# Malleable task-graph scheduling with a practical speed-up model

Loris Marchal[1] *Bertrand Simon*[1] Oliver Sinnen[2] Frédéric Vivien[1]

1: LIP, CNRS, INRIA, ENS de Lyon and Univ. de Lyon. 2: University of Auckland, NZ.

Contact: bertrand.simon@ens-lyon.fr

## Objectives

Optimize the time performance of multifrontal sparse direct solvers (e.g., MUMPS).

- Computations described by a tree of tasks
- Generalization to Series-Parallel graphs – i.e., $G = T \mid G_1; G_2 \mid G_1 \parallel G_2$

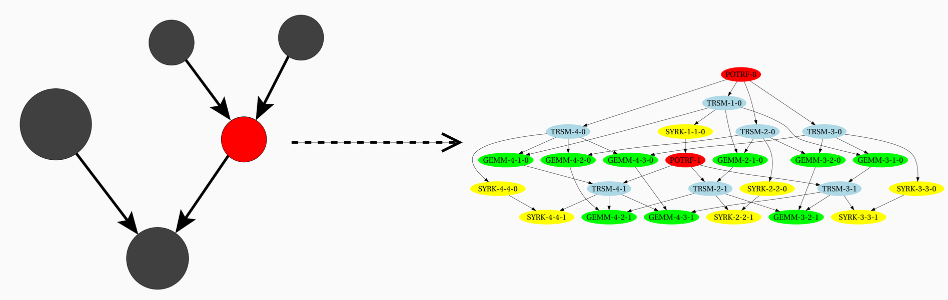
We aim at:

- Guaranteeing widely used algorithms
- Designing better scheduling algorithms

## Why malleable task trees suffice?

**Coarse-grain picture**

- Each task: partial factorization, graph of smaller sub-tasks



**Reason of this abstraction**

- Expand all tasks: high complexity to schedule
- Scheduling trees simpler than general graphs

**Behavior of coarse-grain tasks**

- Parallel and malleable
- Speed-up model → trade-off between:
  - *Accuracy*: fits well the data
  - *Tractability*: guaranteed algorithms

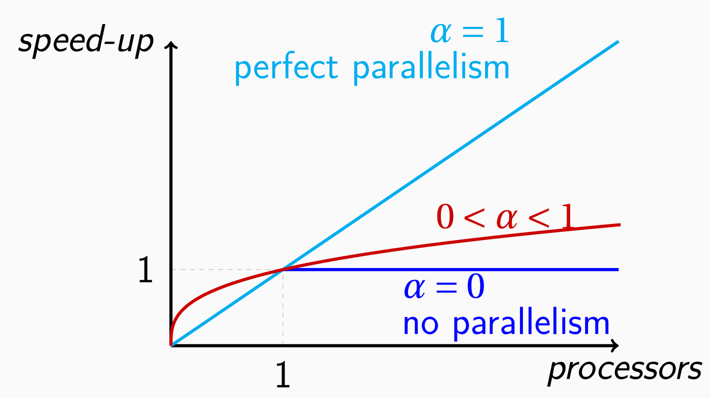
## Previous work: Prasanna & Musicus model

**Focus on two quantities**

$$speed\text{-}up(p) = \frac{time(1\ proc.)}{time(p\ proc.)} \quad \Big| \quad work(p) = p \cdot time(p\ proc.)$$

**Study model:** $speed\text{-}up(p) = p^{\alpha}$

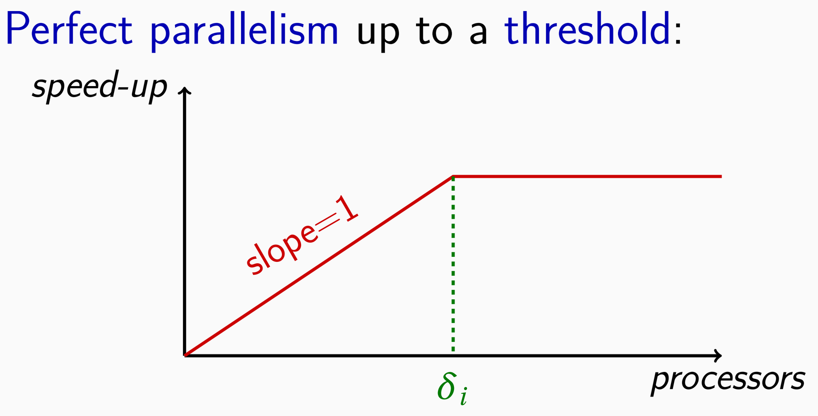
- Average Accuracy 😐
- Rational number of processors 😐
- Optimal algorithms for Series-Parallel graph 🙂
- No guarantees for distributed platforms ☹
- Task finish times complex to compute ☹




## A simple yet practical model

**Parallel malleable tasks**

- Perfect parallelism up to a threshold:




- Total work: $w_i$ — Threshold: $\delta_i$
- Rational allocation for free (McNaughton's wrap-around rule)

## Related work

**Non-increasing speed-up and work**

- Independent tasks: theoretical FPTAS and practical 2-approximations [Jansen 2004, Fan et al. 2012]
- SP-graphs: $\approx 2.6$-approximation [Lepère et al. 2001]. With concave speed-up: $(2+\varepsilon)$-approximation of unspecified complexity [Makarychev et al. 2014]

**Specific speed-up function**

- Same model: 2-approximation [Balmin et al. 2013] named FlowFlex (see experimental setup)
- [Kell et al. 2015]: $time = \frac{w_i}{p} + (p-1)c$; 2-approximation for $p = 3$, open for $p \geq 4$

## NP-Completeness of the problem

**Complexity depending on the model**

- Malleability + perfect parallelism ⇒ P
- Adding thresholds ⇒ NP-complete
- Arguably complex proof [Drozdowski and Kubiak 1999]

**Contribution**

- New NP-completeness proof

## The widely used PropMapping

**Simple allocation for trees or SP-graphs**

- On a series composition $G = (G_1; G_2)$: give all available processors to $G_1$, then to $G_2$
- On $(G_1 \parallel G_2)$: give a constant share to $G_i$, proportional to its weight $w_i$
- Algorithm on graph $G$ with $q$ processors:

PropMapping $(G, q)$

if $G = G_1; G_2; \ldots; G_k$ then
  $\forall i,\ p_i \leftarrow q$

if $G = G_1 \parallel G_2 \parallel \cdots \parallel G_k$ then
  $\forall i,\ p_i = q w_i / \sum w_j$

Call PropMapping $(G_i, p_i)$ for each $G_i$

- Then schedule each task on $p_i$ processors as soon as it is ready

**Notes**

- Moldable schedule (constant allocation)
- Unaware of task thresholds

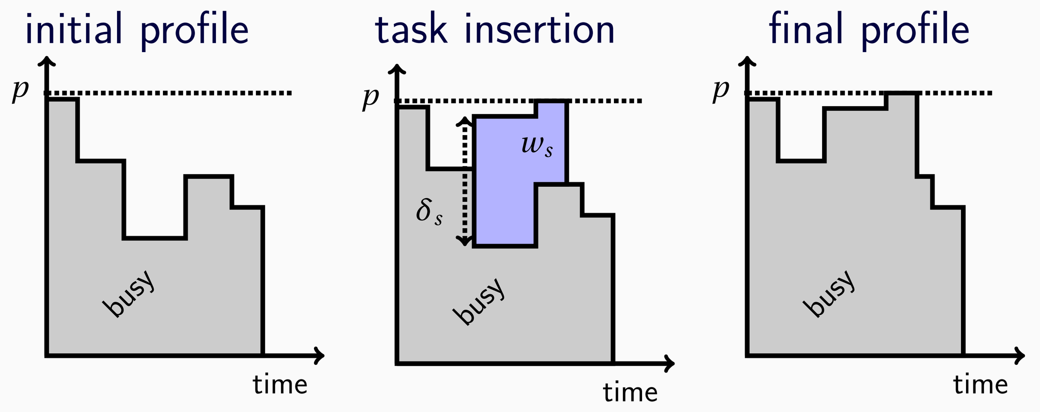
**Theorem:** PropMapping is a 2-approximation.

## A new strategy: Greedy-Filling

**Algorithm**

- Assign priorities to tasks (usually bottom-level)
- Consider free tasks by decreasing priority
- Greedily insert each task in the schedule:
  - Compute the earliest starting time
  - *Pour* task into the available processor space, respecting thresholds

**Illustration**




**Theorem:** Greedy-Filling is a 2-approximation.

## Experimental setup

**Third algorithm for comparison: FlowFlex**

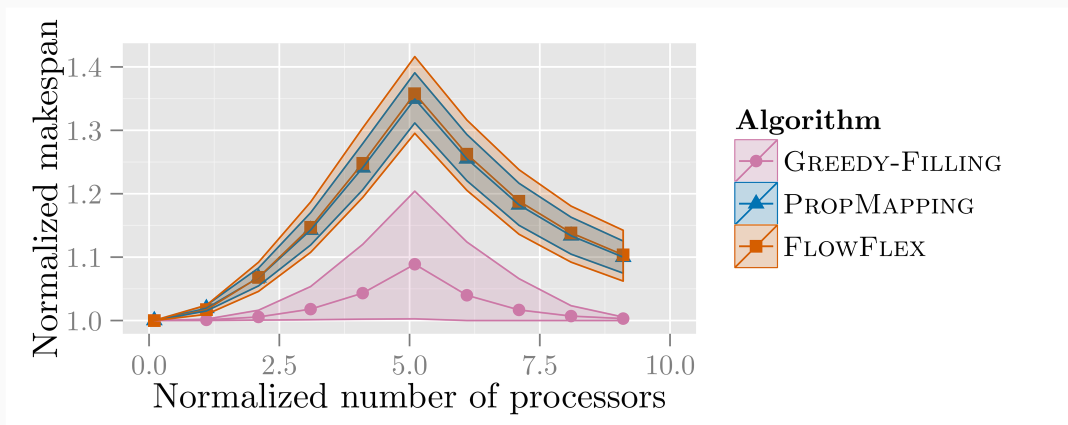
- 2-approximation from [Balmin et al. 2013] to schedule *Malleable Flows of MapReduce Jobs*
- Solve the problem on an infinite number of processors
- Downscale the allocation on intervals when it is needed

**Two datasets**

- Synth: synthetic SP-graphs with $\delta_i = \alpha \times w_i$
- Trees: assembly trees of sparse matrices, $\delta_i = \alpha \times w_i$.

## Validation of Greedy-Filling

**Results on Synth**



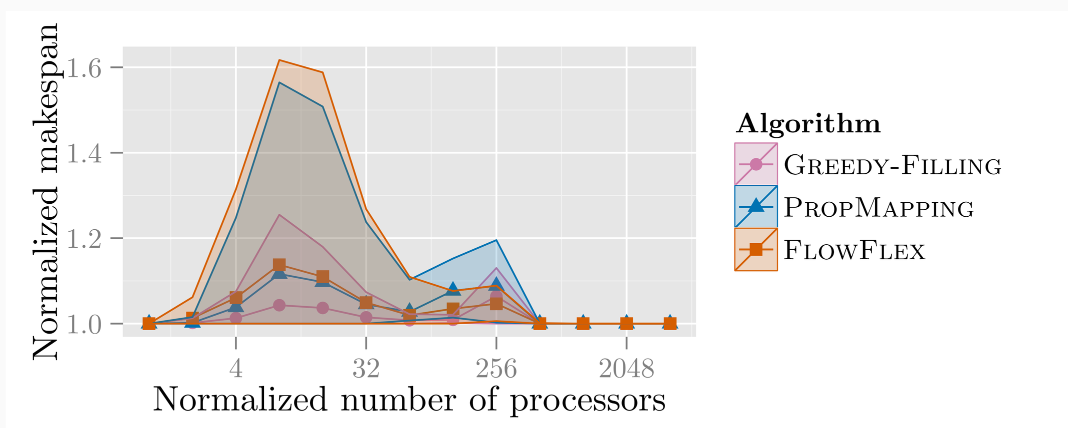

Legend

- Y axis: makespan normalized by the lower bound: $LB = \max(CP,\ W/p)$
- X axis: number of processors normalized by $para = \frac{\text{makespan with all } \delta_i = 1 \text{ and } p = \infty}{\text{makespan with all } \delta_i = 1 \text{ and } p = 1}$

Comments

- Plot: mean + ribbon with 90% of the results
- Small/large number of processors: similar results as the problem is simple
- Greedy-Filling: $\approx 25\%$ of gain, $< 20\%$ from the LB

**Results on Trees**




Comments

- Results shape depends a lot on the matrix
- Here: one matrix with different ordering and amalgamation parameters
- Greedy-Filling is (almost always) better than both others
- Smaller maximum gain (around 15%)

## Conclusion

**On the algorithms**

- PropMapping: does not take advantage of malleability
- FlowFlex: produces gaps that cannot be filled afterwards
- Greedy-Filling: simple, greedy, close to the lower bound

**On the model**

- Simplest model to account for limited parallelism
- Still NP-complete
- Possible to derive theoretical guarantees (2-approximation algorithms)

ENS DE LYON · UNIVERSITÉ DE LYON · SOLHAR · AGENCE NATIONALE DE LA RECHERCHE ANR · THE UNIVERSITY OF AUCKLAND NEW ZEALAND