \frac{\epsilon}{5}}{1 + \frac{20}{P}} \qquad (R_6) \\
&\geq \frac{a_B}{\frac{2+2t_B^*}{P^{1/4}-1} + \frac{t_B^*(1-\mu^M)}{1-P^{-1/4}}} + \frac{t_A}{t_A^*} \cdot \frac{1 - \frac{\epsilon}{5}}{1 + \frac{20}{P}} \qquad (R_7) \\
&\geq \frac{a_B}{(4 + 4t_B^*)P^{-1/4} + t_B^*(1 - \mu^M)(1 + 2P^{-1/4})} + \frac{t_A}{t_A^*} \cdot \frac{1 - \frac{\epsilon}{5}}{1 + \frac{20}{P}} \,.
\end{aligned}
$$

The last step above assumes $\frac{1}{P^{1/4}-1} \leq 2P^{-1/4}$ and $\frac{1}{1-P^{-1/4}} \leq 1 + 2P^{-1/4}$, both of which are true if $P^{1/4} \geq 2$, i.e., $P \geq 16$. We conclude with the following derivations:

$$
\begin{aligned}
\frac{T}{T^*} &\geq \frac{a_B}{t_B^*(1 - \mu^M) + (4 + 6t_B^*)P^{-1/4}} + \frac{t_A}{t_A^*} \cdot \frac{1 - \frac{\epsilon}{5}}{1 + \frac{20}{P}} \\
&\geq \frac{a_B}{a_B^*(1 - \mu^M) + 604P^{-1/4}} + \frac{t_A}{t_A^*} \cdot \frac{1 - \frac{\epsilon}{5}}{1 + \frac{20}{P}} \qquad (R_4, R_6) \\
&= \frac{a_B}{a_B^*(1 - \mu^M)} \cdot \frac{1}{1 + \frac{604P^{-1/4}}{a_B^*(1-\mu^M)}} + \frac{t_A}{t_A^*} \cdot \frac{1 - \frac{\epsilon}{5}}{1 + \frac{20}{P}} \\
&\geq \frac{a_B}{a_B^*(1 - \mu^M)} \cdot \frac{1}{1 + 12080P^{-1/4}} + \frac{t_A}{t_A^*} \cdot \frac{1 - \frac{\epsilon}{5}}{1 + \frac{20}{P}} \qquad (R_4, R_6, \mu^M \leq 0.5) \\
&\geq \frac{a_B}{a_B^*(1 - \mu^M)} \left(1 - 12080P^{-1/4}\right) + \frac{t_A}{t_A^*} \left(1 - \frac{20}{P}\right)\left(1 - \frac{\epsilon}{5}\right) \\
&\geq \frac{a_B}{a_B^*(1 - \mu^M)} + \frac{t_A}{t_A^*} - 120800P^{-1/4} - \frac{100}{P} - \epsilon \qquad (R_3, R_5, \mu^M \leq 0.5) \\
&\geq \frac{a_B}{a_B^*(1 - \mu^M)} + \frac{t_A}{t_A^*} - 120800P^{-1/4} - 100P^{-1/4} - \epsilon \\
&= \frac{a_B}{a_B^*(1 - \mu^M)} + \frac{t_A}{t_A^*} - 120900P^{-1/4} - \epsilon \\
&\geq \frac{a_B}{a_B^*(1 - \mu^M)} + \frac{t_A}{t_A^*} - 2\epsilon \qquad (F_1) \\
&\geq \frac{1}{\mu^M} - 3\epsilon \,.
\end{aligned}
$$

The last step above applies Theorem 5, and the result proves this theorem and the optimal competitive ratio of our algorithm for these speedup models. □

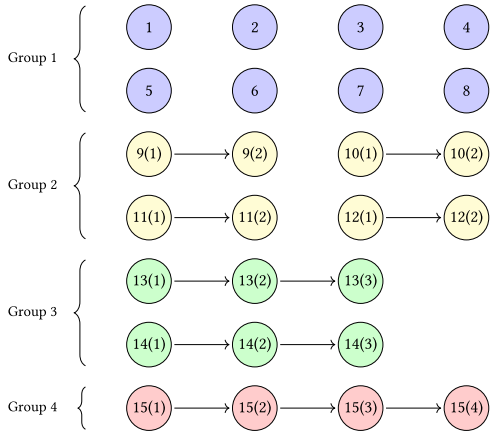

Fig. 3. A lower bound instance in Theorem 7 with $\ell = 2$, $K = 4$, and $n = 15$ linear task chains. Each circle represents a task and the number inside each circle indicates the ID of the linear chain the task is in (and the number in the parentheses indicates the task's position in that linear chain).

Remarks. Since the Amdahl's model is a special case of the general model, its lower bound also applies to the general model.

6 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 three special cases. In this section, we show that the competitive ratio of any deterministic online algorithm (including ours) can be unbounded under an *arbitrary* speedup model.[4]

Theorem 7. *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 $t(p) = \frac{1}{\lg(p)+1}$.[5] Here, lg denotes logarithm to the base 2. We set the total number of processors to be $P = K \cdot 2^{K-1}$.

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 proving 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,

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

[5]As this execution time function satisfies the monotonic properties (defined in Section 3.3), our lower bound result also applies to the monotonic model.

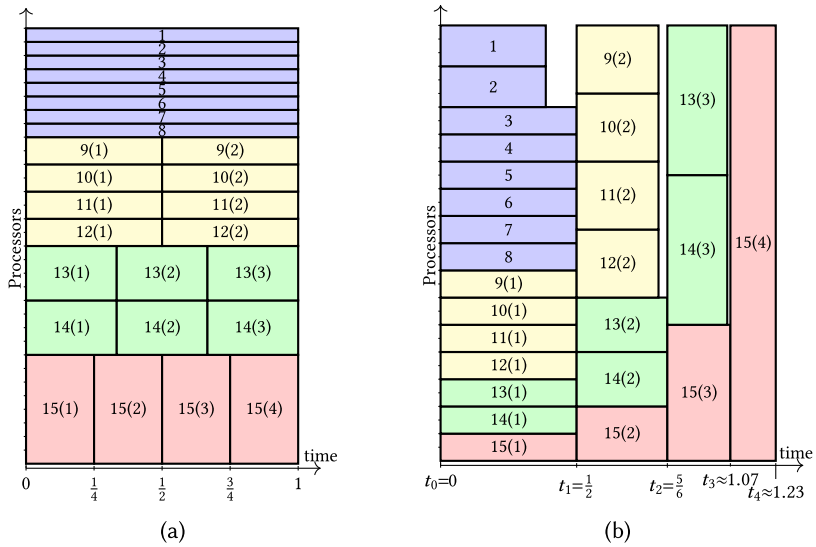

Fig. 4. For the lower bound instance of Figure 3: (a) An offline schedule with a makespan of 1; (b) An online algorithm's schedule, allocating (approximately) the same number of processors to all linear chains and producing a makespan of $t_4 \approx 1.23$.

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 $j \leq i$, 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.

Lemma 18. *Any deterministic online algorithm could produce a schedule that satisfies:* $t_i - t_{i-1} \geq \frac{1}{\ell+i}$, $\forall i \in [1, K]$.

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 ith task by definition (this also holds for t_0 and t_K). Hence, all tasks in the ith 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 $t(p) = \frac{1}{\lg(p)+1} \leq \frac{1}{\ell+i}$, hence their processor allocation must be at least $p \geq 2^{\ell+i-1} = K \cdot 2^{i-1}$. As the area of the task $a(p) = pt(p) = \frac{p}{\lg(p)+1}$ 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 $2^{K-i} \cdot a(K \times 2^{i-1}) = \frac{2^{K-i} \cdot K \cdot 2^{i-1}}{\lg(K \cdot 2^{i-1})+1} = \frac{K \cdot 2^{K-1}}{\ell+i} = \frac{P}{\ell+i}$. 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 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 18 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:

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

□

7 CONCLUSION AND FUTURE WORK

In this article, we have studied the online scheduling of moldable task graphs to minimize makespan with tasks obeying several common speedup models. To the best of our knowledge, no competitive ratio was known under this setting, except for the roofline model [13]. Owing to the design of a new online algorithm and a novel analysis framework, 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 shown that no online list scheduling algorithm with deterministic local decisions for processor allocation may have a better competitive ratio than ours for the roofline, communication, and Amdahl's models. 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 consider extending the algorithm and analysis 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 benchmarking the performance of our algorithm using realistic workflows.

ACKNOWLEDGMENTS

We thank Anne Benoit and Yves Robert for many helpful discussions and for their inputs to the preliminary version of this article [7]. We also thank the anonymous reviewers for valuable comments, which helped improve the final version of this article.

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] Krishna P. Belkhale and Prithviraj Banerjee. 1990. An approximate algorithm for the partitionable independent task scheduling problem. In *ICPP*. 72–75.
[4] Krishna P. Belkhale, Prithviraj Banerjee, and W. Springfield Av. 1991. A scheduling algorithm for parallelizable dependent tasks. In *IPPS*. 500–506.
[5] 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*.
[6] 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.
[7] Anne Benoit, Lucas Perotin, Yves Robert, and Hongyang Sun. 2022. Online scheduling of moldable task graphs under common speedup models. In *ICPP*. 51:1–51:11.
[8] J. Blazewicz, M. Machowiak, G. Mounié, and D. Trystram. 2001. Approximation algorithms for scheduling independent malleable tasks. In *Euro-Par*. 191–197.

[9] Louis-Claude Canon, Loris Marchal, Bertrand Simon, and Frédéric Vivien. 2020. Online scheduling of task graphs on heterogeneous platforms. *IEEE Trans. Parallel Distrib. Syst.* 31, 3 (2020), 721–732.
[10] 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.
[11] Richard A. Dutton and Weizhen Mao. 2007. Online scheduling of malleable parallel jobs. In *PDCS*. 136–141.
[12] 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.
[13] Anja Feldmann, Ming-Yang Kao, Jiří Sgall, and Shang-Hua Teng. 1998. Optimal on-line scheduling of parallel jobs with dependencies. *J. Combinator. Optim.* 1, 4 (1998), 393–411.
[14] R. L. Graham. 1969. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math.* 17, 2 (1969), 416–429.
[15] Jessen T. Havill and Weizhen Mao. 2008. Competitive online scheduling of perfectly malleable jobs with setup times. *Eur. Journal of Oper. Res.* 187 (2008), 1126–1142.
[16] Klaus Jansen. 2012. A $(3/2 + \epsilon)$ approximation algorithm for scheduling moldable and non-moldable parallel tasks. In *SPAA*. 224–235.
[17] K. Jansen and F. Land. 2018. Scheduling monotone moldable jobs in linear time. In *IPDPS*. 172–181.
[18] Klaus Jansen and Ralf Thöle. 2010. Approximation algorithms for scheduling parallel jobs. *SIAM J. Comput.* 39, 8 (2010), 3571–3615.
[19] Klaus Jansen and Hu Zhang. 2005. Scheduling malleable tasks with precedence constraints. In *SPAA*. 86–95.
[20] Klaus Jansen and Hu Zhang. 2006. An approximation algorithm for scheduling malleable tasks under general precedence constraints. *ACM Trans. Algor.* 2, 3 (2006), 416–434.
[21] Berit Johannes. 2006. Scheduling parallel jobs to minimize the makespan. *J. Sched.* 9, 5 (2006), 433–452.
[22] Theodore Johnson, Timothy A. Davis, and Steven M. Hadfield. 1996. A concurrent dynamic task graph. *Parallel Comput.* 22, 2 (1996), 327–333.
[23] Nathaniel Kell and Jessen Havill. 2015. Improved upper bounds for online malleable job scheduling. *J. Sched.* 18, 4 (2015), 393–410.
[24] Renaud Lepère, Denis Trystram, and Gerhard J. Woeginger. 2001. Approximation algorithms for scheduling malleable tasks under precedence constraints. In *ESA*. 146–157.
[25] Walter Ludwig and Prasoon Tiwari. 1994. Scheduling malleable and nonmalleable parallel tasks. In *SODA*. 167–176.
[26] Gregory Mounié, Christophe Rapine, and Dennis Trystram. 1999. Efficient approximation algorithms for scheduling malleable tasks. In *SPAA*. 23–32.
[27] Gregory Mounié, Christophe Rapine, and Denis Trystram. 2007. A 3/2-approximation algorithm for scheduling independent monotonic malleable tasks. *SIAM J. Comput.* 37, 2 (2007), 401–412.
[28] Daniel D. Sleator and Robert E. Tarjan. 1985. Amortized efficiency of list update and paging rules. *Commun. ACM* 28, 2 (1985), 202–208.
[29] John Turek, Joel L. Wolf, and Philip S. Yu. 1992. Approximate algorithms scheduling parallelizable tasks. In *SPAA*.
[30] Jeffrey D. Ullman. 1975. NP-complete scheduling problems. *J. Comput. Syst. Sci.* 10, 3 (1975), 384–393.
[31] Qingzhou Wang and Kam Hoi Cheng. 1992. A heuristic of scheduling parallel tasks and its analysis. *SIAM J. Comput.* 21, 2 (1992), 281–294.
[32] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: An insightful visual performance model for multicore architectures. *Commun. ACM* 52, 4 (2009), 65–76.
[33] Deshi Ye, Danny Z. Chen, and Guochuan Zhang. 2018. Online scheduling of moldable parallel tasks. *J. Sched.* 21, 6 (2018), 647–654.

Received 22 April 2023; revised 11 October 2023; accepted 20 October 2023