Online Scheduling of Moldable Task Graphs under Common Speedup Models

Anne Benoit[1] Lucas Perotin[1] Yves Robert[1,2]
Hongyang Sun (Speaker)[3]

[1]École Normale Supérieure de Lyon, France

[2]University of Tennessee Knoxville, USA

[3]University of Kansas, USA

ENS DE LYON

THE UNIVERSITY OF TENNESSEE KNOXVILLE

KU THE UNIVERSITY OF KANSAS

ICPP'22 Best Paper @ ICPP'23
Salt Lake City, Utah, USA, August 8, 2023

INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING

Scheduling Problems

Taxonomy of scheduling problems:

- **Offline Scheduling vs. Online Scheduling**
  - Offline: All tasks are known in advance (NP-hard problems)
  - Online: Tasks are released on the fly (over time or one-by-one)

- **Scheduling Independent Tasks vs. Task Graphs**
  - Independent tasks: There are no dependencies among tasks
  - Task graphs: Tasks have dependencies in the form of a directed acyclic graph (DAG)

Scheduling Problems

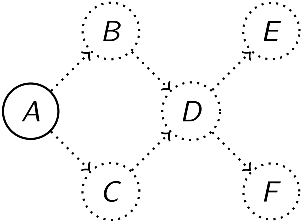
Taxonomy of scheduling problems:

- **Offline Scheduling vs. Online Scheduling**
  - Offline: All tasks are known in advance (NP-hard problems)
  - Online: Tasks are released on the fly (over time or one-by-one)
- **Scheduling Independent Tasks vs. Task Graphs**
  - Independent tasks: There are no dependencies among tasks
  - Task graphs: Tasks have dependencies in the form of a directed acyclic graph (DAG)

In this work, we focus on **online scheduling of task graphs**

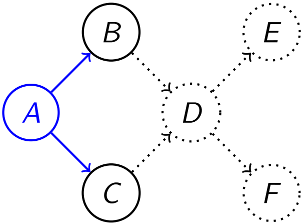
- A task is not known until all predecessors are completed
- Has applications in dynamic workflow scheduling

Example



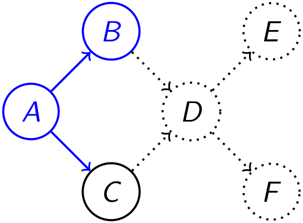
- At first, only task A is known, and others are unknown yet

Example



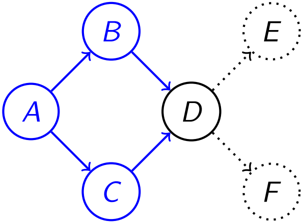
- When task A is done, the scheduler discovers tasks B and C

Example



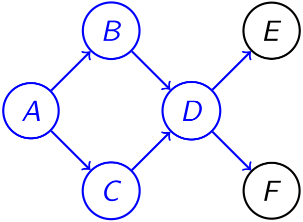
- When task B is done, task D is still not known yet

Example



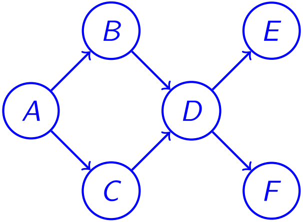
- Only when task C is also done, task D becomes known

Example



- Finally, when task D is done, tasks E and F are discovered

Example



- Tasks E and F are then processed to complete whole graph

Parallel Tasks

Taxonomy of parallel tasks:

- **Rigid tasks**: Processor allocation is fixed
- **Moldable tasks**: Processor allocation is decided by the system but cannot be changed once task starts running
- **Malleable tasks**: Processor allocation can be dynamically changed during runtime

Parallel Tasks

Taxonomy of parallel tasks:

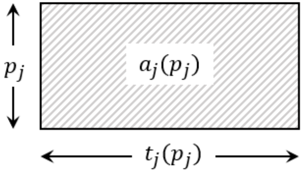
- **Rigid tasks**: Processor allocation is fixed
- **Moldable tasks**: Processor allocation is decided by the system but cannot be changed once task starts running
- **Malleable tasks**: Processor allocation can be dynamically changed during runtime

In this work, we focus on **moldable tasks**

- Easily adapt to amount of available resources (contrarily to rigid tasks)
- Easy to design and implement (contrarily to malleable tasks)

Scheduling Model

- A graph of n moldable tasks. Each task only becomes known when all of its predecessors are completed (i.e., online)
- P identical processors to process the tasks
- For each task j:
  - Execution time $t_j(p_j)$ depends on number of processors p_j allocated to it, and this function also becomes known when the task is discovered
  - Area is $a_j(p_j) = p_j \times t_j(p_j)$



Speedup Models

We mainly focus on a general speedup model:

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

which contains several common models as special cases

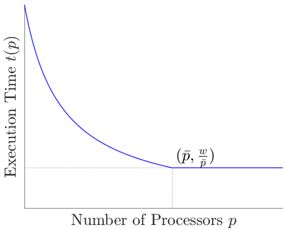
Speedup Models

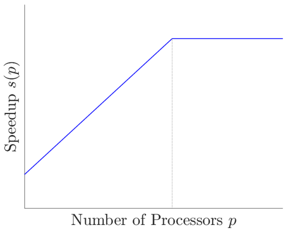
We mainly focus on a general speedup model:

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

which contains several common models as special cases

- Roofline model: $t_j(p_j) = \frac{w_j}{\min(p_j, \bar{p}_j)}$
  where $\bar{p}_j$ is maximum degree of parallelism





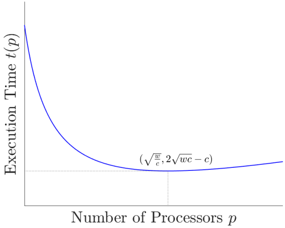
Speedup Models

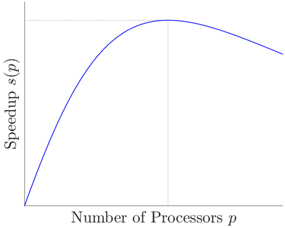
We mainly focus on a general speedup model:

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

which contains several common models as special cases

- Communication model: $t_j(p_j) = \frac{w_j}{p_j} + (p_j - 1)c_j$
  where c_j is communication overhead





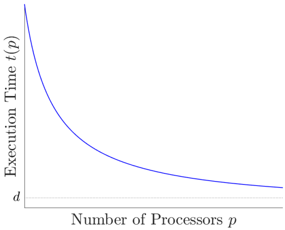
Speedup Models

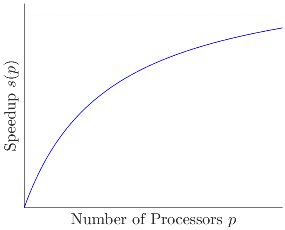
We mainly focus on a general speedup model:

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

which contains several common models as special cases

- Amdahl's model: $t_j(p_j) = \frac{w_j}{p_j} + d_j$
  where d_j is inherently sequential work





Speedup Models

We mainly focus on a general speedup model:

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

which contains several common models as special cases

- Additionally, we consider the arbitrary model, where $t_j(p_j)$ can be an arbitrary function of p_j

Scheduling Objective

Find an online moldable schedule (i.e., processor allocation p_j and starting time s_j for each task j):

- minimizes makespan: $T = \max_j(s_j + t_j(p_j))$
- subject to processor constraint: $\sum_{j \text{ active at time } t} p_j \leq P, \forall t$
- subject to precedence constraint: $j_1 \rightarrow j_2 \;\Rightarrow\; s_{j_2} \geq s_{j_1} + t_{j_1}$

Competitive Ratio:

An online algorithm is said to be *r-competitive* if its makespan T for any task graph satisfies:

$$\frac{T}{T_{\mathrm{OPT}}} \leq r$$

where T_{OPT} is the optimal offline makespan for the same graph

Our Main Results

- New online algorithm with almost tight competitive ratios for several common speedup models

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

- Negative result for the arbitrary speedup model:
  Any deterministic online algorithm is $\Omega(\ln(D))$-competitive, where D is the length of the longest path in the graph

(Closely) Related Work

- **Feldmann, Kao, Sgall, Teng (1998)**:
  - Online scheduling of moldable task graphs in "*non-clairvoyant*" setting (i.e., work of a task is unknown until completion)
  - A 2.62-competitive algorithm for roofline model

(Closely) Related Work

- **Feldmann, Kao, Sgall, Teng (1998)**:
  - Online scheduling of moldable task graphs in "*non-clairvoyant*" setting (i.e., work of a task is unknown until completion)
  - A 2.62-competitive algorithm for roofline model
- **Lepère, Trystram, Woeginger (2001)**:
  - Offline scheduling of moldable task graphs
  - A 5.24-approximation algorithm for monotonic model (i.e., $t(p)$ is non-increasing and $a(p)$ is non-decreasing with p)

(Closely) Related Work

- **Feldmann, Kao, Sgall, Teng (1998)**:
  - Online scheduling of moldable task graphs in "*non-clairvoyant*" setting (i.e., work of a task is unknown until completion)
  - A 2.62-competitive algorithm for roofline model
- **Lepère, Trystram, Woeginger (2001)**:
  - Offline scheduling of moldable task graphs
  - A 5.24-approximation algorithm for monotonic model (i.e., $t(p)$ is non-increasing and $a(p)$ is non-decreasing with p)
- **Havill, Mao (2008)**:
  - Online scheduling of independent moldable tasks that arrive over time
  - A 4-competitive algorithm for communication model

(Closely) Related Work

- **Feldmann, Kao, Sgall, Teng (1998)**:
  - Online scheduling of moldable task graphs in "*non-clairvoyant*" setting (i.e., work of a task is unknown until completion)
  - A 2.62-competitive algorithm for roofline model
- **Lepère, Trystram, Woeginger (2001)**:
  - Offline scheduling of moldable task graphs
  - A 5.24-approximation algorithm for monotonic model (i.e., $t(p)$ is non-increasing and $a(p)$ is non-decreasing with p)
- **Havill, Mao (2008)**:
  - Online scheduling of independent moldable tasks that arrive over time
  - A 4-competitive algorithm for communication model
- **Ye, Chen, Zhang (2018)**:
  - Online scheduling of independent moldable tasks in "*one-by-one*" setting (i.e., tasks are released sequentially and each task must be scheduled immediately upon release)
  - A 16.74-competitive algorithm for arbitrary model

Outline

Lower Bound on Makespan

For each task j:

- Minimum area: $a_j^{\min} = \min_p a_j(p)$
- Minimum execution time: $t_j^{\min} = \min_p t_j(p)$

Lower Bound on Makespan

For each task j:

- Minimum area: $a_j^{\min} = \min_p a_j(p)$
- Minimum execution time: $t_j^{\min} = \min_p t_j(p)$

For task graph:

- Minimum total area: $A_{\min} = \sum_{j=1}^{n} a_j^{\min}$
- Minimum critical-path length: $C_{\min} = \max_f \sum_{j \in f} t_j^{\min}$

Lower Bound on Makespan

For each task j:

- Minimum area: $a_j^{\min} = \min_p a_j(p)$
- Minimum execution time: $t_j^{\min} = \min_p t_j(p)$

For task graph:

- Minimum total area: $A_{\min} = \sum_{j=1}^{n} a_j^{\min}$
- Minimum critical-path length: $C_{\min} = \max_f \sum_{j \in f} t_j^{\min}$

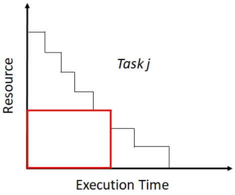
Proposition

The ***optimal makespan*** *satisfies:*

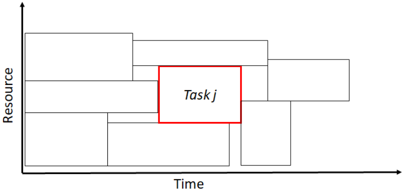
$$T_{\text{OPT}} \geq \max\left(\frac{A_{\min}}{P}, C_{\min}\right)$$

Two-Phase Approach [Turek et al. '92]

- **Phase 1**: Determine a resource allocation for each task once it becomes available



- **Phase 2**: Construct a schedule based on resource allocations of the available tasks



Phase 1: (Local) Resource Allocation

- Step (1): Initial allocation [Benoit et al. 20]
  Find an allocation $p_j \in [1, P]$ from the following problem:

$$\min_{p} \ \alpha(p) \triangleq \frac{a_j(p)}{a_j^{\min}}$$

$$\text{s.t. } \beta(p) \triangleq \frac{t_j(p)}{t_j^{\min}} \leq \frac{1-2\mu}{\mu(1-\mu)}$$

$\Rightarrow$ Allocate resource locally for each task: minimize area subject to a time constraint

Phase 1: (Local) Resource Allocation

- Step (1): Initial allocation [Benoit et al. 20]
  Find an allocation $p_j \in [1, P]$ from the following problem:

$$\min_p \ \alpha(p) \triangleq \frac{a_j(p)}{a_j^{\min}}$$

$$\text{s.t. } \beta(p) \triangleq \frac{t_j(p)}{t_j^{\min}} \leq \frac{1-2\mu}{\mu(1-\mu)}$$

  $\Rightarrow$ Allocate resource locally for each task: minimize area subject to a time constraint

- Step (2): Adjusted allocation [Lepère et al. 01]
  If $p_j > \lceil \mu P \rceil$ **then** $p'_j \leftarrow \lceil \mu P \rceil$ **else** $p'_j \leftarrow p_j$
  $\Rightarrow$ Reduce high allocation to increase overall resource utilization: choice of $\mu \in (0, 0.5)$ depends on speedup model

Phase 2: (Online) List Scheduling

- Insert a task in a list (i.e., waiting queue) as it becomes available
- Whenever an existing task completes, which releases resources, scan the list and schedule each task that fits

Note: when a task becomes available, it is not required to be immediately scheduled (one-by-one model)

Outline

(1) Local Analysis

Can we say something about each individual task?

Proposition

For a given speedup model M, there exists an (α, β) pair and an initial resource allocation p_j for any task j such that:

$$a_j(p_j) \leq \alpha \cdot a_j^{\min}$$
$$t_j(p_j) \leq \beta \cdot t_j^{\min}$$

(1) Local Analysis

Can we say something about each individual task?

Proposition

For a given speedup model M, there exists an (α, β) pair and an initial resource allocation p_j for any task j such that:

$$a_j(p_j) \leq \alpha \cdot a_j^{\min}$$
$$t_j(p_j) \leq \beta \cdot t_j^{\min}$$

These local bounds will carry over to the global analysis!

$$\sum_{j \in J} a_j(p_j) \leq \alpha \cdot \sum_{j \in J} a_j^{\min}$$
$$\sum_{j \in f} t_j(p_j) \leq \beta \cdot \sum_{j \in f} t_j^{\min}$$

(1) Local Analysis

Can we say something about each individual task?

Proposition

For a given speedup model M, there exists an (α, β) *pair and an initial resource allocation* p_j *for any task* j *such that:*

$$a_j(p_j) \leq \alpha \cdot a_j^{\min}$$
$$t_j(p_j) \leq \beta \cdot t_j^{\min}$$

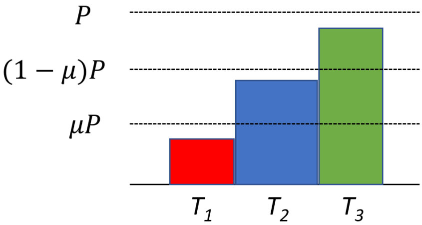
These local bounds will carry over to the global analysis!

$$\sum_{j \in J} a_j(p_j) \leq \alpha \cdot \sum_{j \in J} a_j^{\min} \leq \alpha \cdot A_{\min}$$
$$\sum_{j \in f} t_j(p_j) \leq \beta \cdot \sum_{j \in f} t_j^{\min} \leq \beta \cdot C_{\min}$$

(2) Global Analysis

Total makespan interval $[0, T]$ divided in three sets [Lepère et al. 01]:

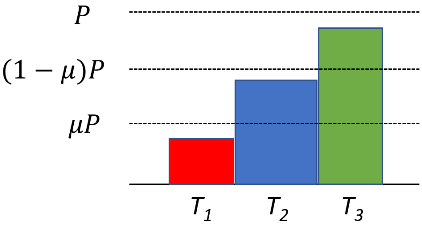
- T_1: Less than μP processors are used.
- T_2: Between μP and $(1-\mu)P$ processors are used
- T_3: More than $(1-\mu)P$ processor are used



(2) Global Analysis

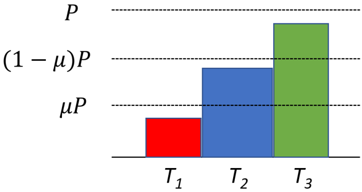
Total makespan interval $[0, T]$ divided in three sets [Lepère et al. 01]:

- T_1: Less than μP processors are used.
- T_2: Between μP and $(1-\mu)P$ processors are used
- T_3: More than $(1-\mu)P$ processor are used



T_1 and T_2 can be charged to the critical-path length
T_2 and T_3 can be charged to the total area

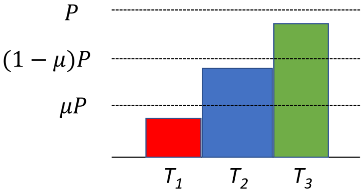
(3) Combining Two Analyses



Critical-path bound: $\frac{T_1}{\beta} + \mu T_2 \leq C_{\min}$

Total area bound: $\mu T_2 + (1 - \mu) T_3 \leq \frac{\alpha \cdot A_{\min}}{P}$

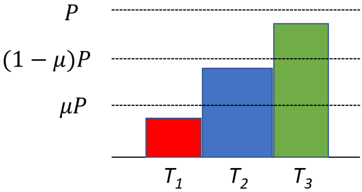
(3) Combining Two Analyses



Critical-path bound: $\frac{T_1}{\beta} + \mu T_2 \leq C_{\min} \leq T_{\text{OPT}}$

Total area bound: $\mu T_2 + (1-\mu) T_3 \leq \frac{\alpha \cdot A_{\min}}{P} \leq \alpha \cdot T_{\text{OPT}}$

(3) Combining Two Analyses



Critical-path bound: $\dfrac{T_1}{\beta} + \mu T_2 \leq C_{\min} \leq T_{\text{OPT}}$

Total area bound: $\mu T_2 + (1-\mu) T_3 \leq \dfrac{\alpha \cdot A_{\min}}{P} \leq \alpha \cdot T_{\text{OPT}}$

Proposition

Combining the two bounds with $T = T_1 + T_2 + T_3$, we get:

$$\frac{T}{T_{\text{OPT}}} \leq \frac{\mu\alpha + 1 - 2\mu}{\mu(1-\mu)} \quad \textit{subject to} \quad \beta \leq \frac{1-2\mu}{\mu(1-\mu)}$$

Final Results

Proposition

Combining the two bounds with $T = T_1 + T_2 + T_3$, we get:

$$\frac{T}{T_{\mathrm{OPT}}} \leq \frac{\mu\alpha + 1 - 2\mu}{\mu(1-\mu)} \quad \textit{subject to} \quad \beta \leq \frac{1-2\mu}{\mu(1-\mu)}$$

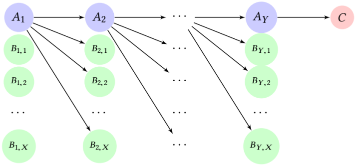
Optimization procedure for a given speedup model:

1. Find an upper bound for α as a function of μ
2. Find μ minimizing the ratio subject to β constraint

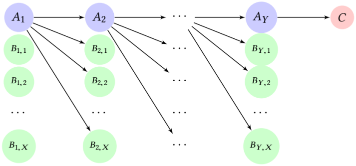
Model	Roofline	Comm.	Amdahl	General
Choice of μ	0.382	0.324	0.271	0.211
Upper bound	2.62	3.61	4.74	5.72

Outline

Instance for Common Speedup Models

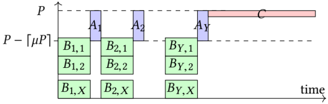


Instance for Common Speedup Models

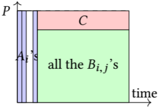


Task parameters are chosen so that:

- **For online algorithm** (a): Barely impossible to process a full layer in parallel
- **For optimal algorithm** (b): First process all A's and then B's and C's in parallel

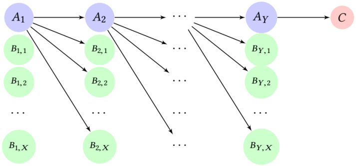


(a)



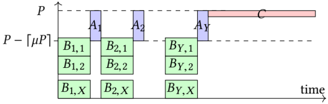
(b)

Instance for Common Speedup Models

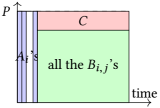


Task parameters are chosen so that:

- **For online algorithm** (a): Barely impossible to process a full layer in parallel
- **For optimal algorithm** (b): First process all A's and then B's and C's in parallel



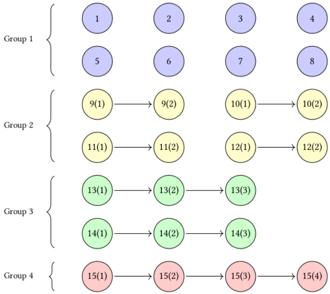
(a)



(b)

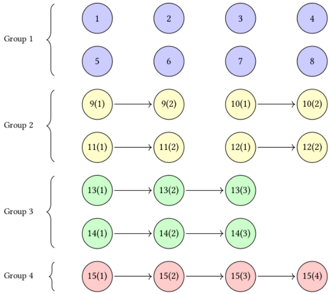
Model	Roofline	Comm.	Amdahl	General
Upper bound	2.62	3.61	4.74	5.72
Lower bound	2.61	3.51	4.73	5.25

Instance for Arbitrary Speedup Model

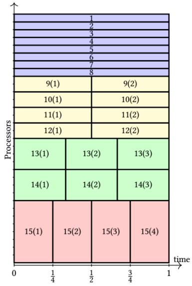


- $D = 2^\ell$ groups of identical tasks with execution time function $t(p) = \frac{1}{\lg(p)+1}$

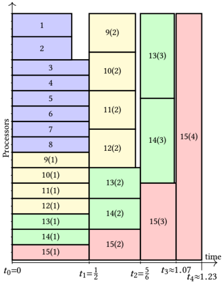
Instance for Arbitrary Speedup Model



- $D = 2^\ell$ groups of identical tasks with execution time function $t(p) = \frac{1}{\lg(p)+1}$
- **For optimal algorithm** (a): 2^{i-1} processors for tasks in group $i \Rightarrow$ makespan of 1
- **For online algorithm** (b): same processors for all tasks (best online strategy) $\Rightarrow$ makespan of $\Omega(\ln(D))$



(a)



(b)

Outline

Conclusion

- A new algorithm for online scheduling of moldable task graphs
- Almost tight competitive ratios for several common speedup models
- No constant competitive ratio for arbitrary speedup model by any deterministic online algorithm

Future work:

- Consider other speedup models or special task graphs
- Improve the ratios for upper and/or lower bounds
- Experimental evaluation of the algorithm's performance

Latest Results

Model	Roofline	Comm.	Amdahl	General
Old Results	≈ 2.62	≈ 3.61	≈ 4.74	≈ 5.72
New Results[1]	≈ 2.62	≈ 3.39	≈ 4.55	≈ 4.63

with matching lower bounds

- New upper bounds benefit from a tighter (α, β) analysis: worst-case time and area bounds don't happen simultaneously
- New lower bounds also apply to a class of algorithms with deterministic local processor allocation (i.e., stronger)

[1] Lucas Perotin, Hongyang Sun. Improved Online Scheduling of Moldable Task Graphs under Common Speedup Models. 2023. https://arxiv.org/abs/2304.14127