Abstract

Lucas: From TOPC paper

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

New results, Fair Heuristic

February 11, 2025

1 Introduction

Lucas: From TOPC24 paper

This work investigates the online scheduling of parallel task graphs on a set of identical processors, where each task in the graph is *moldable*. In the scheduling literature, a moldable task (or job) is a parallel task that can be executed on an arbitrary but fixed number of processors. The execution time of the task depends upon the number of processors used to execute it, which is chosen once and for all when the task starts its execution but cannot be modified later on during execution. This corresponds to a variable static resource allocation, as opposed to a fixed static allocation (*rigid* tasks) and to a variable dynamic allocation (*malleable* tasks) [8].

Moldable tasks offer a nice trade-off between rigid and and malleable tasks: they easily adapt to the number of available resources, contrarily to rigid tasks, while being easy to design and implement, contrarily to malleable tasks. This explains that many computational kernels in scientific libraries for numerical linear algebra and tensor computations are provided as moldable tasks that can be deployed on a wide range of processor numbers. We assume that the scheduling of each task is non-preemptive and without restarts [9], which is a highly desirable approach to avoid high overheads incurred by checkpointing partial results, context switching, and task migration.

Because of the importance and wide availability of moldable tasks, scheduling algorithms for such tasks have received considerable attention. The scheduling problem comes in many flavors:

- **Offline Scheduling vs. Online Scheduling.** In the offline version of the scheduling problem, all tasks are known in advance, before the execution starts. The problem is $\mathcal{NP}$-complete, and the goal is to design good approximation algorithms. On the contrary, in the online version of the problem, tasks are released on the fly, and the objective is to derive competitive ratios [20] for the performance of a scheduling algorithm against an optimal offline scheduler, which knows in advance all the tasks and and their dependencies in the graph. The competitive ratio is established against all possible strategies devised by an adversary trying to force the online algorithm to take *bad* decisions.
- **Independent Tasks vs. Task Graphs.** There are two versions of the online problem, with independent tasks or with task graphs. For the version with independent tasks, the tasks are released on the fly and the scheduler discovers their characteristics only upon release. For the version with task graphs, the whole graph is released at the start, but the scheduler discovers a new task and its characteristics only when all of its predecessors have completed execution. In other words, the shape of the graph and the nature of the tasks are not known in advance and are revealed only as the execution progresses.

In this work, we investigate the most difficult instance of the problem, namely, the online scheduling of moldable tasks graphs, with the goal of minimizing the overall completion time, or the makespan. Our main contribution resides in the design of a new online algorithm and in several new competitive ratios, which greatly depend upon the speedup model of the tasks. Several common yet realistic speedup models have been introduced and analyzed, including the roofline model, the communication model, the Amdahl's model, a general combination of them (see Section 3.1 for definitions), and an extension on the communication model. We provide a constant competitive ratio for each of these five models.

Lucas: To fill : In previous papers, we needed an exact model on the speedup model. Here we don't need anything and match or surpass previous algorithms, in competitive ratio and in performance in practice

The rest of this paper is organized as follows. Section 2 surveys related work. The formal model and problem statement are presented in Section 3. Section 4 is the heart of the paper: we introduce a new online algorithm and prove its competitive ratios for different speedup models; we also prove a lower bound for each model. Section **??** is devoted to proving a lower bound of any deterministic online algorithm for the arbitrary speedup model. Finally, Section **??** concludes the paper and suggests future directions.

Table 1: Instances of the scheduling problem.

Problem Instance	Offline	Online
Independent moldable tasks	[21, 13, 12]	[24, 11, 7, 18]
Moldable task graphs	[22, 6, 19, 15]	[9], [This paper]

2 Related Work

Lucas: From TOPC paper

Several prior studies have considered offline scheduling of independent moldable tasks, and derived approximation results. While some results depend on specific speedup models for the tasks, other results hold for the arbitrary model. Turek et al. [21] designed a 2-approximation list-based algorithm for the arbitrary model. Furthermore, when each task only admits a subset of all possible processor allocations, Jansen [13] presented a $(1.5+\epsilon)$-approximation algorithm, which is tight since it was also shown that the problem cannot have an approximation ratio better than 1.5 unless $\mathcal{P} = \mathcal{NP}$ [16]. For the monotonic model, where the execution time is non-increasing and the area (processor allocation times execution time) is non-decreasing with the number of processors, Jansen and Land [12] further proposed a polynomial-time approximation scheme (PTAS).

For online scheduling of independent moldable tasks that are released on-the-fly, Ye et al. [24] designed a 16.74-competitive algorithm. They also explained how to transform an algorithm for rigid tasks whose makespan is at most ρ times the lower bound into a 4ρ-competitive algorithm for moldable tasks. Further, some algorithms designed in the offline setting will also work online if they make scheduling decisions independently for each task; see for instance [11, 7, 18], which studied the communication model.

For offline scheduling of moldable tasks with dependencies, Wang and Cheng [22] showed that the earliest completion time algorithm is a $(3-2/P)$-approximation for the roofline model, where P denotes the total number of processors on the platform. For the monotonic model, Lepère et al. [19] proposed an algorithm with approximation ratio $3+\sqrt{5}$, which was later improved to 4.73 by Jansen and Zhang [15]. Chen and Chu [6] further proposed improved approximations for a more restrictive model, where the area is a concave function and the execution time is strictly decreasing with the number of processors.

Feldmann et al. [9] designed an online algorithm for moldable tasks with dependencies, under the roofline model. By keeping the system utilization above a given bound and by carefully tuning this bound, their algorithm achieves 2.618-competitiveness, even when the task execution times and the DAG structure are unknown. Canon et al. [5] focused on hybrid platforms with several types of processors (for instance, CPUs and GPUs), and derived competitive ratios depending on the number of such resources, but they did not consider moldable tasks.

Benoit et al. [3, 4] recently investigated the problem of scheduling independent moldable tasks subject to failures, where tasks need to be re-executed after a failure until a successful completion. This corresponds to a semi-online setting, since all tasks are known at the beginning, but failed tasks are only discovered on-the-fly. We do not consider task failures in this paper, but rather focus on the general online scheduling of moldable task graphs (as in [9]). However, our results can readily carry over to the failure scenario.

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

3 Problem Statement

Lucas: From TOPC Paper

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

3.1 Model and Objective

We consider the online scheduling of a ***Directed Acyclic Graph (DAG)*** of moldable tasks on a platform with P identical processors. Let $G = (V, E)$ denote the task graph, where $V = \{1, 2, \ldots, n\}$ represents a set of n tasks and $E \subseteq V \times V$ represents a set of precedence constraints (or dependencies) among the tasks. An edge $(i, j) \in E$ indicates that task j depends on task i, and therefore it cannot be executed before task i is completed. Task i is called the ***predecessor*** of task j, and task j is called the ***successor*** of task i. 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 paper, we focus on the following execution time functions:

- ***Roofline Model*** [23]:

$$t_j(p_j) = \frac{w_j}{\min(p_j, \bar{p}_j)} \ . \tag{1}$$

  This model assumes that the task has a linear speedup until a maximum degree of parallelism $\bar{p}_j \leq P$.

- ***Communication Model*** [11]:

$$t_j(p_j) = \frac{w_j}{p_j} + c_j(p_j - 1) \ . \tag{2}$$

  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]:

$$t_j(p_j) = \frac{w_j}{p_j} + d_j \ . \tag{3}$$

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

- ***Power Communication Model***

$$t_j(p_j) = \frac{w_j}{p_j} + c_j(p_j)^{\gamma_j} \ . \tag{4}$$

  Contrarily to the communication model where the communication cost grow linearly with the number of processor, we add an exponent $0 \leq \gamma_j \leq 1$ to address more possible scenarios. If $\gamma_j = 0$ it corresponds to Amdahl's model whereas if $\gamma_j = 1$, the communication cost grows linearly with p_j as in the Communication Model.

- ***General Model***:

$$t_j(p_j) = \frac{w_j}{\min(p_j, \bar{p}_j)} + d_j + c_j(p_j - 1) \ , \tag{5}$$

  This model is a generalization of the last three models.

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 [17, 9, 1, 5]. 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

Given the execution time function in Equation (5), let us define $s_j = \sqrt{w_j/c_j}$. We can then compute the maximum number of processors that should be allocated to the task as:

$$p_j^{\max} = \min\left(P, \bar{p}_j, \tilde{p}_j\right) \ , \tag{6}$$

where

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

Indeed, allocating more than $p_j^{\max}$ processors to the task will no longer decrease its execution time while only increasing its area. Thus, we can safely assume that the processor allocation of the task should never exceed $p_j^{\max}$ by any reasonable algorithm.

Furthermore, the task is said to satisfy the ***monotonic*** property [19] if the following two conditions hold:

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

  Lucas: This assumption is unnecessary except for the minor dichotomic search in Algorithm 1

- The area is a ***non-decreasing*** function of the processor allocation, i.e., $a_j(p) \leq a_j(q)$ for all $1 \leq p < q \leq p_j^{\max}$.

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

$$\frac{t_j(p)}{t_j(q)} \leq \frac{q}{p} \text{ for all } 1 \leq p < q \leq p_j^{\max} . \tag{7}$$

Lemma 1. *A task j with execution time function given in Equation (5) 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 value 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 Lemma 1, 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 $a_j^{\min} = a_j(1)$. Further, we let t_j^{opt} and a_j^{opt} denote the execution time and area of the task under the processor allocation of an optimal schedule. We now define two quantities that 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 longest length of any 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 [10]. The following lemma states this result.

Lemma 2. $T^{\text{opt}} \geq \max\left(\frac{A^{\text{opt}}}{P}, C^{\text{opt}}\right)$.

4 Online Algorithm

Lucas: Updated from TOPC Paper

In this section, we present FAIR, an online scheduling algorithm, and derive its competitive ratio for the general speedup model (Equation (5)) and its three special cases. We also prove lower bounds on the competitiveness of our algorithm under these speedup models.

4.1 Algorithm Description

We first introduce FAIR's subroutine, Allocate_Processor(j, R), that affects a number of processor to each task j and is presented in Algorithm 1.

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

Algorithm 1: Allocate_Processor(j, R)

Input: Task j and Ratio R
Output: Processor allocation p'_j for the task

// Step 1: Initial Allocation
Compute $p_j^{\max}$ based on Equation (6)
Compute $t_j^{\min} = t_j(p_j^{\max})$ and $a_j^{\min} = a_j(1)$
Find an allocation $p_j \in [1, p_j^{\max}]$ that minimizes $R_j \triangleq \max_p \left(\frac{t_j(p)}{t_j^{\min}}, \frac{a_j(p)}{a_j^{\min}} \right)$
// Step 2: Allocation Adjustment
if $p_j > \lceil \mu(R)P \rceil$ **then** $p'_j \leftarrow \lceil \mu(R)P \rceil$ **else** $p'_j \leftarrow p_j$
return R_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 [3, 4]. Specifically, for each possible allocation $p \in [1, p_j^{\max}]$, we define $g_j(p) \triangleq a_j(p)/a_j^{\min}$ to be the ratio between the area of the task and its minimum area, and $f_j(p) \triangleq t_j(p)/t_j^{\min}$ to be the ratio between the execution time of the task and its minimum execution time. We then find an allocation p that minimizes $\max_p(f_j(p), g_j(p))$. We denote as R_j this resulting maximum. 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 $\lceil \mu(R)P \rceil$ if it is more than $\lceil \mu(R)P \rceil$; otherwise the allocation will be unchanged. It uses R which corresponds to the highest ratio we've seen so far among all the tasks that arrived in the system. More precisely,

$$\mu(R) = \frac{2R + 1 - \sqrt{4R^2 + 1}}{2R} \tag{8}$$

Let p_j denote the initial allocation for the task and p'_j the final allocation. Thus, after the second step, we have:

$$p'_j = \begin{cases} \lceil \mu(R)P \rceil, & \text{if } p_j > \lceil \mu(R)P \rceil \\ p_j, & \text{otherwise} \end{cases} . \tag{9}$$

This step adopts the technique first proposed in [19] and subsequently used in [15, 14].The purpose is to enable the execution of more tasks at any time, thus potentially increasing the overall resource utilization of the platform and reducing the makespan. Finally, Algorithm 1 returns the ratio R obtained in the first step.

Lucas: I probably disagree or at least don't love the formulation, it's to avoid having some tasks ready and a poor processor utilization at the same time, which would not be efficient for the area nor the critical path and could be an issue in the worst case scenario

Algorithm 2 presents the pseudocode of the online scheduling algorithm, which at any time maintains the set of available tasks in a waiting queue Q. At time 0 or whenever a running task completes execution, it checks if new tasks have become available. If so, for each newly available task j, it finds a processor allocation p_j for the task (using Algorithm 1) before inserting it into the queue Q. Then, it checks whether any of the new task resulted in a higher ratio NEWR than the one we had until then. If so, it updates the highest ratio R and update the processor allocation of all tasks in the waiting queue Q (note that since $\mu(R)$ decreases with R, the allocation may only decrease). Finally, it applies the well-known ***list scheduling*** strategy [10] 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.

4.2 General Analysis Framework

We first outline a general analysis framework, under which the competitive ratio of the proposed online algorithm will be derived for different speedup models. The framework is inspired by the analysis shown in [19, 15, 14] for list scheduling as well as the analysis used in [3, 4] for local processor allocation. Together, the result nicely connects the makespan of the online algorithm to the lower bound (Lemma 2), thus proving the competitive ratio.

We start with a small lemma on the function $\mu(x)$ defined in Equation 8

Lemma 3. *$\mu(x) = \frac{2x+1-\sqrt{4x^2+1}}{2x}$ is well defined, positive and decreasing in $[1, \infty)$. Furthermore, $\mu(x) \leq \frac{3-\sqrt{5}}{2}$ and $\frac{x}{1-\mu(x)} = \frac{1}{\mu(x)} - x$ increases with x.*

Algorithm 2: FAIR

```
initialize a waiting queue Q
R = 1
when at time 0 or a running task completes execution do
    // Processor Allocation
    newR = R
    for each new task j that becomes available do
        newR = max(newR, Allocate_Processor(j, R))
        insert task j into the waiting queue Q
    end
    // If the highest ratio changed, update processor allocation
    if newR > R then
        R = newR
        for each task j in the waiting queue Q do
            Allocate_Processor(j, R)
        end
    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
```

Proof. For $x \geq 1$, $\mu(x)$ has the same sign as $2x+1-\sqrt{4x^2+1} > 2x+1-\sqrt{4x^2+4x+1} = 0$, hence it is well defined and positive. To show the other results, we first define $v(x) = \frac{x}{1-\mu(x)}$

$$
\begin{aligned}
v(x) = \frac{x}{1-\mu(x)} &= \frac{x}{\frac{\sqrt{4x^2+1}-1}{2x}} = \frac{2x^2}{\sqrt{4x^2+1}-1} = \frac{\sqrt{4x^2+1}+1}{2} \\
&= \frac{2x+1+\sqrt{4x^2+1}}{2} - x = \frac{(2x+1+\sqrt{4x^2+1})(2x+1+\sqrt{4x^2-1})}{2(2x+1-\sqrt{4x^2+1})} - x \\
&= \frac{(2x+1)^2-(4x^2+1)}{2(2x+1-\sqrt{4x^2+1})} - x = \frac{1}{\mu(x)} - x
\end{aligned}
$$

With $v(x) = \frac{\sqrt{4x^2+1}+1}{2}$, we have shown $v(x)$ is increasing, and with $v(x) = \frac{1}{\mu(x)} - x$, we have shown $\frac{1}{\mu(x)}$ must increase since $v(x)$ increases and $-x$ decreaes with x. □

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. Since no tasks gets ready within an interval, the current value of R is also constant and denoted as $R(I)$ for each interval $I \in \mathcal{I}$. For simplicity we define as μ_I as $\mu(R(I))$. Finally, 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 $p(I) \in (0, \lceil(1-\mu_I)P\rceil)$;
- $\mathcal{I}_3$: subset of intervals that satisfy $p(I) \in [\lceil(1-\mu_I)P\rceil, P]$.

Both are well defined and partition $\mathcal{I}$ since $\mu_I < \frac{1}{2}$ (Lemma 3). The following lemma shows a property for the subset of intervals in $\mathcal{I}_0$.

Lemma 4. *There exists a path f in the graph in which a task is always running in $\mathcal{I}_0$.*

Proof. During $\mathcal{I}_0$, the processor utilization is at most $\lceil(1-\mu_I)P\rceil-1$, so there are at least $P-(\lceil(1-\mu_I)P\rceil-1) \geq \lceil\mu_I P\rceil$ available processors. Based on Algorithm 1, any task in the queue $\mathcal{Q}$ is allocated at most $\lceil\mu_I P\rceil = \lceil\mu(R(I))P\rceil$ processors, since whenever R increases, all allocations are recomputed and then Algorithm 2 tries to process each one of them. Because any potential task in $\mathcal{Q}$ would fit, the queue has to be empty and there is no available task in the queue during $\mathcal{I}_0$. When a task graph is scheduled by the list scheduling algorithm, it is well known that there exists a path f in the graph such that some task along that path will be running whenever there is no available task in the queue [9, 19, 15], hence the result. □

We then consider the following times $t_1, t_2, \cdots t_{m-1}$ in which the value of R was updated, with $t_0 = 0$ and $t_m = T$. R may only be updated upon task discoveries so these timestamps correspond to moment where a task is completed therefore $m \leq n$ is finite. We subdivise $\mathcal{I}_0$ into m parts, $(\mathcal{I}_0^k)_{1\leq k\leq m}$, where $\mathcal{I}_0^k = \mathcal{I}_0 \cap [t_{k-1}, t_k]$. All intervals $I \in \mathcal{I}_0^k$ share the same value of R and $\mu(R)$, denoted as R_k and μ_k, and the $(\mathcal{I}_0^k)_{1\leq k\leq m}$ partition $\mathcal{I}_0$. Similarly, we split $\mathcal{I}_3$ into $(\mathcal{I}_3^j)_{1\leq k\leq m}$, where $\mathcal{I}_3^k = \mathcal{I}_3 \cap [t_{k-1}, t_k]$.

Using the path f stated in Lemma 4, we further split $\mathcal{I}_0^k$ into the following two sub-categories:

- $\mathcal{I}_1^k$: subset of $\mathcal{I}_0^k$ where the processor allocation for the currently running task in f was not reduced (by the second step of Algorithm 1 during the last call for the task);
- $\mathcal{I}_2^k$: subset of $\mathcal{I}_0^k$ where the processor allocation for the currently running task in f was reduced (i.e., the task is running on $\lceil \mu_k P \rceil$ processors).

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

- $\mathcal{I}_{2'}^k$: subset of $\mathcal{I}_2^k$ where the currently running task in f was allocated with strictly fewer processors than in the optimal schedule;
- $\mathcal{I}_{2''}^k$: subset of $\mathcal{I}_2^k$ 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. For any $k \in [1, m]$, let $T_1^k = \sum_{I\in\mathcal{I}_1^k} |I|$, $T_2^k = \sum_{I\in\mathcal{I}_2^k} |I|$, $T_{2'}^k = \sum_{I\in\mathcal{I}_{2'}^k} |I|$, $T_{2''}^k = \sum_{I\in\mathcal{I}_{2''}^k} |I|$ and $T_3^k = \sum_{I\in\mathcal{I}_3^k} |I|$ denote the total durations of the different categories of intervals, respectively. $\mathcal{I}_0$ and $\mathcal{I}_3$ partition $\mathcal{I}$; $(\mathcal{I}_0^k)_{1\leq k\leq m}$ partition $\mathcal{I}_0$ and $(\mathcal{I}_3^k)_{1\leq k\leq m}$ partition $\mathcal{I}_3$. Finally, for any given k, $\mathcal{I}_1^k$ and $\mathcal{I}_2^k$ partition $\mathcal{I}_0^k$, while $\mathcal{I}_{2'}^k$ and $\mathcal{I}_{2''}^k$ partition $\mathcal{I}_2^k$. Therefore, we have $T = \sum_k T_1^k + \sum_k T_2^k + \sum_k T_3^k$. Finally, for any $k \in [1, m]$, we define $z_k \in [0, 1]$ such that $T_{2'}^k = z_k T_2^k$ and $T_{2''}^k = (1 - z_k)T_2$.

The next two lemmas relate these duration 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 5. *For any $k \in [1, m]$, we have:*

$$\mu_k \left(z_k + \frac{1 - z_k}{R_k} \right) T_2^k + \frac{(1 - \mu_k)}{R_k} T_3^k \leq \frac{A_k^{\text{opt}}}{P} . \tag{10}$$

Proof. The area of each task j is non-decreasing with its processor allocation and $p'_j \leq p_j$. Furthermore, assuming the task in running in $\mathcal{I}_2^k \cup \mathcal{I}_3^k \subset [t_{k-1}, t_k]$, it started before t_k and its ratio must be below R_k. Hence, the final area of the task should satisfy $a_j(p'_j) \leq a_j(p_j) \leq R_k a_j^{\min} \leq R_k a_j^{\text{opt}}$. Furthermore, during $\mathcal{I}_{2'}^k$, any running task j from path f satisfies $a_j(p'_j) \leq a_j(p_j^{\text{opt}}) = a_j^{\text{opt}}$ (since by definition, they use less processors than the optimal schedule). We let $A_{2'|f}^k$ (resp. $A_{2''|f}^k$) denote the total area of the fraction of tasks from f running in $\mathcal{I}_{2'}^k$ (resp. $\mathcal{I}_{2''}^k$), and $A_{2'|f}^{k,\text{opt}}$ (resp. $A_{2''|f}^{k,\text{opt}}$) the corresponding fraction of area in an optimal schedule. We have $A_{2'|f}^k \leq A_{2'|f}^{k,\text{opt}}$ and $A_{2''|f}^k \leq R_k A_{2''|f}^{k,\text{opt}}$. Since $\lceil \mu_k P \rceil \geq \mu_k P$ processors are used to run tasks from f in $\mathcal{I}_{2'}^k \cup \mathcal{I}_{2''}^k$, we have $\mu_k T_{2'}^k \leq \frac{A_{2'|f}^k}{P} \leq \frac{A_{2'|f}^{k,\text{opt}}}{P}$ and $\mu T_{2''}^k \leq \frac{A_{2''|f}^k}{P} \leq \frac{R_k A_{2''|f}^{k,\text{opt}}}{P}$.

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

Thus, altogether we can derive:

$$\begin{aligned}
&\mu_k \left(z_k + \frac{1 - z_k}{R_k} \right) T_2^k + \frac{(1 - \mu_k)}{R_k} T_3^k \\
&= \mu_k T_{2'}^k + \frac{\mu_k T_{2''}^k}{R_k} + \frac{(1 - \mu_k)}{R_k} T_3^k \\
&\leq \frac{A_{2'|f}^{k,\text{opt}}}{P} + \frac{A_{2''|f}^{k,\text{opt}}}{P} + \frac{A_3^{k,\text{opt}}}{P} \\
&\leq \frac{A^{k,\text{opt}}}{P} .
\end{aligned}$$

□

Lemma 6. *For any $k \in [1, m]$, if $L^{k,\text{opt}}(f)$ denote the length for the portion of path f executed during $[t_{k-1}, t_k]$ under an optimal schedule, we have:*

$$\frac{T_1^k}{R_k} + (\mu_k z_k + 1 - z_k) T_2^k \leq L^{k,\text{opt}}(f) \ . \tag{11}$$

Note that $\sum_k L^{k,\text{opt}}(f)$ corresponds to the total length of f in an optimal schedule and there fore, $\sum_k L^{k,\text{opt}}(f) \leq C^{\text{opt}}$.

Proof. For any task j from path f running during $\mathcal{I}_1^k$, its processor allocation was not reduced by the second step of Algorithm 1, thus we must have $p'_j = p_j \leq \lceil \mu_k P \rceil$. Therefore, its execution time should satisfy $t_j(p'_j) = t_j(p_j) \leq R_k t_j^{\min} \leq R_k t_j^{\text{opt}}$.

For any task j from path f running during $\mathcal{I}_{2'}^k$, its processor allocation has been reduced, i.e., $p'_j = \lceil \mu_k P \rceil$. Based on Equation (7), the task's execution time should satisfy $\frac{t_j(p'_j)}{t_j^{\min}} = \frac{t_j(\lceil \mu_k P \rceil)}{t_j(p_j^{\max})} \leq \frac{p_j^{\max}}{\lceil \mu_k P \rceil} \leq \frac{P}{\mu_k P} = \frac{1}{\mu_k}$. Thus, we have $t_j(p'_j) \leq \frac{1}{\mu_k} t_j^{\min} \leq \frac{1}{\mu_k} t_j^{\text{opt}}$.

Finally, for any task j from path f running during $\mathcal{I}_{2''}^k$, 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 $L_{1|f}^{k,\text{opt}}$ (resp. $L_{2'|f}^{k,\text{opt}}$ and $L_{2''|f}^{k,\text{opt}}$) denote the length for the portion of path f executed during $\mathcal{I}_1^k$ (resp. $\mathcal{I}_{2'}^k$ and $\mathcal{I}_{2''}^k$) under an optimal schedule. The argument above implies that $T_1^k \leq R_k L_{1|f}^{k,\text{opt}}$, $T_{2'}^k \leq \frac{1}{\mu_k} L_{2'|f}^{k,\text{opt}}$ and $T_{2''}^k \leq L_{2''|f}^{k,\text{opt}}$. Thus, we can derive:

$$\begin{aligned}
&\frac{T_1^k}{R_k} + (\mu_k z_k + 1 - z_k) T_2^k \\
&= \frac{T_1^k}{R_k} + \mu_k T_{2'}^k + T_{2''}^k \\
&\leq L_{1|f}^{k,\text{opt}} + L_{2'|f}^{k,\text{opt}} + L_{2''|f}^{k,\text{opt}} \\
&\leq L^{k,\text{opt}}(f) \ .
\end{aligned}$$

The last inequality is again due to the makespan lower bound shown in Lemma 2.

□

Based on the results of Lemmas 5 and 6, we can now derive an upper bound on the makespan of the online scheduling algorithm as shown below.

Lemma 7. *Given R_m the overall maximum ratio obtained among the tasks of the instance,*

$$\frac{T}{T^{\text{opt}}} \leq \frac{1}{\mu_m} = \frac{2R_m + 1 + \sqrt{4R_m^2 + 1}}{2}. \tag{12}$$

Proof. As the makespan is given by $T^k = T_1^k + T_2^k + T_3^k$, we can multiply both sides by $\frac{1-\mu_k}{R_k}$ and apply Equation (10) to remove the T_3^k term, which gives:

$$\frac{1-\mu_k}{R_k} T^k \leq \frac{1-\mu_k}{R_k} T_1^k + \left(\frac{1 - \mu_k - z_k R_k \mu_k - (1 - z_k)\mu_k}{R_k} \right) T_2^k + \frac{A^{k,\text{opt}}}{P} \ .$$

We can then multiply both sides of the above inequality by $\frac{1}{(1-\mu_k)}$ and apply Equation (11) to remove the T_1^k term. This gives:

$$\frac{T^k}{R_k} \leq \left(\frac{1 - \mu_k - z_k R_k \mu_k - (1 - z_k)\mu_k}{(1-\mu_k) R_k} - \mu_k z_k + z_k - 1 \right) T_2^k + L^{k,\text{opt}}(f) + \frac{1}{(1-\mu_k)} \frac{A^{k,\text{opt}}}{P} \ .$$

Lucas: New after this

We fix $f(z_k) = \frac{1-\mu_k-z_k R_k \mu_k-(1-z_k)\mu_k}{(1-\mu_k)R_k} - \mu_k z_k + z_k - 1$ and will show that we must have $f(z_k) \leq 0$ for $z_k \in [0, 1]$ to upperbound the T_2^k term by 0. First let's derive this function in respect to z_k :

$$f'(z_k) = \frac{\mu_k(1-R_k)}{(1-\mu_k)R_k} + (1-\mu_k) = \frac{\mu_k + R_k\left[(1-\mu_k)^2 - \mu_k\right]}{(1-\mu_k)R_k}$$

All terms are positive since $\frac{3-\sqrt{5}}{2}$ is the solution of $(1-\mu_k)^2 - \mu_k = 0$ and $\forall x \leq \mu(1), (1-x)^2 - x \geq 0$. $\mu_k \leq \frac{3-\sqrt{5}}{2}$ (Lemma 3, so $f'(z_k) \geq 0$ and

$$f(z_k) \leq f(1) = \frac{1-\mu_k - R_k\mu_k}{(1-\mu_k)R_k} - \mu_k$$

.

Lucas: TODO : uniformiser $\mu_k R_k$ (dans cet ordre)

Furthermore,

$$\begin{aligned} f(1) = 0 &\Leftrightarrow \mu_k R_k(1-\mu_k) = 1 - \mu_k - \mu_k R_k \\ &\Leftrightarrow (-R_k)\mu_k^2 + (2R_k+1)\mu_k - 1 = 0 \end{aligned}$$

We solve the second degree equation and get the smallest solution, $\mu_k = \mu(R_k) = \frac{2R_k+1-\sqrt{(2R_k+1)^2-4R_k}}{2R_k}$ which corresponds to the definition given in Equation 8.

Since for any $z_k \in [0,1]$, $f(z_k) \leq f(1) = 0$, we can omit the T_2^k term in the above inequality and get:

$$T^k \leq R_k L^{k,\text{opt}}(f) + \frac{R_k}{(1-\mu_k)}\frac{A^{k,\text{opt}}}{P} \ .$$

Lemma 3 states that $\frac{R_k}{(1-\mu_k)} = \frac{R_k}{(1-\mu(R_k))}$ increases with R_k so we get

$$T^k \leq R_m L^{k,\text{opt}}(f) + \frac{R_m}{(1-\mu_m)}\frac{A^{k,\text{opt}}}{P} \ .$$

Finally,

$$\begin{aligned} T = \sum_k T^k &\leq \sum_k R_m L^{k,\text{opt}}(f) + \frac{R_m}{(1-\mu_m)}\frac{A^{k,\text{opt}}}{P} \\ &= R_m \sum_k L^{k,\text{opt}}(f) + \frac{R_m}{(1-\mu_m)}\sum_k \frac{A^{k,\text{opt}}}{P} \\ &\leq \left(R_m + \frac{R_m}{(1-\mu_m)}\right) T^{opt} = \frac{T^{opt}}{\mu_m} \end{aligned}$$

The last step uses another result from Lemma 3 in which we shown $\frac{1}{\mu(x)} - x = \frac{1+\sqrt{4x^2+1}}{2}$, hence the result. □

Remark 1. *We can easily show that* $\frac{T}{T^{opt}} \leq 2R_m + \frac{1}{2} + \frac{1}{4R_m}$

Lucas: For instance in moldable error (moldable independent tasks subject to failure, the ratio of LPA is $2R_m$ which is almost identical

5 Special case for our speedup models

For each model M, we will give a R^M such that any task following the model has a processor allocation verifying verifying $\frac{t(p)}{t_{min}}$ and $\frac{a(p)}{a_{min}} = 1$

Lemma 8. *For any task that follows the roofline speedup model, there exists a processor allocation that achieves* $R^{\text{Roo}} = 1$. *The corresponding competitive ratio is* $\frac{3+\sqrt{5}}{2} \approx 2.62$

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 $\alpha^{\text{Roo}} = \beta^{\text{Roo}} = 1$. □

Lemma 9. *For any task that follows the communication model, there exists a processor allocation that achieves* $R^{\text{Com}} = \sqrt{2}$. *The corresponding competitive ratio is* $2 + \sqrt{2} \approx 3.41$

Proof. **Lucas:** Part of this proof are identical to ICPP24 paper

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

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

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 $p = p^{\max}$, 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: $w' \leq 2 + 2\sqrt{2}$. 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 $\frac{t(p)}{t^{\min}} = f(w', 1) \leq f(2 + 2\sqrt{2}, 1) = \frac{2+2\sqrt{2}}{\frac{2+2\sqrt{2}}{2}+1} = \frac{2+2\sqrt{2}}{2+\sqrt{2}} = (1+\sqrt{2})(2-\sqrt{2}) = \sqrt{2}$, since $p^{\max} = 2$ when $w' = 2 + 2\sqrt{2}$.

Case 2: $2\sqrt{2} < w' \leq 16$. In this case, we set $p = 2$ and get $\frac{a(p)}{a^{\min}} = \frac{c(w'+2)}{cw'} = 1 + \frac{2}{w'} < 1 + \frac{1}{1+\sqrt{2}} = 1 + \frac{\sqrt{2}-1}{(1+\sqrt{2})(\sqrt{2}-1)} = \sqrt{2}$. As $w' > p^2 = 4$, we can also get $\frac{t(p)}{t^{\min}} = f(w', 2) \leq f(16, 2) = \frac{\frac{16}{2}+1}{\frac{16}{4}+3} = \frac{9}{7} < \sqrt{2}$, since $p^{\max} = 4$ when $w' = 16$.

Case 3: $16 < w' \leq 49$. In this case, we set $p = 3$ and get $\frac{a(p)}{a^{\min}} = \frac{c(w'+3)}{cw'} = 1 + \frac{3}{w'} < 1 + \frac{3}{16} < \sqrt{2}$. As $w' > p^2 = 9$, we can also get $\frac{t(p)}{t^{\min}} = f(w', 3) \leq f(49, 3) = \frac{\frac{49}{3}+2}{\frac{49}{7}+6} = \frac{55}{39} < \sqrt{2}$, since $p^{\max} = 7$ when $w' = 49$.

Case 4: $w' > 49$. In this case, we have $t^{\min} \geq c(2\sqrt{w'} - 1)$, which is the minimum possible execution time if the processor allocation could be non-integers. We set $p = \left\lfloor \sqrt{\frac{w'}{3}} + \frac{1}{2} \right\rfloor$ and obtain $\frac{a(p)}{a^{\min}} = \frac{c(w'+p(p-1))}{cw'} \leq 1 + \frac{1}{w'}\left(\sqrt{\frac{w'}{3}} + \frac{1}{2}\right)\left(\sqrt{\frac{w'}{3}} - \frac{1}{2}\right) \leq 1 + \frac{1}{w'}\frac{w'}{3} = \frac{4}{3}$. Finally, $\frac{t(p)}{t^{\min}} \leq \frac{c\left(\frac{w'}{\sqrt{\frac{w'}{3}}-\frac{1}{2}} + \sqrt{\frac{w'}{3}}\right)}{c(2\sqrt{w'}-1)} = \frac{1}{2-\frac{1}{\sqrt{w'}}}\left(\frac{1}{\frac{1}{\sqrt{3}}-\frac{1}{2\sqrt{w'}}} + \frac{1}{\sqrt{3}}\right)$. This function is clearly decreasing with w', and using $w' > 49$, we get $\frac{t(p)}{t^{\min}} \leq \frac{1}{2-\frac{1}{7}}\left(\frac{1}{\frac{1}{\sqrt{3}}-\frac{1}{14}} + \frac{1}{\sqrt{3}}\right) \approx 1.38 < \sqrt{2}$. □

Lemma 10. *For any task that follows the Amdahl's model, there exists a processor allocation that achieves $R^{\text{AMD}} = 2$. The corresponding competitive ratio is $\frac{5+\sqrt{17}}{2} \approx 4.56$*

Proof. The minimum execution time verify $t^{\min} > d$, and the minimum area with just one processor, i.e., $a^{\min} = a(1) = w + d$.

We fix $p = \min(\lceil \frac{w}{d} \rceil, P)$. Since $p \leq \lceil \frac{w}{d} \rceil$, $\frac{a(p)}{a^{\min}} = \frac{w+dp}{w+d} \leq \frac{w+d\left(\frac{w}{d}+1\right)}{w+d} = \frac{w+d+w}{w+d} \leq 2$. If $p = \min(\lceil \frac{w}{d} \rceil, P)$, $\frac{t(p)}{t^{\min}} \leq \frac{\frac{w}{p}+d}{d} \leq 2$. Otherwise, $p = P$ and we get $t(p) = t^{\min}$ and thus $\frac{t(p)}{t^{\min}} = 1 < 2$.

□

Lemma 11. *For extended communication model, we can obtain $R = 2.0017$. The corresponding competitive ratio is approximetaly 4.56.*

Proof. See Appendix A □

Lemma 12. *For General model, we can obtain $R^{GEN} = 2.018$. The corresponding ratio is approximately 4.597.*

Proof. See Appendix B □

We apply the 5 Lemmas with Theorem 1 and get the table :

Table 2: Summary of parameters and competitive ratios for different speedup models. (rounded to inferior for best possible, superior for our heuristics

Model M	ICPP22	TOPC24	Fair(R^M)	Fair(1)	Best Possible (LB)**
Roofline (Roo)	2.62	2.62	2.62	2.62	$\frac{2}{3-\sqrt{5}} > 2.61$
Comm. (Com)	3.61	3.40	3.41	3.41	$\frac{18}{23-\sqrt{313}} > 3.39$
Amdahl (Amd)	4.74	4.55	4.56	4.56	$\frac{2}{1-\sqrt{8\sqrt{2}-11}} > 4.54$
General (Gen)	5.72	4.58*(4.63)	4.597	4.597	4.54
Extended COM (COM2)			4.56	4.56	4.54
Arbitrary (no superlinear speedup)			$2\sqrt{P}+1$	$2\sqrt{P}+1$	$\frac{\sqrt{P}}{2} - \frac{1}{4}$***

Lucas: *: New result when applying directly the α,β obtained in appendix

Lucas: **: Assuming local deterministic processor allocation (not based on state of the platform)

Lucas: ***: Absolute lower bound, without assuming local decision or list scheduling

Remark 2. Fair *(1) doesn't belong to the class of algorithms that has local deterministic processor allocation, since μ varies with time. However it is because it requires no knowledge of the model at first. If instead of initializing R to 1 we would initialize to R^M for each different model, we would have the same competitive ratio and no modification of R and μ during the processing, hence the lower bounds would then apply.*

6 Absolute lower bound for monotonic model

Lucas: Probably too short to do a section? Can move it in previous section.

We consider $P = K^2$ identical tasks in $\sqrt{P}$ different layers of $\sqrt{P}$ independent tasks. More precisely, we denote tasks as $\mathcal{T}_{i,j}$ with $i \in [1, \sqrt{P}]$ and $j \in [1, \sqrt{P}]$. For each $i \in [1, \sqrt{P}-1]$ there exists a single j_i such that $\mathcal{T}_{i,j_i}$ has successors, and its successors are $\mathcal{T}_{i+1,j}$ for all $j \in [1, \sqrt{P}]$. There are no further precedence constraints. We consider the time function to be of the form $t(p) = A - B(p-1)$ which is affine : each new processor reduces the time function by B. This type of function doesn't achieve the highest possible lower bound, but we choose it since the function seems rather reasonable. More precisely, we chose A and B such that $t(1) = \sqrt{P}$ and $t(P) = 1$, which corresponds to $A = \sqrt{P}$ and $B = \frac{\sqrt{P}-1}{P-1}$.

Theorem 1. *Any algorithm may have an execution time at least $\frac{\sqrt{P}-1}{2}$ times higher than an optimal schedule.*

Proof. Since all tasks are indistinguishable, it is possible that for each i, $\mathcal{T}_{i,j_i}$ is the last completed task of the layer i. To upper-bound the execution time of a layer, we distinguish two cases:

- At least one task has at most $\sqrt{P}$ processors utilized. Then its time of processing is at least $t(\sqrt{P}) = \sqrt{P} - (\sqrt{P}-1)\frac{\sqrt{P}-1}{P-1} = \sqrt{P} - \frac{P-2\sqrt{P}+1}{P-1} \geq \sqrt{P} - 1$
- All tasks use more than $\sqrt{P}$ processors. Then the execution time is at least the area of the layer. Since the area increases with the number of processors, this area is at least $\frac{\sqrt{P}a(\sqrt{P})}{P} = \frac{Pt(\sqrt{P})}{P} \geq \sqrt{P} - 1$

Therefore the total execution time of any algorithm could be higher than $\sqrt{P}$ times the upperbound of the execution time of each layer, and therefore we may always have $T \geq P - \sqrt{P}$.

A feasible schedule would be to execute each $\mathcal{T}_{i,j_i}$ sequentially with P processors, then all the remaing tasks with 1 processors for an execution time of $\sqrt{P} - 1 + \sqrt{P} \geq T^{\text{opt}}$. We get $\frac{T}{T^{\text{opt}}} \geq \frac{P-\sqrt{P}}{2\sqrt{P}-1} = \frac{\sqrt{P}}{2} + \frac{\frac{\sqrt{P}}{2}}{2\sqrt{P}-1} \geq \frac{\sqrt{P}}{2} - \frac{1}{4}$ □

Lucas: No need for figure?

7 Experiments

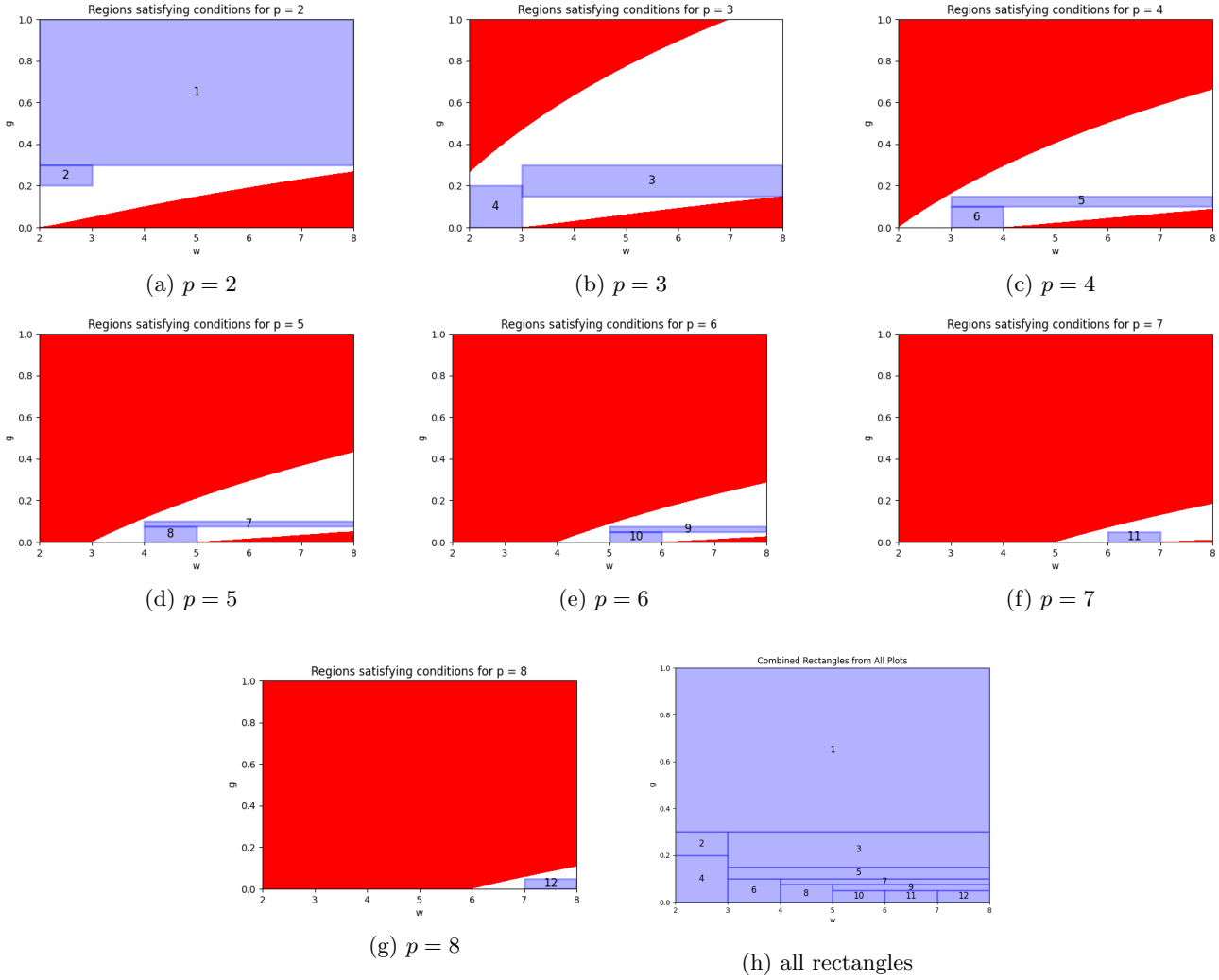
8 Conclusion

A Appendix : Proof extended communication ratio

Theorem 2. *For extended communication model, we can obtain* $\alpha = \beta = 2.002$

Proof. For this model, $t_{min} \geq (\gamma + 1) \left(\frac{w}{g}\right)^{\frac{\gamma}{\gamma+1}}$

Case 1: $w \leq 2$. If $p^{\max} = 1$, then we chose $p = 1$ and get $\alpha = \beta = 1$. Otherwise, we chose $p = 2$ and $a(2) \leq 2a_{min}$ since $t(2) \leq t(1)$. Finally, $t(2) = \frac{w}{2} + 2^{\gamma} \leq 1 + 2^{\gamma} \leq 1 + p^{\max \gamma} \leq 1 + t_{min}$. Since $t_{min} \geq 1$ we have $\beta \leq 2$.



(a) $p = 2$ (b) $p = 3$ (c) $p = 4$

(d) $p = 5$ (e) $p = 6$ (f) $p = 7$

(g) $p = 8$ (h) all rectangles

Figure 1: Comparative plots for different values of p.

Case 2: $2 \leq w < 8$. Let $\alpha(w, \gamma, p) = \frac{a(w,\gamma,p)}{a_{min}(\gamma,p)} = \frac{w+p^{\gamma+1}}{w+1}$ and $\beta(w, \gamma, p) = \frac{\frac{w}{p}+p^{\gamma}}{\frac{w}{p^{\max}}+p^{\max \gamma}}$. The objective is to show that for any couple (w, γ), there exist a p such that $\alpha(w, \gamma, p) \leq 2$ and $\beta(w, \gamma, p) \leq 2$.

We notice that $\alpha(w, \gamma, p) = 1 + \frac{p^{\gamma+1}-1}{w+1}$ increases with γ and decreases with w. Similarly, if $p^{\max} \geq p$, $\beta(w, \gamma, p)$ increases with w since a higher w means a higher parallelism and therefore a higher ratio $\frac{t}{t_{min}}$, and decreases with γ for the opposite reason. Finally, $p^{\max}$ increases with w and decreases with γ. Therefore, if for a couple (w, γ), we are given $w_1 \leq w \leq w_2$, $\gamma_2 \leq \gamma \leq \gamma_1$ and p such that $p^{\max}(w_1, \gamma_1) \geq p$, $\alpha(w_1, \gamma_1, p) \leq 2$ and $\beta(w_2, \gamma_2, p) \leq 2$, we will be assured $\alpha(w, \gamma, p) \leq 2$ and $\beta(w, \gamma, p) \leq 2$. We further illustrates this with Figure 1: for each $p \in [2, 8]$ we color in red the areas for which $\alpha(w, \gamma, p) > 2$ (top left area) and for which $\beta(w, \gamma, p) > 2$ (bottom right area). We will subdivise the zone $(w \in [2, 8], \gamma \in [0, 1])$ into 13 subrectangles, each of them being assigned to one value of p. For instance, rectangle 3 is assigned to $p = 3$. To show that the rectangle doesn't cover any region that doesn't

Rectangle	(w_1, γ_1)	(w_2, γ_2)	p	$p_{\max}(w_1, \gamma_1, p)$	$\alpha(w_1, \gamma_1, p)$	$\beta(w_2, \gamma_2, p)$
1	$(2, 1)$	$(8, 0.3)$	2	1	2.00	1.89
2	$(2, 0.3)$	$(3, 0.2)$	2	4	1.49	1.41
3	$(3, 0.3)$	$(8, 0.15)$	3	6	1.79	1.99
4	$(2, 0.2)$	$(3, 0)$	3	7	1.91	2.00
5	$(3, 0.15)$	$(8, 0.1)$	4	14	1.98	1.92
6	$(3, 0.1)$	$(4, 0)$	4	22	1.90	2.00
7	$(4, 0.1)$	$(8, 0.075)$	5	29	1.97	1.83
8	$(4, 0.075)$	$(5, 0)$	5	40	1.93	2.00
9	$(5, 0.075)$	$(8, 0.05)$	6	50	1.98	1.82
10	$(5, 0.05)$	$(6, 0)$	6	80	1.93	2.00
11	$(6, 0.05)$	$(7, 0)$	7	96	1.96	2.00
12	$(7, 0.05)$	$(8, 0)$	8	111	1.98	2.00

Figure 2: Verification of all extreme values in sub-areas

satisfy our conditions (in red), we simply need to verify the top left corner for α, the bottom right corner for β, and $p \leq p^{\max}$ for the top left corner. All these verifications are summed up in the table in Figure **??** (note that the value are rounded but the 2.00 are exactly 2)

Case 3: $w \geq 8$ and $\gamma \leq 0.1$ We use $p = \left\lceil w^{\frac{1}{\gamma+1}} \right\rceil$. Then

$$\begin{aligned}
\frac{t(p)}{t_{min}} &= \frac{\frac{w}{p} + p^\gamma}{t_{min}} \\
&\leq \frac{\frac{w}{w^{\frac{1}{\gamma+1}}} + \left(w^{\frac{1}{\gamma+1}} + 1\right)^\gamma}{(\gamma+1)\left(\frac{w}{\gamma}\right)^{\frac{\gamma}{\gamma+1}}} \\
&\leq \frac{w^{\frac{\gamma}{\gamma+1}} + w^{\frac{\gamma}{\gamma+1}}\left(1 + \frac{1}{w^{\frac{1}{\gamma+1}}}\right)^\gamma}{w^{\frac{\gamma}{\gamma+1}}\left(\frac{\gamma+1}{\gamma^{\frac{\gamma}{\gamma+1}}}\right)} \\
&\leq \frac{\gamma^{\frac{\gamma}{\gamma+1}}}{\gamma+1}\left(1 + \left(1 + \frac{1}{w^{\frac{1}{\gamma+1}}}\right)^\gamma\right) \\
&\leq \frac{1^{\frac{\gamma}{\gamma+1}}}{\gamma+1}\left(1 + \left(1 + \frac{1}{1^{\frac{1}{\gamma+1}}}\right)^\gamma\right) = \frac{2^\gamma + 1}{\gamma+1}
\end{aligned}$$

If we define $f(\gamma) = 2^\gamma + 1$ and $g(\gamma) = \gamma + 1$, we have $f(0) \leq 2g(0)$ and $f(1) \leq g(1)$. g is affine and f is convexe thus $\forall \gamma, f(\gamma) \leq 2g(\gamma)$ and $\beta = \frac{t(p)}{t_{min}} \leq 2$. □

$$\begin{aligned}\frac{a(p)}{a_{min}} &= \frac{w+p^{\gamma+1}}{w+1}\\ &\leq \frac{w+\left(w^{\frac{1}{\gamma+1}}+1\right)^{\gamma+1}}{w+1}\\ &= \frac{w+w\left(1+w^{-\frac{1}{\gamma+1}}\right)^{\gamma+1}}{w+1}\\ &\leq \frac{w+w\left(1+w^{\frac{-1}{1.1}}\right)^{1.1}}{w+1}\end{aligned}$$

Because $(1+x)^a = \sum_{k=0}^{\infty} \frac{a(a-1)\cdots(a-k+1)}{k!}x^k$, with $1 < a < 2$ and $x < 1$, we can upperbound the sum by the first three terms. Indeed the fourth term is negative and from there it's a sum that alternates positive and negative values of decreasing norm, hence $\sum_{k=3}^{\infty} \frac{a(a-1)\cdots(a-k+1)}{k!}x^k < 0$, therefore $\left(1+w^{\frac{-1}{1.1}}\right)^{1.1} \leq 1 + 1.1w^{\frac{-1}{1.1}} + \frac{0.11}{2}w^{\frac{-2}{1.1}} \leq 1 + 1.1w^{\frac{-1}{1.1}} + 0.1w^{-1}$. We conclude

$$\begin{aligned}\frac{a(p)}{a_{min}} &\leq \frac{w+w(1+1.1w^{\frac{-1}{1.1}}+0.1w^{-1})}{w+1}\\ &\leq \frac{2w+1.1w^{\frac{1}{11}}+0.1)}{w+1}\\ &= 2+\frac{1.1w^{\frac{1}{11}}-1.9}{w+1}\end{aligned}$$

If $w < \frac{1.9^{11}}{1.1}$, then $\frac{a(p)}{a_{min}} \leq 2$, otherwise $\frac{a(p)}{a_{min}} \leq 2 + 1.1w^{-10/11} \leq 2 + 1.9^{-10} < 2.002$. In both case $\alpha < 2.002$.

Case 4: $w \geq 8$ and $\gamma > 0.1$. We use $p = \left\lfloor w^{\frac{1}{\gamma+1}} \right\rfloor$.

$$\begin{aligned}\frac{t(p)}{t_{min}} &= \frac{\frac{w}{p}+p^{\gamma}}{t_{min}}\\ &\leq \frac{\frac{w}{w^{\frac{1}{\gamma+1}}-1}+\left(w^{\frac{1}{\gamma+1}}\right)^{\gamma}}{(\gamma+1)\,\frac{w}{\gamma}^{\frac{\gamma}{\gamma+1}}}\\ &= \frac{\gamma^{\frac{\gamma}{\gamma+1}}}{\gamma+1}\frac{\frac{w^{\frac{\gamma}{\gamma+1}}}{1-w^{\frac{-1}{\gamma+1}}}+w^{\frac{\gamma}{\gamma+1}}}{w^{\frac{\gamma}{\gamma+1}}}\\ &\leq \frac{1}{\gamma+1}\left(\frac{1}{1-w^{\frac{-1}{\gamma+1}}}+1\right)\end{aligned}$$

Since the function decreases with γ and w, we can replace w by 8 and γ by 0.1 which gives $\beta < 1.98$.

Finally,

$$\begin{aligned}\frac{a(p)}{a_{min}} &= \frac{w+p^{\gamma+1}}{w+1}\\ &\leq \frac{w+\left(w^{\frac{1}{\gamma+1}}\right)^{\gamma+1}}{w+1} \leq \frac{2w}{w+1} < 2\end{aligned}$$

B Appendix : Proof of General ratio

Lemma 13. *For any task that follows the general model, there exists a processor allocation that achieves* $\alpha^{\text{GEN}} = 2.018$ *and* $\beta^{\text{GEN}} = 2.018$.

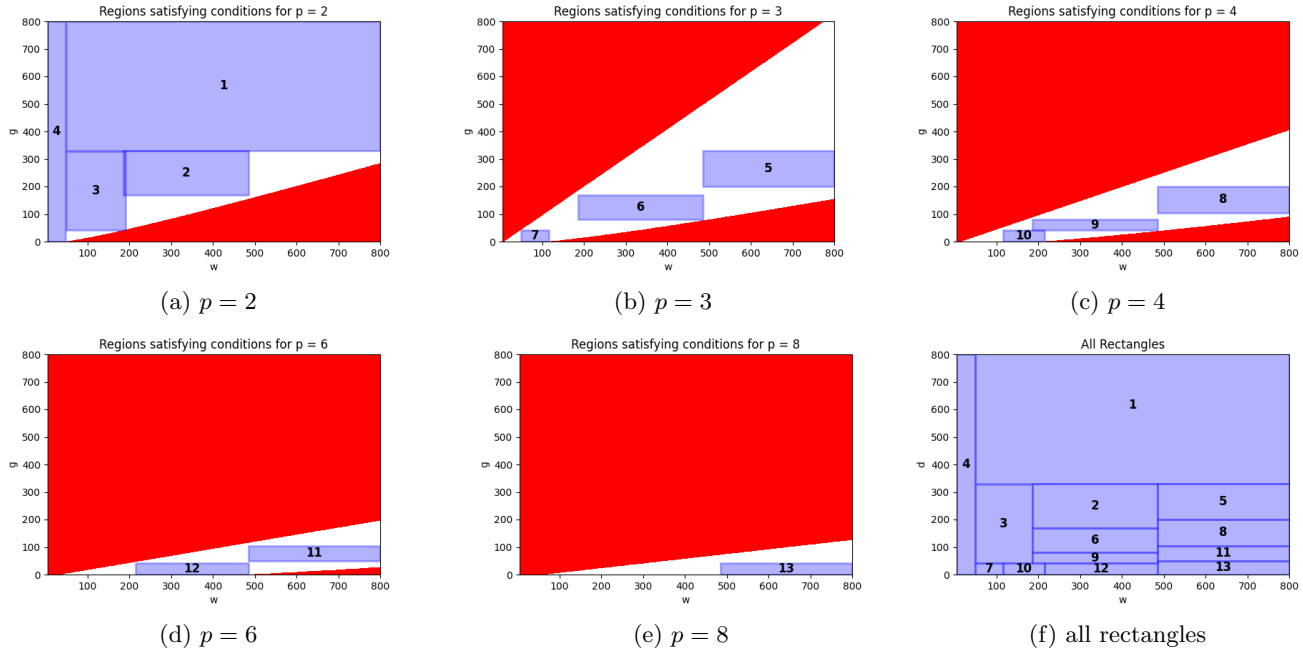
Proof. If we allow the processor allocation to take non-integer values and assuming unbounded $\bar{p}$, the execution time function $t(p)$ would be minimized at $p^* = \sqrt{w'}$. Thus, the minimum execution time should satisfy $t^{\min} \geq c(2\sqrt{w'} + d' - 1)$. Note that this bound will hold true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

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

We consider three cases.

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

Case 2: $4 < w' \leq 800$ and $d > 800$.In this case, we chose $p = 1$ since $t_{min} > d > w'$ and $t(1) = w + d < 2t_{min}$.



(a) $p = 2$ (b) $p = 3$ (c) $p = 4$

(d) $p = 6$ (e) $p = 8$ (f) all rectangles

Figure 3: Comparative plots for different values of p.

Case 3: Here we will first assume $\bar{p}$ is unbounded. Similarly to previous appendix, we notice that $\alpha(w, d, p) = 1 + \frac{(p-1)(d+p)}{w+d}$ decreases with w. When derivating subject to d, we get $\frac{\partial\alpha(w,d,p)}{\partial d} = \frac{(w-p)(p-1)}{(w+d)^2} > 0$ if $w > p$. Similarly, if $p^{\max} \geq p$, $\beta(w, d, p)$ increases with w since a higher w means a higher parallelism and therefore a higher ratio $\frac{t}{t_{min}}$, and decreases with d for the opposite reason. Finally, $p^{\max}$ increases with w and decreases with d.

Therefore, if for a couple (w, d), we are given $w_1 \leq w \leq w_2$, $d_2 \leq d \leq d_1$ and p such that $p^{\max}(w_1, d_1) \geq p$, $w_1 \geq p$, $\alpha(w_1, d_1, p) \leq 2$ and $\beta(w_2, d_2, p) \leq 2$, we will be assured $\alpha(w, d, p) \leq 2$ and $\beta(w, d, p) \leq 2$. We further illustrates this with Figure 1: for each $p \in [2, 8]$ we color in red the areas for which $\alpha(w, d, p) > 2$ (top left area) and for which $\beta(w, d, p) > 2$ (bottom right area). Again, we will subdivise the zone $(w \in [4, 800], d \in [0, 800])$ into 12 subrectangles, each of them being assigned to one value of p. For instance, rectangle 5 is assigned to $p = 3$. To show that the rectangle doesn't cover any region that doesn't satisfy our conditions (in red), we simply need to verify the top left corner for α, the bottom right corner for β, and $p \leq p^{\max}$ for the top left corner. All these verifications are summed up in the table in Figure 4.

We have shown that if $\bar{p}$ is unbounded, there exists p' such that $\frac{t(p')}{t_{min}} \leq 2.018$ and $\frac{a(p')}{a_{min}} \leq 2.018$. If $\bar{p} > p'$, then we just fix $p = \bar{p} = p^{\max}$, the area will decrease and we get $t(p) = t_{min}$

Case 4: $w' > 800$ and $\bar{p} \geq 2$.

Rectangle	(w_1, γ_1)	(w_2, γ_2)	p	$p_{min}(w_1, \gamma_1, p)$	$f(w_1, \gamma_1, p)$	$g(w_2, \gamma_2, p)$
1	$(49, 800)$	$(800, 330)$	2	7	1.945	1.896
2	$(185, 330)$	$(485, 170)$	2	14	1.645	1.941
3	$(49, 330)$	$(185, 42)$	2	7	1.876	1.986
4	$(4, 800)$	$(49, 0)$	2	2	1.998	1.962
5	$(485, 330)$	$(800, 200)$	3	22	1.817	1.834
6	$(185, 170)$	$(485, 80)$	3	14	1.975	1.980
7	$(49, 42)$	$(115, 0)$	3	7	1.989	1.972
8	$(485, 200)$	$(800, 105)$	4	22	1.893	1.918
9	$(185, 80)$	$(485, 42)$	4	14	1.951	1.955
10	$(115, 42)$	$(215, 0)$	4	11	1.879	2.003
11	$(485, 105)$	$(800, 50)$	6	22	1.941	1.784
12	$(215, 42)$	$(485, 0)$	6	15	1.934	1.994
13	$(485, 50)$	$(800, 0)$	8	22	1.759	1.925

Figure 4: Verification of all extreme values in sub-areas

Lucas: This derivation is from ICPP24 paper, except we use 800 instead of 49 for better bounds

In this case, we will set $p = \min\left(\left\lfloor \frac{w'+d'}{\sqrt{w'}+d'} + \frac{1}{2} \right\rfloor, \bar{p}\right)$ and get:

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

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

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

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

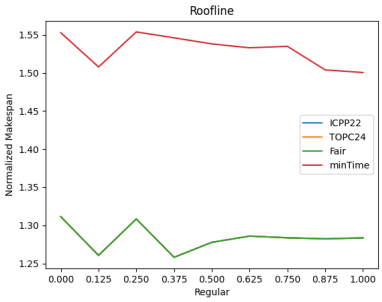
$$\begin{aligned}
\frac{t(p)}{t^{\min}} &\leq \frac{\frac{w'}{p} + d' + p - 1}{\sqrt{w'} + d'} \\
&\leq \frac{\frac{w'(\sqrt{w'}+d')}{w'-\frac{1}{2}\sqrt{w'}}}{\sqrt{w'} + d'} + \frac{d' + \frac{w'+d'}{\sqrt{w'}+d'}}{\sqrt{w'} + d'} \\
&\leq \frac{1}{1 - \frac{1}{2\sqrt{w'}}} + \frac{d'(\sqrt{w'} + d')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

  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 $w' > 800$, we get $\frac{t(p)}{t^{\min}} \leq \frac{1}{1-\frac{1}{\sqrt{800}}} + 1 < 2.018$.

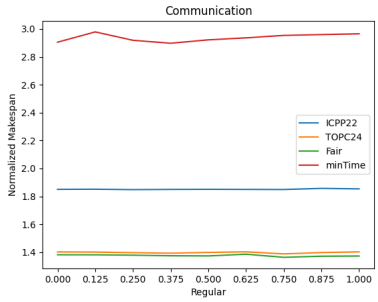
- If $p = \bar{p} < \left\lfloor \frac{w'+d'}{\sqrt{w'}+d'} + \frac{1}{2} \right\rfloor$, and since $\bar{p}$ is an integer, then it is necessarily the case that $\bar{p} \leq \left\lfloor \frac{w'+d'}{\sqrt{w'}+d'} + \frac{1}{2} \right\rfloor - 1 \leq \frac{w'+d'}{\sqrt{w'}+d'} \leq \sqrt{w'}$ (because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

C Appendix : Full Experimental Results

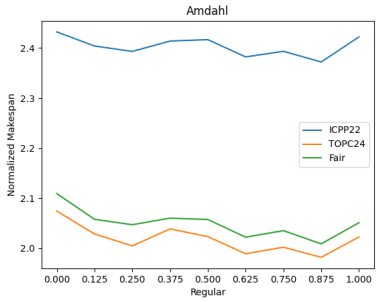
C.1 Lines Figures



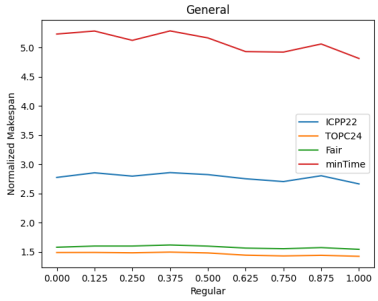
(a) Roofline



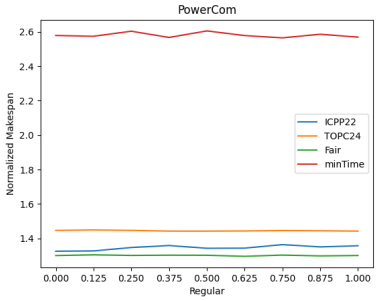
(b) Communication



(c) Amdahl

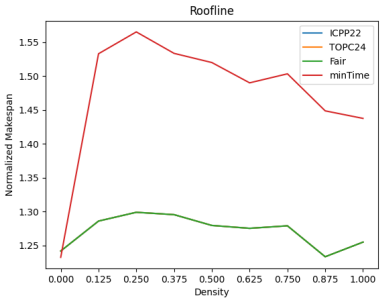


(d) General

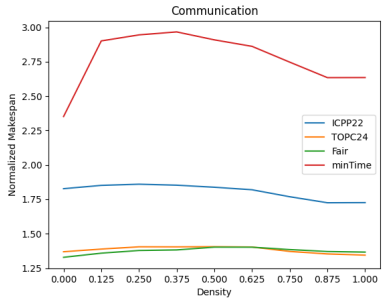


(e) PowerCom

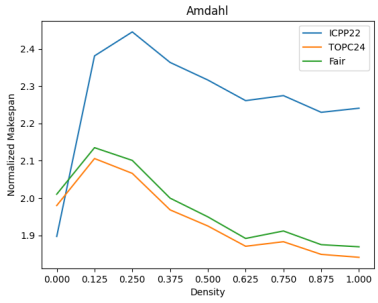
Figure 5: Lines Figure for Regular



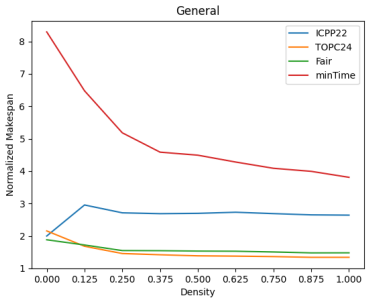
(a) Roofline



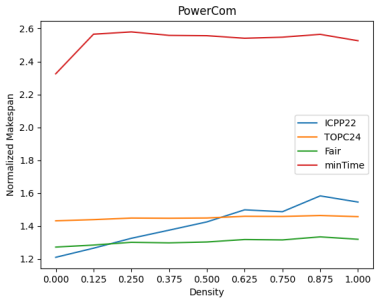
(b) Communication



(c) Amdahl

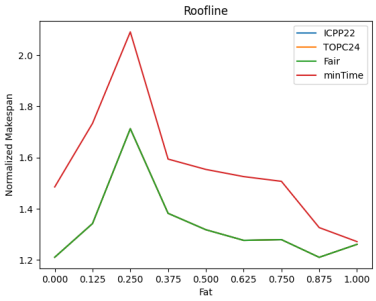


(d) General

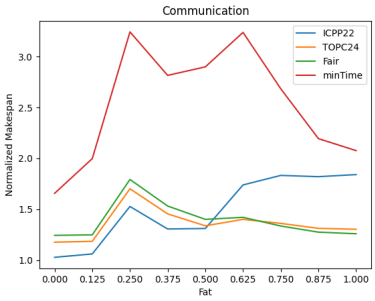


(e) PowerCom

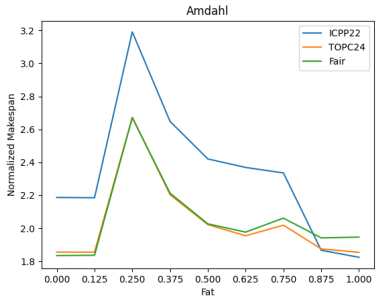
Figure 6: Lines Figure for Density



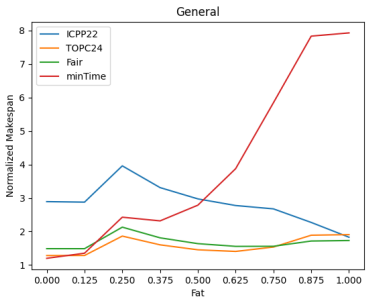
(a) Roofline



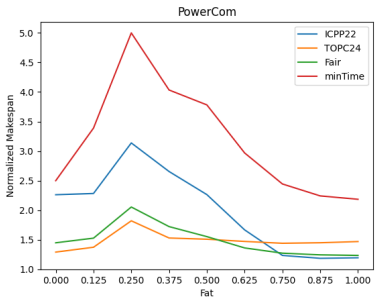
(b) Communication



(c) Amdahl

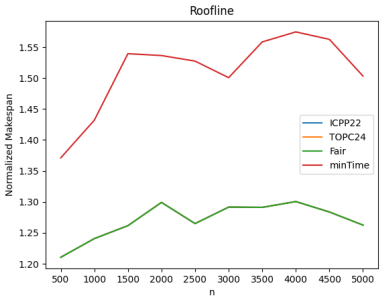


(d) General

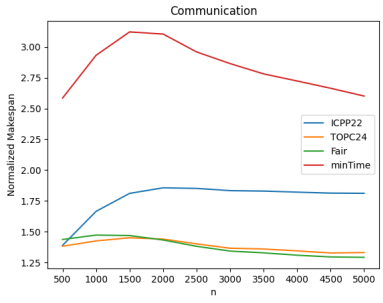


(e) PowerCom

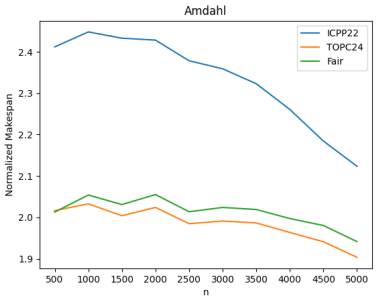
Figure 7: Lines Figure for Fat



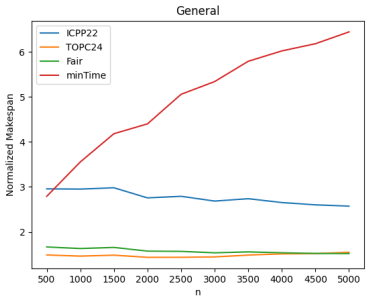
(a) Roofline



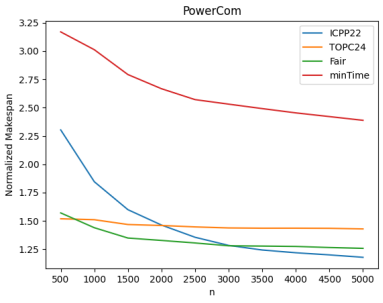
(b) Communication



(c) Amdahl

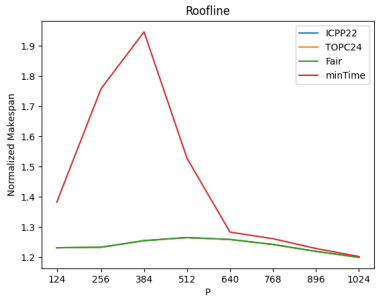


(d) General

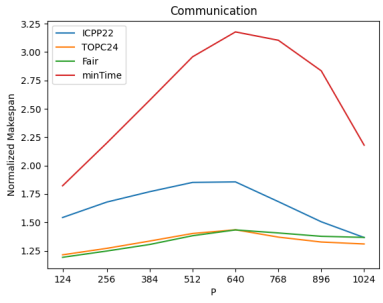


(e) PowerCom

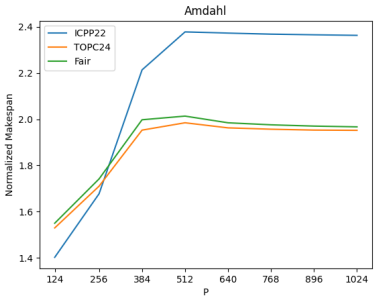
Figure 8: Lines Figure for n



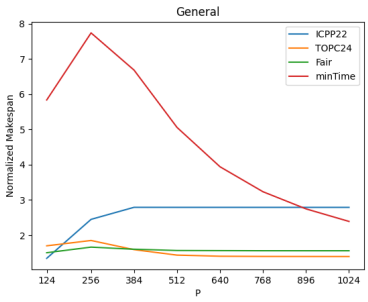
(a) Roofline



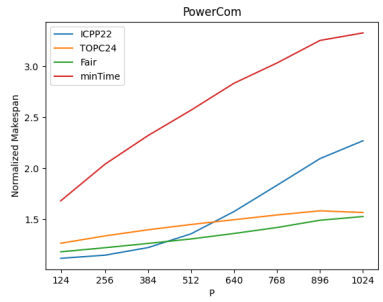
(b) Communication



(c) Amdahl

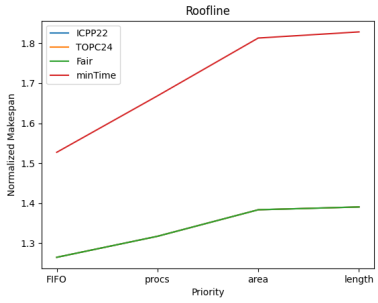


(d) General

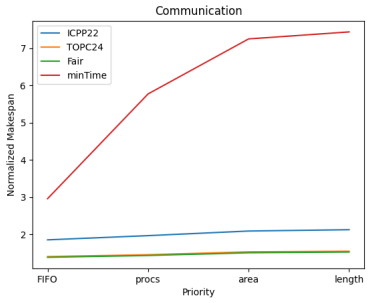


(e) PowerCom

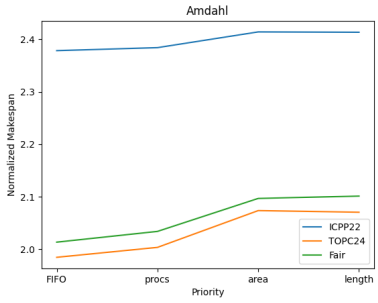
Figure 9: Lines Figure for P



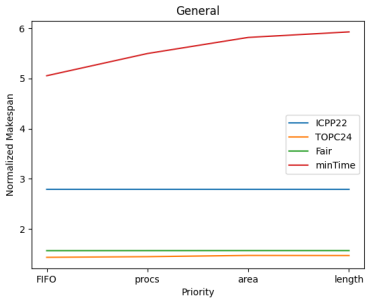
(a) Roofline



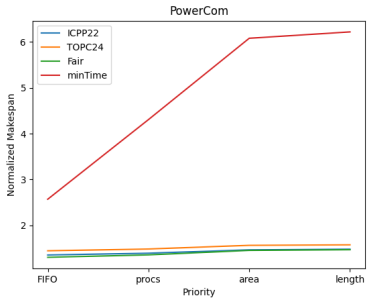
(b) Communication



(c) Amdahl

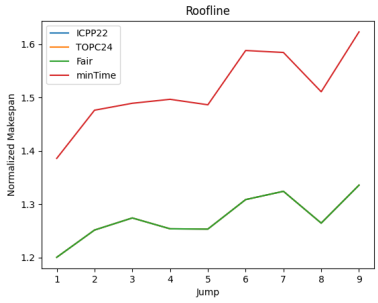


(d) General

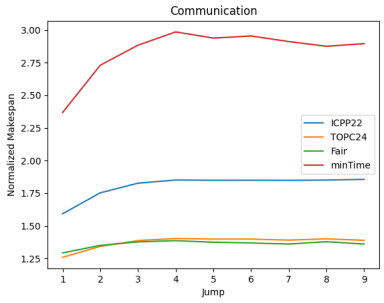


(e) PowerCom

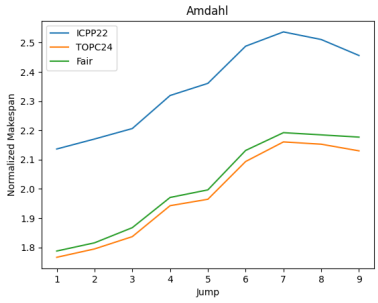
Figure 10: Lines Figure for Priority



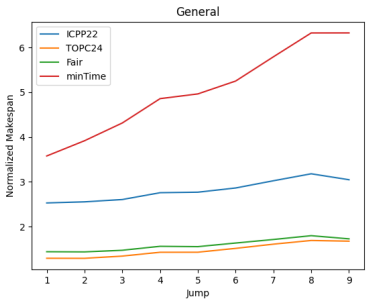
(a) Roofline



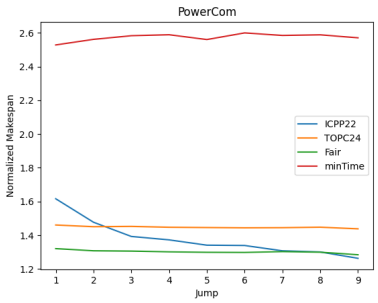
(b) Communication



(c) Amdahl



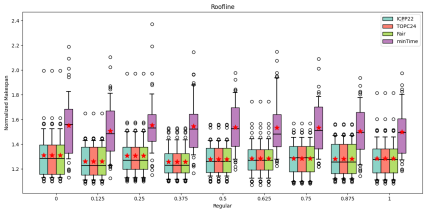
(d) General



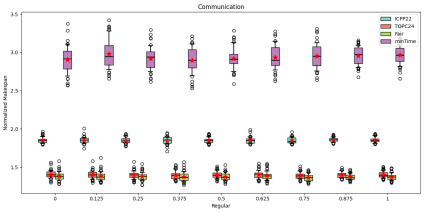
(e) PowerCom

Figure 11: Lines Figure for Jump

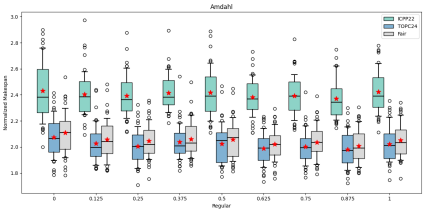
C.2 Boxplot Figures



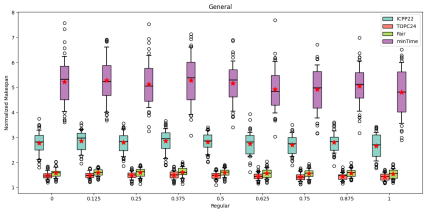
(a) Roofline



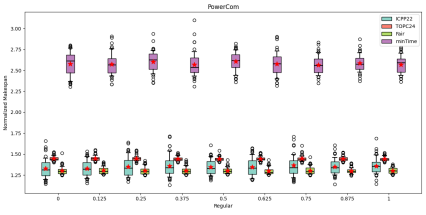
(b) Communication



(c) Amdahl

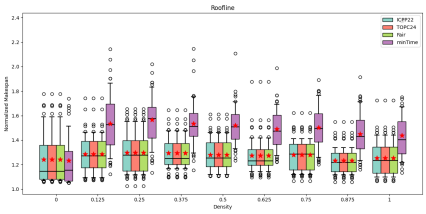


(d) General

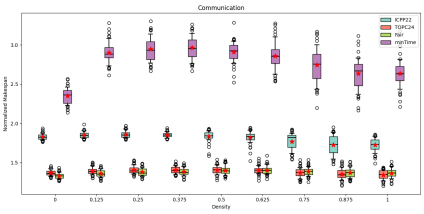


(e) PowerCom

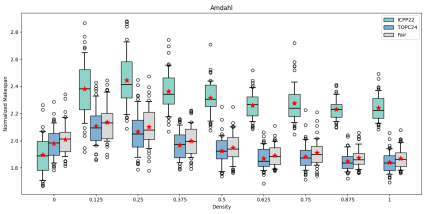
Figure 12: Boxplot Figure for Regular



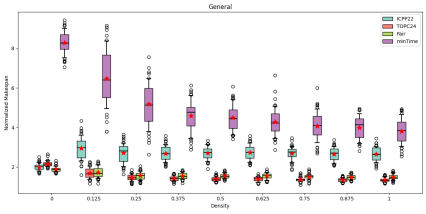
(a) Roofline



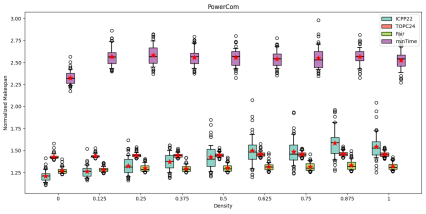
(b) Communication



(c) Amdahl

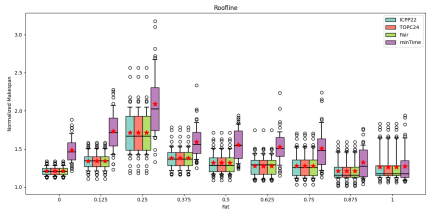


(d) General

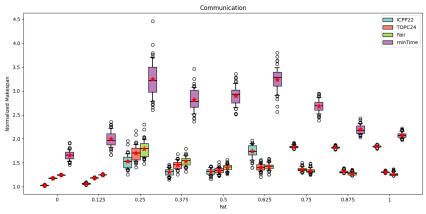


(e) PowerCom

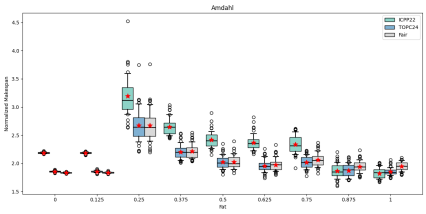
Figure 13: Boxplot Figure for Density



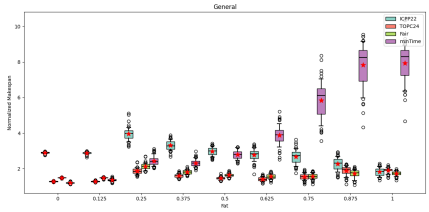
(a) Roofline



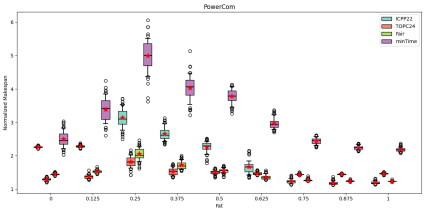
(b) Communication



(c) Amdahl

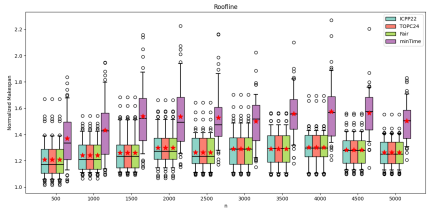


(d) General

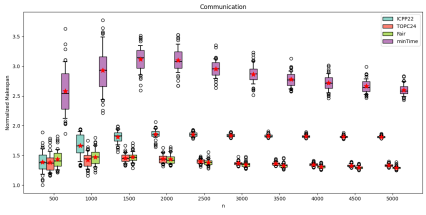


(e) PowerCom

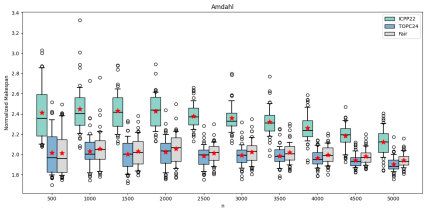
Figure 14: Boxplot Figure for Fat



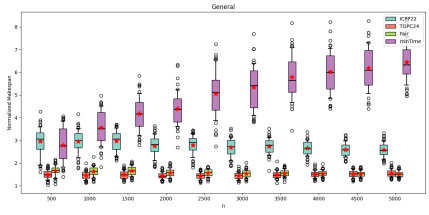
(a) Roofline



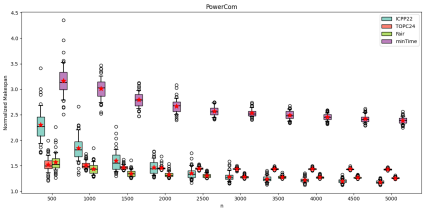
(b) Communication



(c) Amdahl

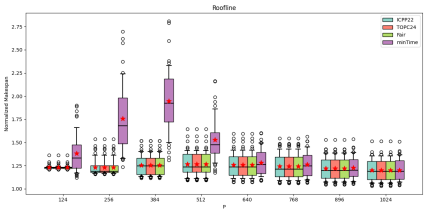


(d) General

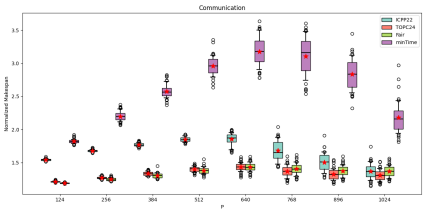


(e) PowerCom

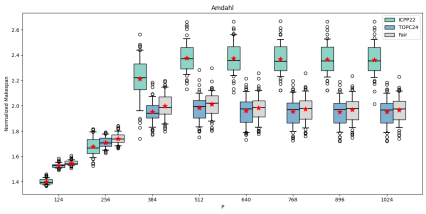
Figure 15: Boxplot Figure for n



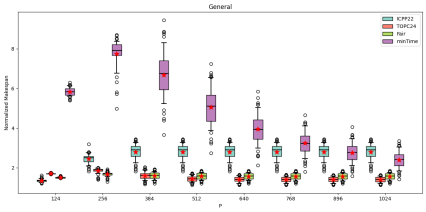
(a) Roofline



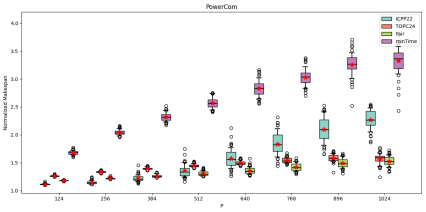
(b) Communication



(c) Amdahl

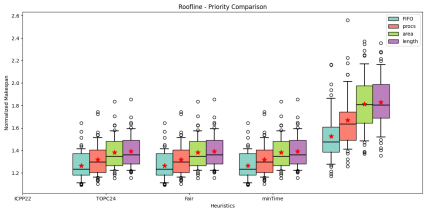


(d) General

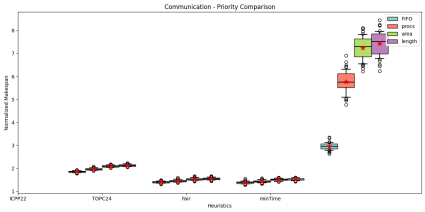


(e) PowerCom

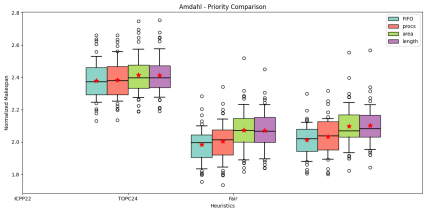
Figure 16: Boxplot Figure for P



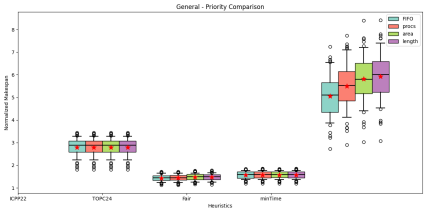
(a) Roofline



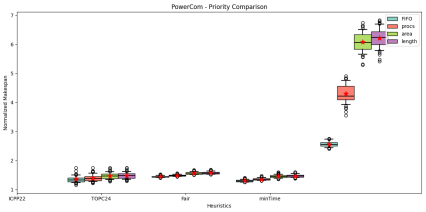
(b) Communication



(c) Amdahl

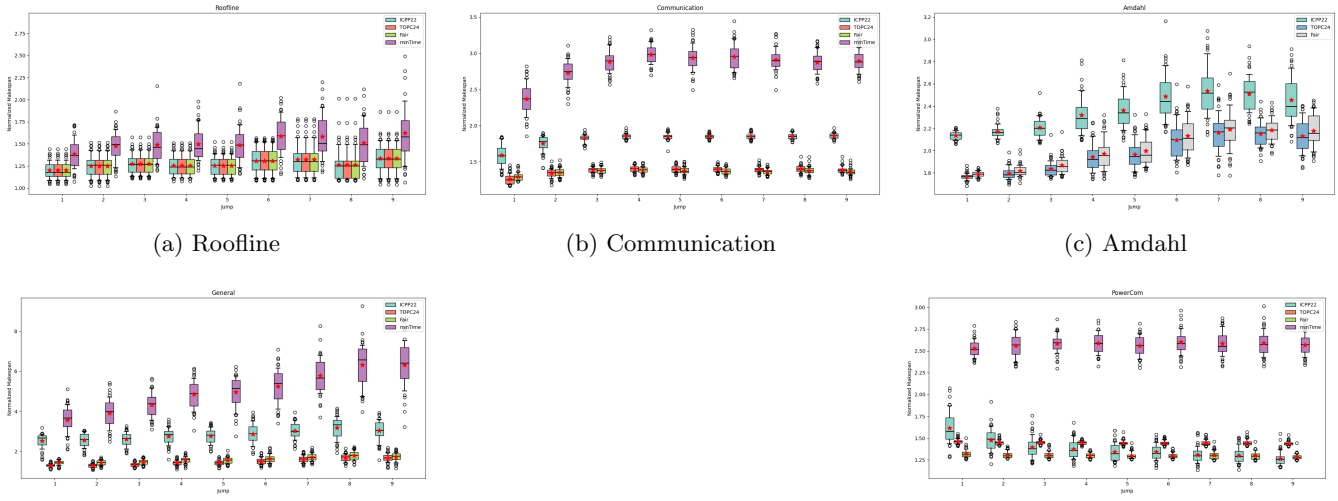


(d) General



(e) PowerCom

Figure 17: Boxplot Figure for Priority



(a) Roofline (b) Communication (c) Amdahl

(d) General (e) PowerCom

Figure 18: Boxplot Figure for Jump

C.3 Average Values

Table 3: Average Values for Each Model and Heuristic

Model	ICPP22	TOPC24	Fair	minTime	minArea
Roofline	1.28	1.28	1.28	1.52	11.28
Communication	1.75	1.38	1.38	2.97	10.22
Amdahl	2.32	1.98	2.01	21.31	7.88
General	2.74	1.50	1.60	4.86	5.37
PowerCom	1.52	1.46	1.35	2.84	12.76
Average	1.92	1.52	1.52	6.70	9.50

C.4 Maximum Values

Table 4: Maximum Values for Each Model and Heuristic

Model	ICPP22	TOPC24	Fair	minTime	minArea
Roofline	2.56	2.56	2.56	3.18	117.75
Communication	2.22	2.17	2.30	8.45	34.30
Amdahl	4.52	3.75	3.76	33.26	23.45
General	5.11	2.64	2.70	10.34	14.76
PowerCom	3.71	2.16	2.46	6.83	69.98
Maximum	5.11	3.75	3.76	33.26	117.75

References

[1] Kunal Agrawal, Charles E. Leiserson, and Jim Sukha. Executing task graphs using work-stealing. In *IPDPS*, pages 1–12, 2010.

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

[3] Anne Benoit, Valentin Le Fèvre, Lucas Perotin, Padma Raghavan, Yves Robert, and Hongyang Sun. Resilient scheduling of moldable jobs on failure-prone platforms. In *IEEE Cluster*, 2020.

[4] Anne Benoit, Valentin Le Fèvre, Lucas Perotin, Padma Raghavan, Yves Robert, and Hongyang Sun. Resilient scheduling of moldable parallel jobs to cope with silent errors. *IEEE Transactions on Computers*, 71(07):1696–1710, 2022.

[5] Louis-Claude Canon, Loris Marchal, Bertrand Simon, and Frédéric Vivien. Online scheduling of task graphs on heterogeneous platforms. *IEEE Trans. Parallel Distributed Syst.*, 31(3):721–732, 2020.

[6] Chi-Yeh Chen and Chih-Ping Chu. A 3.42-approximation algorithm for scheduling malleable tasks under precedence constraints. *IEEE Trans. Parallel Distrib. Syst.*, 24(8):1479–1488, 2013.

[7] Richard A. Dutton and Weizhen Mao. Online scheduling of malleable parallel jobs. In *PDCS*, pages 136–141, 2007.

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

[9] Anja Feldmann, Ming-Yang Kao, Jiří Sgall, and Shang-Hua Teng. Optimal on-line scheduling of parallel jobs with dependencies. *Journal of Combinatorial Optimization*, 1(4):393–411, 1998.

[10] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.

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

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

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

[14] Klaus Jansen and Hu Zhang. Scheduling malleable tasks with precedence constraints. In *SPAA*, page 86–95, 2005.

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

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

[17] Theodore Johnson, Timothy A. Davis, and Steven M. Hadfield. A concurrent dynamic task graph. *Parallel Computing*, 22(2):327–333, 1996.

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

[19] Renaud Lepère, Denis Trystram, and Gerhard J. Woeginger. Approximation algorithms for scheduling malleable tasks under precedence constraints. In *ESA*, pages 146–157, 2001.

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

[21] John Turek, Joel L. Wolf, and Philip S. Yu. Approximate algorithms scheduling parallelizable tasks. In *SPAA*, 1992.

[22] Qingzhou Wang and Kam Hoi Cheng. A heuristic of scheduling parallel tasks and its analysis. *SIAM J. Comput.*, 21(2):281–294, 1992.

[23] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An insightful visual performance model for multicore architectures. *Commun. ACM*, 52(4):65–76, 2009.

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