Scheduling Series-Parallel Graphs of Malleable Tasks

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

1: CNRS, INRIA, ENS Lyon and Univ. Lyon, FR.
2: Univ. Auckland, NZ.

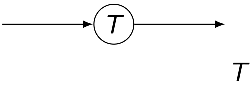
Solhar plenary meeting

December 2nd, 2016

Motivation

Context:

- Optimize the time performance of multifrontal sparse solvers (e.g., MUMPS or QR-MUMPS)
- Computations well described by a tree of tasks
- Generalization to Series-Parallel graphs
- Purpose: find a schedule achieving the shortest makespan



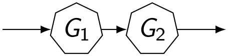
Objectives:

- Provide theoretical guarantees on widely used scheduling algorithms
- Design algorithms with shorter makespan

Motivation

Context:

- Optimize the time performance of multifrontal sparse solvers (e.g., MUMPS or QR-MUMPS)
- Computations well described by a tree of tasks
- Generalization to Series-Parallel graphs
- Purpose: find a schedule achieving the shortest makespan



$G_1; G_2$

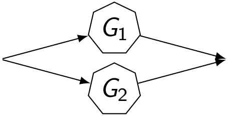
Objectives:

- Provide theoretical guarantees on widely used scheduling algorithms
- Design algorithms with shorter makespan

Motivation

Context:

- Optimize the time performance of multifrontal sparse solvers (e.g., MUMPS or QR-MUMPS)
- Computations well described by a tree of tasks
- Generalization to Series-Parallel graphs
- Purpose: find a schedule achieving the shortest makespan



$G_1 \parallel G_2$

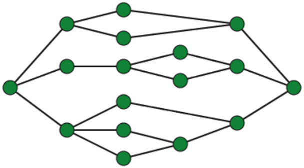
Objectives:

- Provide theoretical guarantees on widely used scheduling algorithms
- Design algorithms with shorter makespan

Motivation

Context:

- Optimize the time performance of multifrontal sparse solvers (e.g., MUMPS or QR-MUMPS)
- Computations well described by a tree of tasks
- Generalization to Series-Parallel graphs
- Purpose: find a schedule achieving the shortest makespan



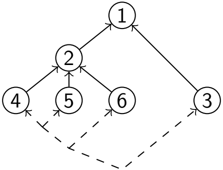
Objectives:

- Provide theoretical guarantees on widely used scheduling algorithms
- Design algorithms with shorter makespan

Motivation

Context:

- Optimize the time performance of multifrontal sparse solvers (e.g., MUMPS or QR-MUMPS)
- Computations well described by a tree of tasks
- Generalization to Series-Parallel graphs
- Purpose: find a schedule achieving the shortest makespan



Objectives:

- Provide theoretical guarantees on widely used scheduling algorithms
- Design algorithms with shorter makespan

Application modeling

Coarse-grain picture: tree of tasks (or SP task graph)

- Each task is itself a parallel task

Behavior of tasks

- parallel and malleable
  (processor allotment can change during task execution)

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

- Speed-up model $\longrightarrow$ trade-off between:
  - Accuracy: fits well the data
  - Tractability: amenable to perf. analysis, guaranteed algorithms

General speed-up models

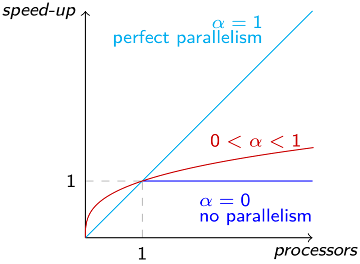
Literature: studies with few assumptions

Non-increasing speed-up and non-decreasing work

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

Prasanna & Musicus' model [Prasanna and Musicus 1996]

- $\mathit{speed\text{-}up}(p) = p^\alpha$, with $0 < \alpha \le 1$



- Task T_i of weight w_i

  Processing time of T_i: $= \arg\min_C \left\{ \int_0^C p_i(t)^\alpha \, \mathrm{d}t \ge w_i \right\}$

Results for Prasanna & Musicus' model

Theorem (Prasanna & Musicus)

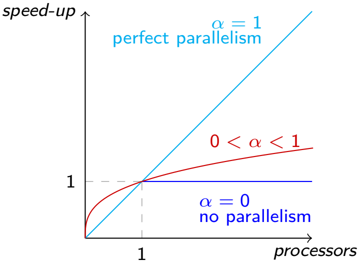
In optimal schedules, at any parallel node $G_1 \parallel G_2$, the ratio of processors given to each branch is constant.

Corollary

- *$G \approx$ equivalent task T_G of weight $\mathcal{W}_G$ defined by:*
  - $\mathcal{W}_{T_i} = L_i$
  - $\mathcal{W}_{G_1\,;\,G_2} = \mathcal{W}_{G_1} + \mathcal{W}_{G_2}$
  - $\mathcal{W}_{G_1 \parallel G_2} = \left(\mathcal{W}_{G_1}^{1/\alpha} + \mathcal{W}_{G_2}^{1/\alpha}\right)^{\alpha}$
- *The (unique) optimal schedule $\mathcal{S}_{\mathrm{PM}}$ can be computed in polynomial time.*

Previous work (Europar 2015, with Abdou Guermouche)

Prasanna & Musicus model [PM 1996]: $\textit{speed-up}(p) = p^{\alpha}$

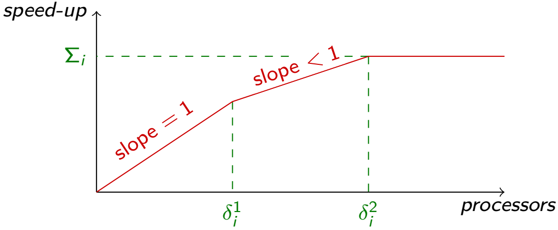


Conclusions:

- Optimal algorithm for SP-graphs ☺
- Average Accuracy
- Rational numbers of processors
- Task finish times complex to compute ☹
- No guarantees for distributed platforms ☹

Simple and reasonable model of a parallel malleable task T_i

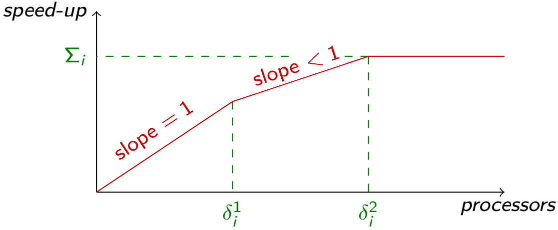
- Perfect then linear then plateau, speedup function s_i:



Today: simpler model

Simple and reasonable model of a parallel malleable task T_i

- Perfect then linear then plateau, speedup function s_i:



Related studies

- $\delta_i^1 = \delta_i^2$: Loris Marchal's talk at last meeting (we refined the model)
  2-approximation [Balmin et al. 13] that we will discuss
- [Kell et al. 2015]: $time = \frac{w_i}{p} + (p-1)c$;
  2-approximation for $p = 3$, open for $p \geq 4$

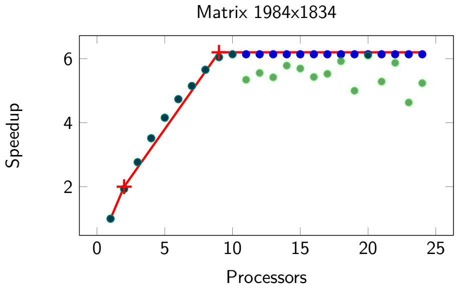
Experimental validation

Setup

- Graph: elimination tree of sparse matrices (task: QR decomposition of a dense rectangular matrix)
- Platform: Miriel node of Plafrim (24 cores)
- Time each task with 1 to 24 cores
  - Plot speedup, correct decrease then compute parameters (δ^1, δ^2, Σ)

Conclusion

- Accurate fitting: median $R^2 = 0.98$ ☺



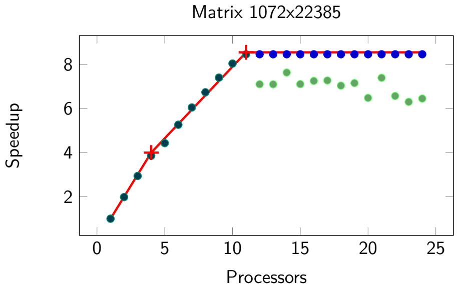
Experimental validation

Setup

- Graph: elimination tree of sparse matrices (task: QR decomposition of a dense rectangular matrix)
- Platform: Miriel node of Plafrim (24 cores)
- Time each task with 1 to 24 cores
  - Plot speedup, correct decrease then compute parameters (δ^1, δ^2, Σ)

Conclusion

- Accurate fitting: median $R^2 = 0.98$ ☺



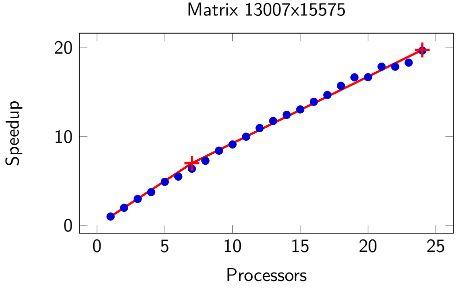
Experimental validation

Setup

- Graph: elimination tree of sparse matrices (task: QR decomposition of a dense rectangular matrix)
- Platform: Miriel node of Plafrim (24 cores)
- Time each task with 1 to 24 cores
  - Plot speedup, correct decrease then compute parameters (δ^1, δ^2, Σ)

Conclusion

- Accurate fitting: median $R^2 = 0.98$ ☺



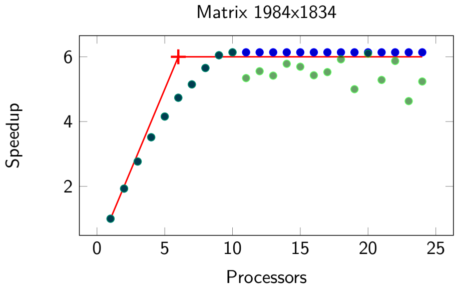
Experimental validation

Setup

- Graph: elimination tree of sparse matrices (task: QR decomposition of a dense rectangular matrix)
- Platform: Miriel node of Plafrim (24 cores)
- Time each task with 1 to 24 cores
  - Plot speedup, correct decrease then compute parameters (δ^1, δ^2, Σ)

Conclusion

- Accurate fitting: median $R^2 = 0.98$ 🙂
- *Single-threshold model: median* $R^2 = 0.90$ ☹

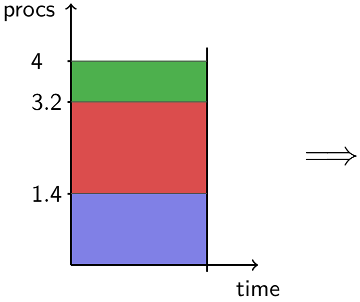


Integer or rational allotments?

Question: should we allow allotments of rational number of cores?

Answer: yes, we can transform such a schedule to integer allotments

Why: piecewise linear speedup ensures McNaughton rule

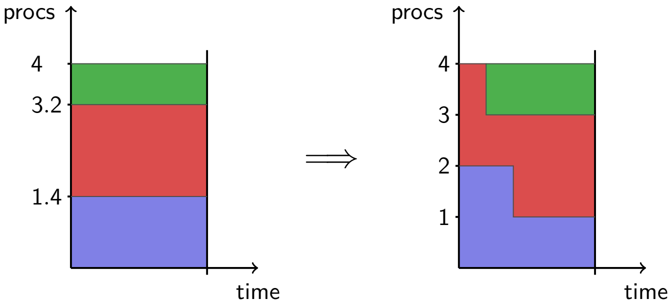


Integer or rational allotments?

Question: should we allow allotments of rational number of cores?

Answer: yes, we can transform such a schedule to integer allotments

Why: piecewise linear speedup ensures McNaughton rule



Outline

1. Analysis of PROPORTIONALMAPPING [Pothen et al. 1993]
2. Design of a greedy strategy
3. Analysis of FLOWFLEX [Balmin et al. 2013]
4. Experimental comparison
5. Conclusion

ProportionalMapping [Pothen et al. 1993]

Description

- Simple allocation for trees or SP-graphs
- On $G_1 \parallel G_2$: constant share to G_i, proportional to its weight W_i

Algorithm 1: ProportionalMapping (graph G , q procs)

1 Define the share allocated to sub-graphs of G:

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

if $G = G_1 \parallel G_2 \parallel \dots G_k$ **then**
  $\forall i,\ p_i \leftarrow qW_i / \sum_j W_j$

2 Call ProportionalMapping (G_i, p_i) for each sub-graph G_i

- Then schedule tasks on p_i processors ASAP

Notes

- Produces a moldable schedule (fixed allocation over time)
- Unaware of task thresholds

Analysis of ProportionalMapping schedules

Theorem

ProportionalMapping *is a* $(1 + r)$*-approximation of the optimal makespan, with* $r = \max_i \left(\delta_i^2 / \Sigma_i\right) \geq 1$.

Proof.

- Consider makespan with perfect speedup: $M_\infty \leq M_{\text{opt}}$
- There is an idle-free path Φ from the entry task to the end
- Split the tasks of Φ in two sets:

Analysis of ProportionalMapping schedules

Theorem

ProportionalMapping *is a* $(1 + r)$*-approximation of the optimal makespan, with* $r = \max_i \left(\delta_i^2 / \Sigma_i\right) \geq 1$.

Proof.

- Consider makespan with perfect speedup: $M_\infty \leq M_{\text{opt}}$
- There is an idle-free path Φ from the entry task to the end
- Split the tasks of Φ in two sets:
  - A = limited by their thresholds: $len(A) \leq$ critical path $\leq M_{\text{opt}}$

Analysis of ProportionalMapping schedules

Theorem

ProportionalMapping *is a* $(1+r)$*-approximation of the optimal makespan, with* $r = \max_i \left(\delta_i^2 / \Sigma_i\right) \geq 1$.

Proof.

- Consider makespan with perfect speedup: $M_\infty \leq M_{\text{opt}}$
- There is an idle-free path Φ from the entry task to the end
- Split the tasks of Φ in two sets:
  - $A =$ limited by their thresholds: $len(A) \leq$ critical path $\leq M_{\text{opt}}$
  - $B =$ limited by the allocation:

$$len(B) = \sum_{i \in B} \frac{w_i}{s_i(p_i)} \quad \text{and} \quad M_\infty \geq \sum_{i \in B} \frac{w_i}{p_i} \quad \text{so} \quad len(B) \leq r M_\infty$$

Analysis of ProportionalMapping schedules

Theorem

ProportionalMapping *is a* $(1+r)$*-approximation of the optimal makespan, with* $r = \max_i \left(\delta_i^2 / \Sigma_i\right) \geq 1$*.*

Proof.

- Consider makespan with perfect speedup: $M_\infty \leq M_{\text{opt}}$
- There is an idle-free path Φ from the entry task to the end
- Split the tasks of Φ in two sets:
  - $A =$ limited by their thresholds: $len(A) \leq$ critical path $\leq M_{\text{opt}}$
  - $B =$ limited by the allocation:

$$len(B) = \sum_{i \in B} \frac{w_i}{s_i(p_i)} \quad \text{and} \quad M_\infty \geq \sum_{i \in B} \frac{w_i}{p_i} \quad \text{so} \quad len(B) \leq rM_\infty$$

- Finally, $M = len(\Phi) = len(A) + len(B) \leq (1+r)M_{\text{opt}}$ □

Optimization of ProportionalMapping

Issue

- Imperfect speedup: tasks do not finish simultaneously
- Idle processors: could reallocate them

Optimization of PROPORTIONALMAPPING

Issue

- Imperfect speedup: tasks do not finish simultaneously
- Idle processors: could reallocate them

Design of PROPMAPEXT from PROPORTIONALMAPPING

- When a task terminates: reallocate its processors to the *sibling* tasks
- Reallocation is done proportionally to the remaining critical path
- PROPMAPEXTTHRESH: idem but never exceeds δ^2

Optimization of ProportionalMapping

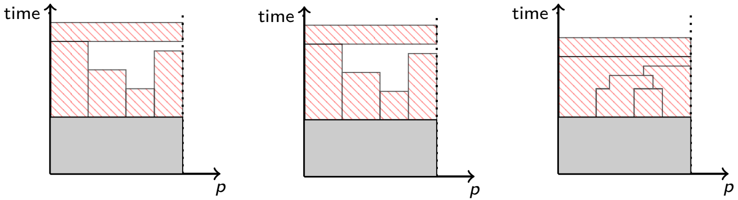
Issue

- Imperfect speedup: tasks do not finish simultaneously
- Idle processors: could reallocate them

Design of PropMapExt from ProportionalMapping

- When a task terminates: reallocate its processors to the *sibling* tasks
- Reallocation is done proportionally to the remaining critical path
- PropMapExtThresh: idem but never exceeds δ^2

PropMapping: Rebalancing: PropMapExt:



Optimization of ProportionalMapping

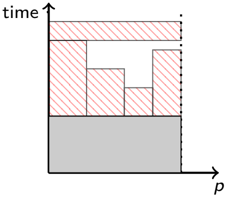
Issue

- Imperfect speedup: tasks do not finish simultaneously
- Idle processors: could reallocate them

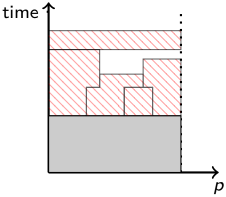
Design of PropMapExt from ProportionalMapping

- When a task terminates: reallocate its processors to the *sibling* tasks
- Reallocation is done proportionally to the remaining critical path
- PropMapExtThresh: idem but never exceeds δ^2

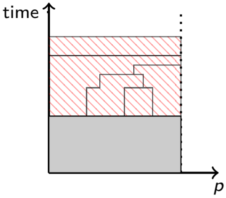
PropMapping:



Rebalancing:



PropMapExt:



Optimization of PROPORTIONALMAPPING

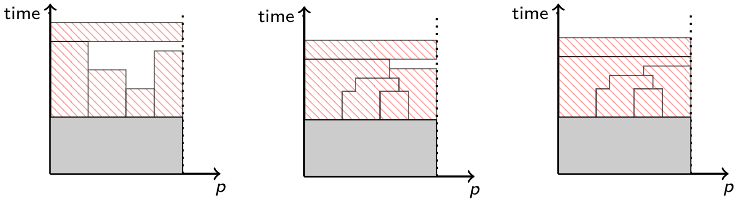
Issue

- Imperfect speedup: tasks do not finish simultaneously
- Idle processors: could reallocate them

Design of PROPMAPEXT from PROPORTIONALMAPPING

- When a task terminates: reallocate its processors to the *sibling* tasks
- Reallocation is done proportionally to the remaining critical path
- PROPMAPEXTTHRESH: idem but never exceeds δ^2

PROPMAPPING: Rebalancing: PROPMAPEXT:



Optimization of PROPORTIONALMAPPING

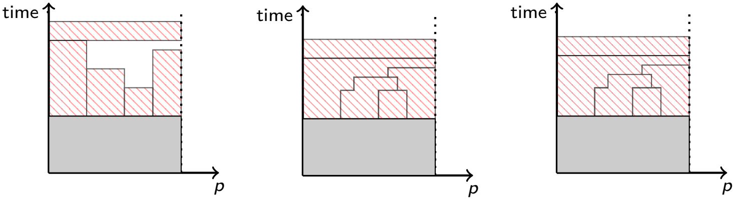
Issue

- Imperfect speedup: tasks do not finish simultaneously
- Idle processors: could reallocate them

Design of PROPMAPEXT from PROPORTIONALMAPPING

- When a task terminates: reallocate its processors to the *sibling* tasks
- Reallocation is done proportionally to the remaining critical path
- PROPMAPEXTTHRESH: idem but never exceeds δ^2

PROPMAPPING: Rebalancing: PROPMAPEXT:



Outline

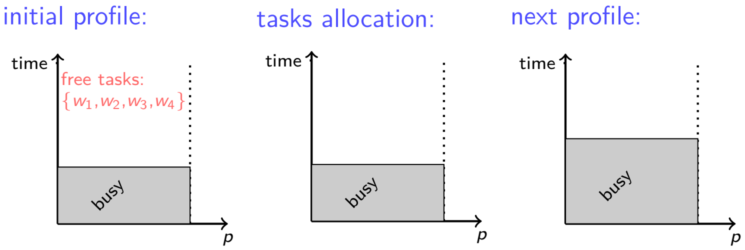
1. Analysis of PROPORTIONALMAPPING [Pothen et al. 1993]
2. Design of a greedy strategy
3. Analysis of FLOWFLEX [Balmin et al. 2013]
4. Experimental comparison
5. Conclusion

Design of a greedy strategy: GREEDY-FILLING

Algorithm

- Assign priorities to tasks (usually by bottom-level)
- Maintain a set of available tasks
- Consider free tasks by decreasing priority:
  - allocate δ_i^1 procs to each task until the limit
  - if remaining procs, increase allocation to δ_i^2 procs
- Stop the allocation when the first task terminates, then repeat

Illustration

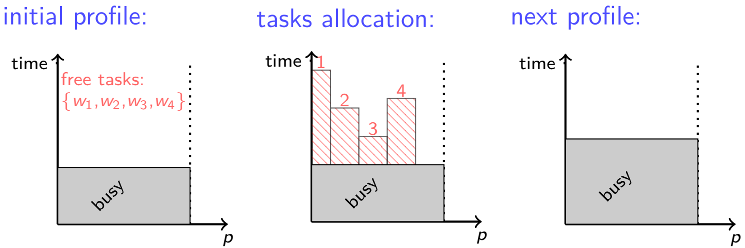


Design of a greedy strategy: GREEDY-FILLING

Algorithm

- Assign priorities to tasks (usually by bottom-level)
- Maintain a set of available tasks
- Consider free tasks by decreasing priority:
  - allocate δ_i^1 procs to each task until the limit
  - if remaining procs, increase allocation to δ_i^2 procs
- Stop the allocation when the first task terminates, then repeat

Illustration

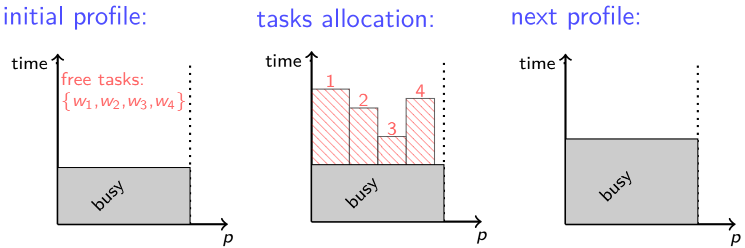


Design of a greedy strategy: GREEDY-FILLING

Algorithm

- Assign priorities to tasks (usually by bottom-level)
- Maintain a set of available tasks
- Consider free tasks by decreasing priority:
  - allocate δ_i^1 procs to each task until the limit
  - if remaining procs, increase allocation to δ_i^2 procs
- Stop the allocation when the first task terminates, then repeat

Illustration



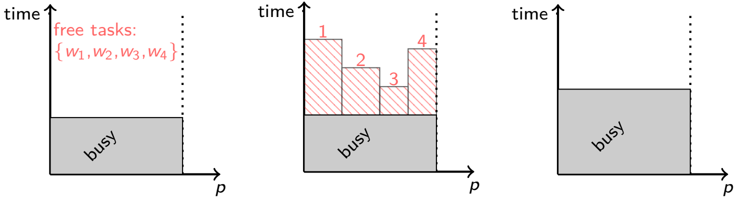
Design of a greedy strategy: GREEDY-FILLING

Algorithm

- Assign priorities to tasks (usually by bottom-level)
- Maintain a set of available tasks
- Consider free tasks by decreasing priority:
  - allocate δ_i^1 procs to each task until the limit
  - if remaining procs, increase allocation to δ_i^2 procs
- Stop the allocation when the first task terminates, then repeat

Illustration

initial profile: tasks allocation: next profile:

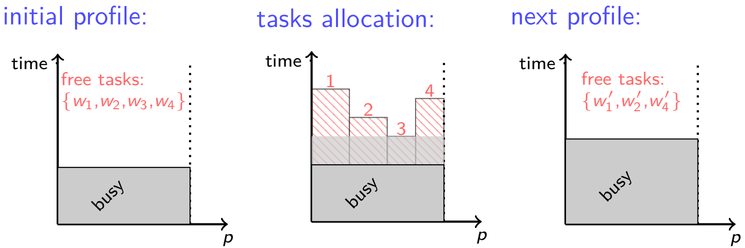


Design of a greedy strategy: GREEDY-FILLING

Algorithm

- Assign priorities to tasks (usually by bottom-level)
- Maintain a set of available tasks
- Consider free tasks by decreasing priority:
  - allocate δ_i^1 procs to each task until the limit
  - if remaining procs, increase allocation to δ_i^2 procs
- Stop the allocation when the first task terminates, then repeat

Illustration



Analysis of GREEDY-FILLING schedules

Theorem

GREEDY-FILLING *is a* $1 + r - \frac{\delta_{\min}^2}{p}$ *approximation to the optimal makespan, with* $r = \max_i \left(\delta_i^2 / \Sigma_i\right) \geq 1$.

Proof.

Transposition of the classical $(2 - \frac{1}{p})$-approximation result by Graham

- Construct a path Φ in G: all idle times happen during tasks of Φ

Analysis of GREEDY-FILLING schedules

Theorem

GREEDY-FILLING *is a* $1 + r - \frac{\delta_{\min}^2}{p}$ *approximation to the optimal makespan, with* $r = \max_i \left(\delta_i^2 / \Sigma_i\right) \geq 1$.

Proof.

Transposition of the classical $(2 - \frac{1}{p})$-approximation result by Graham

- Construct a path Φ in G: all idle times happen during tasks of Φ
- Bound *Used* and *Idle* areas $\left(\textit{Used} + \textit{Idle} = p\,M\right)$

Analysis of GREEDY-FILLING schedules

Theorem

GREEDY-FILLING *is a* $1 + r - \frac{\delta_{\min}^2}{p}$ *approximation to the optimal makespan, with* $r = \max_i \left(\delta_i^2 / \Sigma_i\right) \geq 1$.

Proof.

Transposition of the classical $(2 - \frac{1}{p})$-approximation result by Graham

- Construct a path Φ in G: all idle times happen during tasks of Φ
- Bound *Used* and *Idle* areas ($Used + Idle = p\,M$)
  - At least $\delta_{\min}$ processors busy during Φ so $Idle \leq (p - \delta_{min}^2) M_{\text{opt}}$

Analysis of GREEDY-FILLING schedules

Theorem

GREEDY-FILLING *is a* $1 + r - \frac{\delta_{\min}^2}{p}$ *approximation to the optimal makespan, with* $r = \max_i \left(\delta_i^2 / \Sigma_i\right) \geq 1$.

Proof.

Transposition of the classical $(2 - \frac{1}{p})$-approximation result by Graham

- Construct a path Φ in G: all idle times happen during tasks of Φ
- Bound *Used* and *Idle* areas ($Used + Idle = p\,M$)
  - At least $\delta_{\min}$ processors busy during Φ so $Idle \leq (p - \delta_{min}^2) M_{\text{opt}}$
  - s_i is concave so $Used \leq \sum_i \delta_i^2 \frac{w_i}{\Sigma_i} \leq rpM_{\text{opt}}$

□

Analysis of GREEDY-FILLING schedules

Theorem

GREEDY-FILLING *is a* $1 + r - \frac{\delta_{\min}^2}{p}$ *approximation to the optimal makespan, with* $r = \max_i \left(\delta_i^2 / \Sigma_i\right) \geq 1$.

Proof.

Transposition of the classical $(2 - \frac{1}{p})$-approximation result by Graham

- Construct a path Φ in G: all idle times happen during tasks of Φ
- Bound *Used* and *Idle* areas ($Used + Idle = p\,M$)
  - At least $\delta_{\min}$ processors busy during Φ so $Idle \leq (p - \delta_{min}^2) M_{\text{opt}}$
  - s_i is concave so $Used \leq \sum_i \delta_i^2 \frac{w_i}{\Sigma_i} \leq r p M_{\text{opt}}$

□

Note

- Theorem applies to every strategy without deliberate idle time

Outline

1. Analysis of ProportionalMapping [Pothen et al. 1993]
2. Design of a greedy strategy
3. Analysis of FlowFlex [Balmin et al. 2013]
4. Experimental comparison
5. Conclusion

FLOWFLEX [Balmin et al. 13]

Principle

- 2-approximation in the single-threshold model
- Solve the problem on an infinite number of processors
- On each interval with constant allocations: if the processor limit is exceeded, downscale the allocation proportionally

Adaptation to our model

- Similar to PROPMAPEXTTHRESH: when a task terminates, rebalance idling processors proportionally to the threshold
- *Note: if the single-threshold model is available, downscale the allocation proportionnally to this threshold*

Outline

1. Analysis of ProportionalMapping [Pothen et al. 1993]
2. Design of a greedy strategy
3. Analysis of FlowFlex [Balmin et al. 2013]
4. Experimental comparison
5. Conclusion

Experimental setup

Two datasets

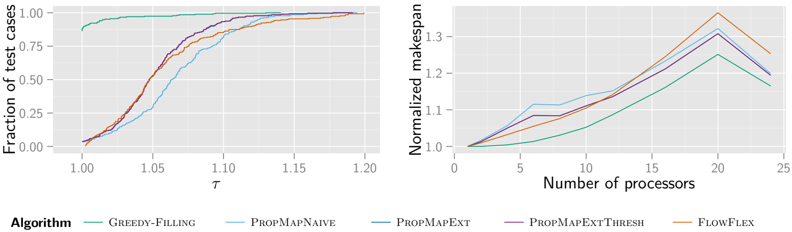
- SYNTH: 30 synthetic SP-graphs of 200 nodes with $\delta_i^1 = \alpha \times w_i$ and δ_i^2 uniform in $[\delta_i^1, 2\delta_i^1]$
- TREES: Assembly trees of 24 sparse matrices from 40 to 6000 nodes (University of Florida Sparse Matrix Collection), speedup deduced from timings explained earlier

Heuristics

- Greedy-Filling, PropMapNaive, PropMapExt, PropMapExtThresh, FlowFlex

Note: we tested 8 variants but only present the main ones

Results on SYNTH

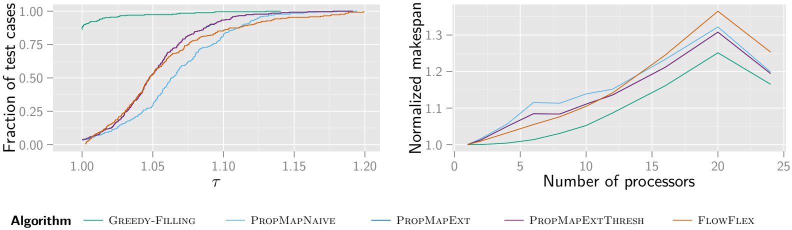


Comparison method: performance profiles (left graph)

- Determine the makespan for each instance (heuristic, graph, #procs)
- Given a heuristic *H* and a value $\tau \geq 1$: compute how often *H* lies within a factor τ of the best heuristic

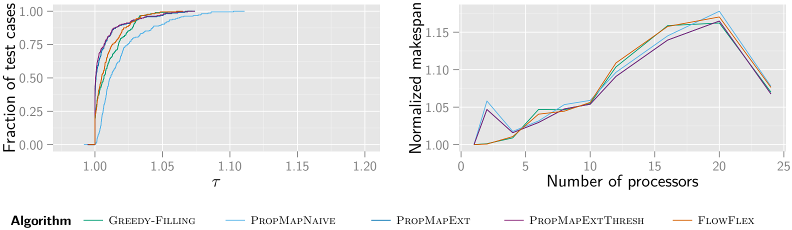
  For $\tau = 1.05$, Greedy-Filling *curve is at* 0.98*:*
  in 98% *of instances, it is within* 5% *of the best result*

Results on SYNTH



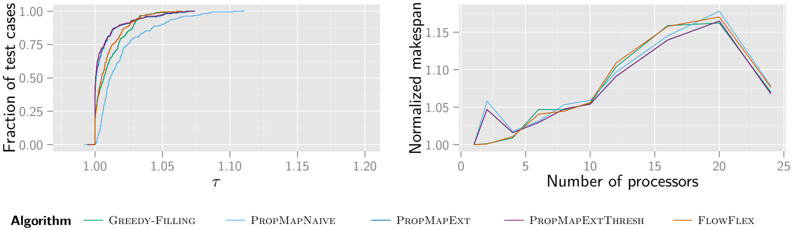
- Left: performance profile *(best is top-left)*
  - Greedy-Filling is almost always optimal and gains $> 5\%$ in 50% of the cases against any other heuristic
- Right: makespan normalized by a LB *(best is 1.0, bottom)*
  - Sample random graph
  - Results on different graphs are quite similar

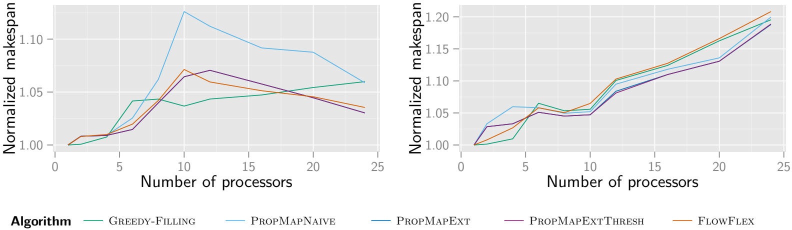
Results on TREES



- Left: performance profile *(best is top-left)*
  - Smaller discrepancies
  - PropMapExt and PropMapExtThresh perform better and are similar
- Right: makespan normalized by a LB *(best is 1.0, bottom)*
  - Exposes the results on a sample tree
  - Trees have different structures, so the heuristic hierarchy depends on the tree and the number of processors

Results on TREES





Outline

1. Analysis of PROPORTIONALMAPPING [Pothen et al. 1993]
2. Design of a greedy strategy
3. Analysis of FLOWFLEX [Balmin et al. 2013]
4. Experimental comparison
5. Conclusion

Conclusion

On the model

- Far more accurate than the single-threshold one
- NP-complete, as the single-threshold one
- Theoretically guaranteed heuristics

Conclusion

On the model

- Far more accurate than the single-threshold one
- NP-complete, as the single-threshold one
- Theoretically guaranteed heuristics

On the heuristics

- GREEDY-FILLING
  - best when the tree can be scheduled without forced idle times
  - best heuristic on SYNTH and other well-balanced instances
- PROPORTIONALMAPPING
  - naive version is not competitive
  - extensions are almost equivalent
  - give the best global results on TREES
  - best when large non-urgent tasks are available soon, or if several paths are critical