Scheduling Series-Parallel Graphs of Malleable Tasks

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

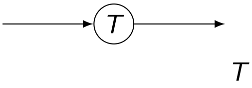
1: CNRS, INRIA, ENS Lyon and Univ. Lyon, FR.
2: Univ. Auckland, NZ.

11th Scheduling for Large Scale Systems Workshop

May 20, 2016

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



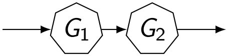
Objectives:

- 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



$G_1; G_2$

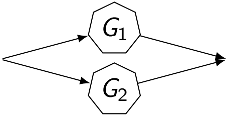
Objectives:

- 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



$G_1 \parallel G_2$

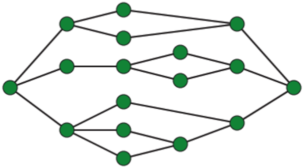
Objectives:

- 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



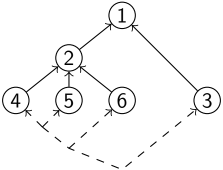
Objectives:

- 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
  - Perfectly-parallel tasks: schedule each task on the whole platform, same problem than scheduling sequential tasks on a single processor

General speed-up models

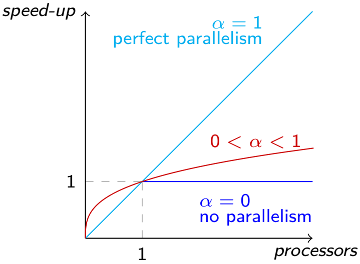
Literature: studies with few assumptions

Non-increasing speed-up and 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 [PM 1996]

- *speed-up*$(p) = p^{\alpha}$, with $0 < \alpha \leq 1$



- Task T_i of size L_i

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

Results for Prasanna & Musicus' model

Theorem (Prasanna & Musicus)

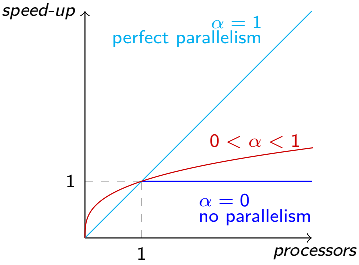
In optimal schedules, at any parallel node $G_1 \parallel G_2$, the ratio of processors given to each branch is constant.

Corollary

- *In optimal schedules:*
  - $\forall i, \; p_i(t)/p(t)$ *is constant*
  - *Children of a node terminate simultaneously*
- $G \approx$ *equivalent task* T_G *of length* $\mathcal{L}_G$ *defined by:*
  - $\mathcal{L}_{T_i} = L_i$
  - $\mathcal{L}_{G_1 \,;\, G_2} = \mathcal{L}_{G_1} + \mathcal{L}_{G_2}$
  - $\mathcal{L}_{G_1 \parallel G_2} = \left(\mathcal{L}_{G_1}^{1/\alpha} + \mathcal{L}_{G_2}^{1/\alpha}\right)^{\alpha}$
- *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 ☹

Integer or rational allotments?

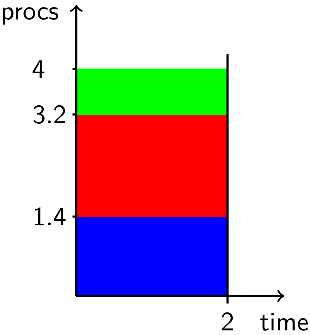
Question: should we allow allotments of rational number of cores to tasks, or keep to integral ones?

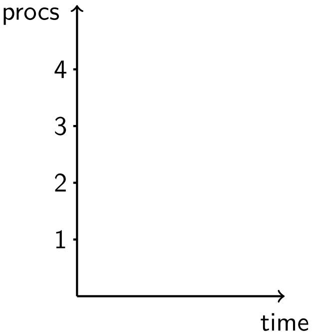
Integer or rational allotments?

Question: should we allow allotments of rational number of cores to tasks, or keep to integral ones?

Answer: for malleable tasks, rational allotments do not lead to any improvement.

McNaughton rule



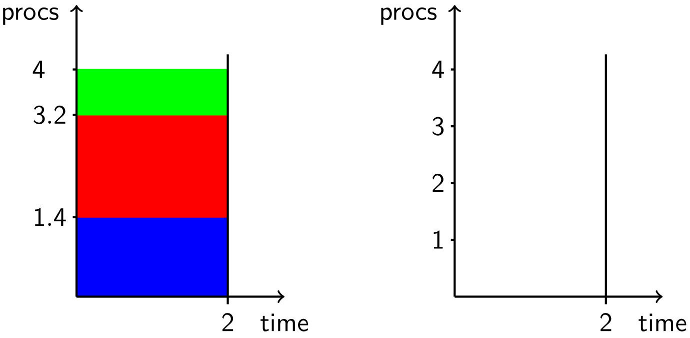


Integer or rational allotments?

Question: should we allow allotments of rational number of cores to tasks, or keep to integral ones?

Answer: for malleable tasks, rational allotments do not lead to any improvement.

McNaughton rule

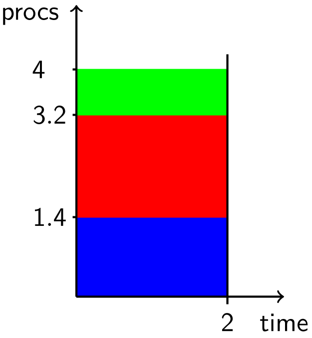


Integer or rational allotments?

Question: should we allow allotments of rational number of cores to tasks, or keep to integral ones?

Answer: for malleable tasks, rational allotments do not lead to any improvement.

McNaughton rule

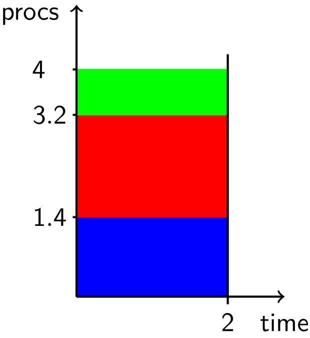


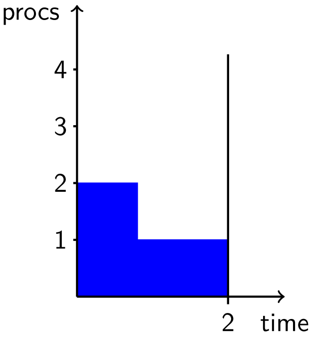
Integer or rational allotments?

Question: should we allow allotments of rational number of cores to tasks, or keep to integral ones?

Answer: for malleable tasks, rational allotments do not lead to any improvement.

McNaughton rule



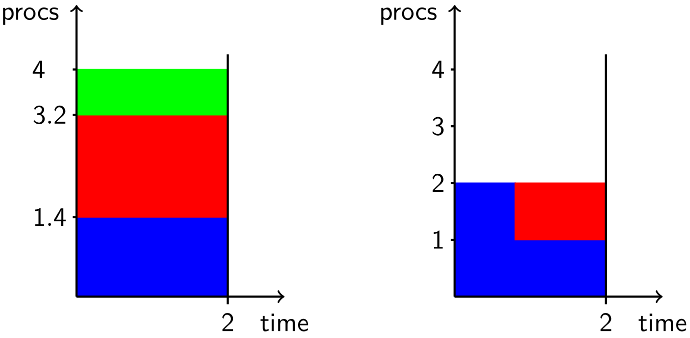


Integer or rational allotments?

Question: should we allow allotments of rational number of cores to tasks, or keep to integral ones?

Answer: for malleable tasks, rational allotments do not lead to any improvement.

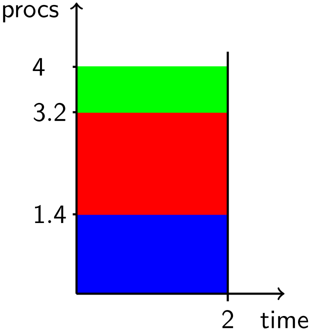
McNaughton rule

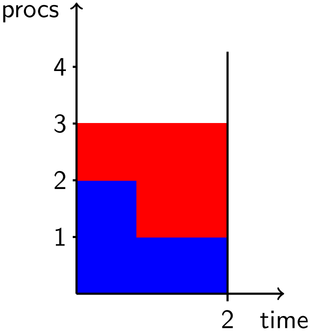


Question: should we allow allotments of rational number of cores to tasks, or keep to integral ones?

Answer: for malleable tasks, rational allotments do not lead to any improvement.

McNaughton rule



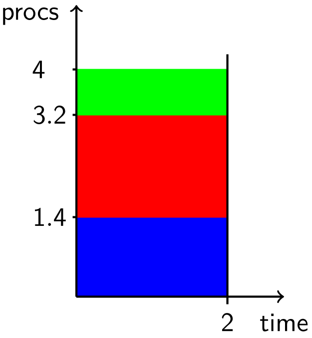


Integer or rational allotments?

Question: should we allow allotments of rational number of cores to tasks, or keep to integral ones?

Answer: for malleable tasks, rational allotments do not lead to any improvement.

McNaughton rule

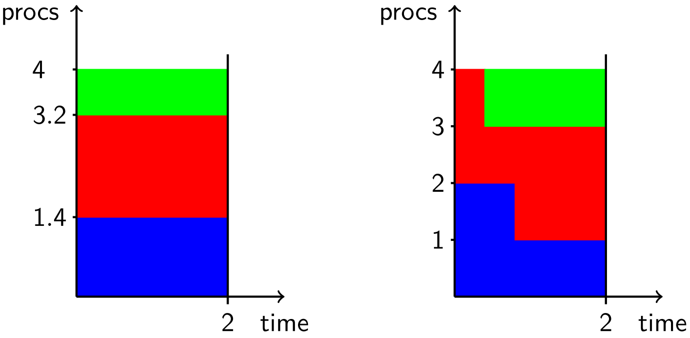


Integer or rational allotments?

Question: should we allow allotments of rational number of cores to tasks, or keep to integral ones?

Answer: for malleable tasks, rational allotments do not lead to any improvement.

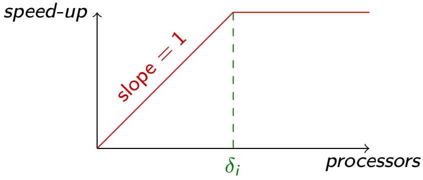
McNaughton rule



Today: simpler model

Simple and reasonable model of a parallel malleable task T_i

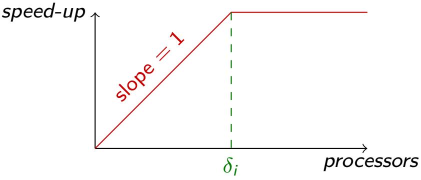
- Perfect parallelism up to a threshold δ_i: $time = w_i \,/\, \min(p, \delta_i)$



Today: simpler model

Simple and reasonable model of a parallel malleable task T_i

- Perfect parallelism up to a threshold δ_i: $time = w_i \,/\, \min(p, \delta_i)$



Related studies

- 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 $p \geq 4$

Outline

Overview of the problem

Given a SP-graph, p processors: compute the optimal makespan

- Problem known as $P|sp\text{-}graph, any, spdp\text{-}lin, \delta_i|C_{\max}$

Contribution

- New NP-completeness proof

Overview of the problem

Given a SP-graph, p processors: compute the optimal makespan

- Problem known as $P|sp\text{-}graph, any, spdp\text{-}lin, \delta_i|C_{\max}$
- Malleability + perfect parallelism $\Longrightarrow$ P ☺

Contribution

- New NP-completeness proof

Overview of the problem

Given a SP-graph, p processors: compute the optimal makespan

- Problem known as $P|sp\text{-}graph, any, spdp\text{-}lin, \delta_i|C_{\max}$
- Malleability + perfect parallelism $\Longrightarrow$ P ☺
- ··· + thresholds $\Longrightarrow$ NP-complete ☹

Contribution

- New NP-completeness proof

Overview of the problem

Given a SP-graph, p processors: compute the optimal makespan

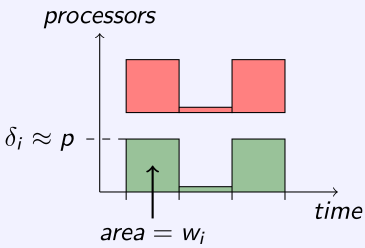
- Problem known as $P|sp\text{-}graph, any, spdp\text{-}lin, \delta_i|C_{max}$
- Malleability + perfect parallelism $\Longrightarrow$ P ☺
- ··· + thresholds $\Longrightarrow$ NP-complete ☹
- Existing proof in [Drozdowski and Kubiak 1999]: arguably complex

Contribution

- New NP-completeness proof

Widget for the proof

Two 3-task chains

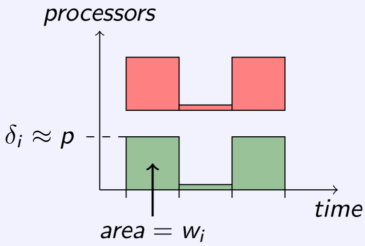


Each task:

- $\delta_i = w_i$
- min. computing time of 1

Widget for the proof

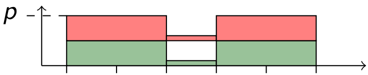
Two 3-task chains



Each task:

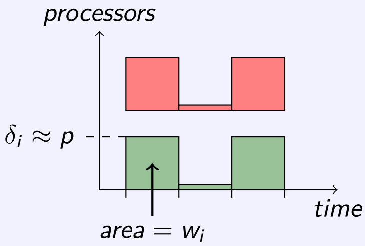
- $\delta_i = w_i$
- min. computing time of 1

Simultaneous start: $C_{max} \approx 5$



Widget for the proof

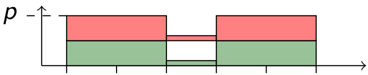
Two 3-task chains



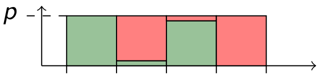
Each task:

- $\delta_i = w_i$
- min. computing time of 1

Simultaneous start: $C_{max} \approx 5$



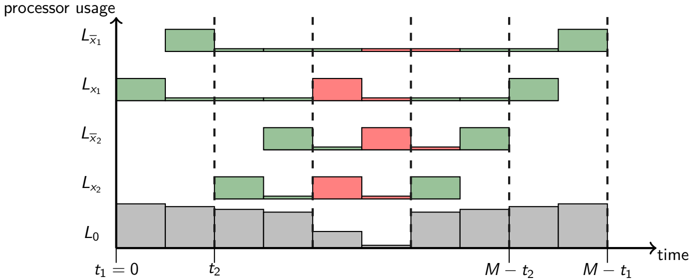
Time-shift: $C_{max} \approx 4$



Proof sketch

Reduction from 3-SAT (ex: x_1 *OR* x_2 *OR* $\overline{x}_2$)

- Idea: each variable ⇒ a modified widget (a chain for both x_i, $\overline{x}_i$)

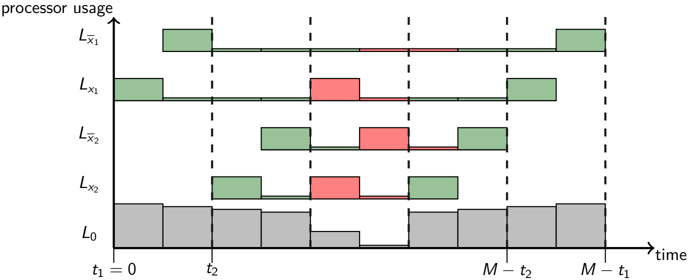


Possible schedule for x_1 OR x_2 or $\overline{x_2}$

Proof sketch

Reduction from 3-SAT (ex: x_1 *OR* x_2 *OR* $\overline{x}_2$)

- Idea: each variable ⇒ a modified widget (a chain for both x_i, $\overline{x}_i$)
- extremities ⇒ variables — middle ⇒ clauses

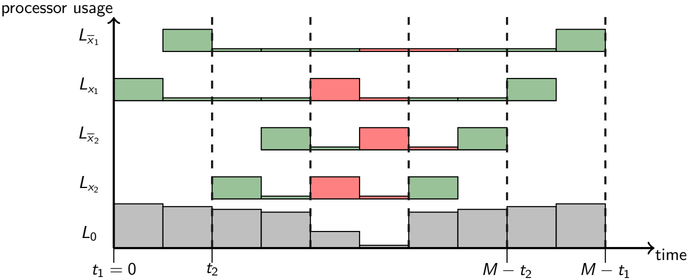


Possible schedule for x_1 OR x_2 or $\overline{x_2}$

Proof sketch

Reduction from 3-SAT (ex: x_1 *OR* x_2 *OR* $\overline{x}_2$)

- Idea: each variable ⇒ a modified widget (a chain for both x_i, $\overline{x}_i$)
- extremities ⇒ variables — middle ⇒ clauses
- The one starting later: TRUE

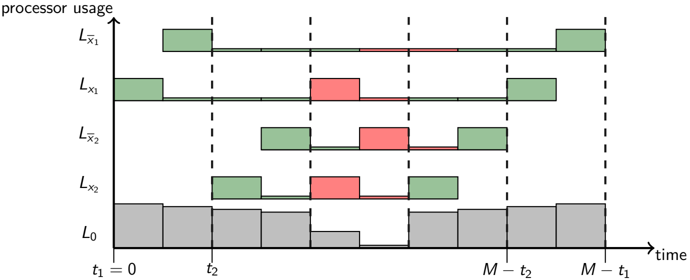


Possible schedule for x_1 OR x_2 or $\overline{x_2}$

Proof sketch

Reduction from 3-SAT (ex: x_1 *OR* x_2 *OR* $\overline{x}_2$)

- Idea: each variable ⇒ a modified widget (a chain for both x_i, $\overline{x}_i$)
- extremities ⇒ variables — middle ⇒ clauses
- The one starting later: TRUE
- Gray chain: profile allowing only *correct* behaviors



Possible schedule for x_1 OR x_2 or $\overline{x_2}$

Outline

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

if $G = G_1 \parallel G_2 \parallel \ldots G_k$ **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 2-approximation of the optimal makespan.*

Proof.

- Consider makespan without thresholds: $M_\infty \leq M_{\text{opt}}$
- There is an idle-free path Φ from the entry task to the end
- Split the tasks of Φ in two sets:

Note

- Approximation ratio asymptotically tight

Analysis of ProportionalMapping schedules

Theorem

ProportionalMapping *is a 2-approximation of the optimal makespan.*

Proof.

- Consider makespan without thresholds: $M_\infty \leq M_{\text{opt}}$
- There is an idle-free path Φ from the entry task to the end
- Split the tasks of Φ in two sets:
  - $A =$ tasks limited by their thresholds: $len(A) \leq$ critical path $\leq M_{\text{opt}}$

Note

- Approximation ratio asymptotically tight

Analysis of ProportionalMapping schedules

Theorem

ProportionalMapping *is a 2-approximation of the optimal makespan.*

Proof.

- Consider makespan without thresholds: $M_\infty \leq M_{\text{opt}}$
- There is an idle-free path Φ from the entry task to the end
- Split the tasks of Φ in two sets:
  - A = tasks limited by their thresholds: $len(A) \leq$ critical path $\leq M_{\text{opt}}$
  - B = tasks limited by the allocation: $len(B) \leq M_\infty \leq M_{\text{opt}}$

Note

- Approximation ratio asymptotically tight

Analysis of ProportionalMapping schedules

Theorem

ProportionalMapping *is a 2-approximation of the optimal makespan.*

Proof.

- Consider makespan without thresholds: $M_\infty \leq M_{\text{opt}}$
- There is an idle-free path Φ from the entry task to the end
- Split the tasks of Φ in two sets:
  - A = tasks limited by their thresholds: $len(A) \leq$ critical path $\leq M_{\text{opt}}$
  - B = tasks limited by the allocation: $len(B) \leq M_\infty \leq M_{\text{opt}}$
- Finally, $M = len(\Phi) = len(A) + len(B) \leq 2M_{\text{opt}}$ □

Note

- Approximation ratio asymptotically tight

Outline

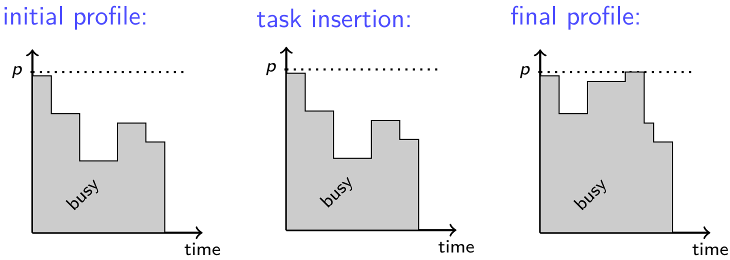
1. Problem complexity
2. Analysis of PROPORTIONALMAPPING [Pothen et al. 1993]
3. Design of a greedy strategy
4. Experimental comparison
5. Conclusion

Design of a greedy strategy: GREEDY-FILLING

Algorithm

- Assign priorities to tasks (usually by bottom-level)
- Consider free tasks by decreasing priority
- Greedily insert each task in the current schedule:

Illustration

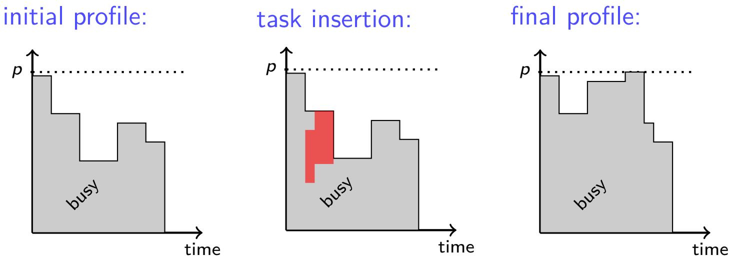


Design of a greedy strategy: GREEDY-FILLING

Algorithm

- Assign priorities to tasks (usually by bottom-level)
- Consider free tasks by decreasing priority
- Greedily insert each task in the current schedule:
    - Compute earliest starting time

Illustration

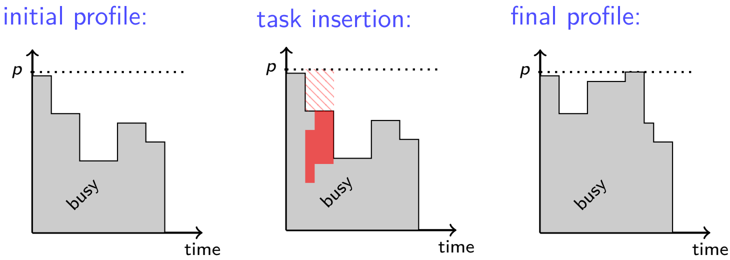


Design of a greedy strategy: Greedy-Filling

Algorithm

- Assign priorities to tasks (usually by bottom-level)
- Consider free tasks by decreasing priority
- Greedily insert each task in the current schedule:
  - Compute earliest starting time

Illustration

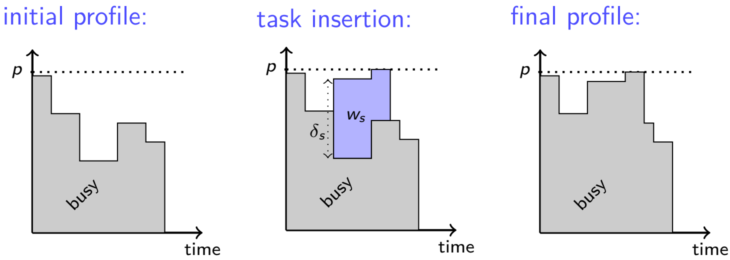


Design of a greedy strategy: GREEDY-FILLING

Algorithm

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

Illustration



Analysis of Greedy-Filling schedules

Theorem

Greedy-Filling *is a* $2 - \frac{\delta_{\min}}{p}$ *approximation to the optimal makespan.*

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

□

Note

- Theorem applies to every strategy without deliberate idle time

Outline

Simulations: a third algorithm to compare to

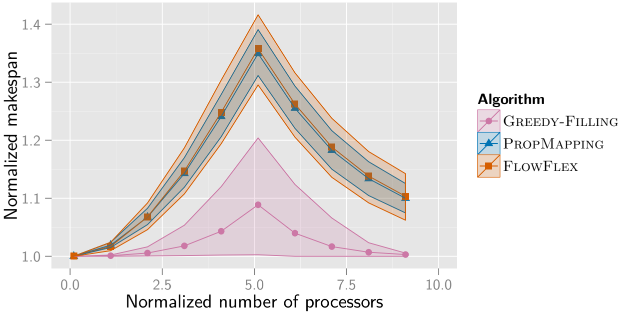
FLOWFLEX

- 2-approximation designed in [Balmin et al. 13] to schedule "Malleable Flows of MapReduce Jobs"
- Solve the problem on an infinite number of processors
- On each interval with constant allocations, if total allocation exceeds the number of available processors, downscale allocations proportionally

Simulations: three datasets

- SYNTH-PROP: Synthetic SP-graphs with $\delta_i = \alpha \times w_i$,
- SYNTH-RAND: Same but with a factor log-uniform in $[0.1\alpha, 10\alpha]$,
- TREES: Assembly trees of sparse matrices, $\delta_i = \alpha \times w_i$.

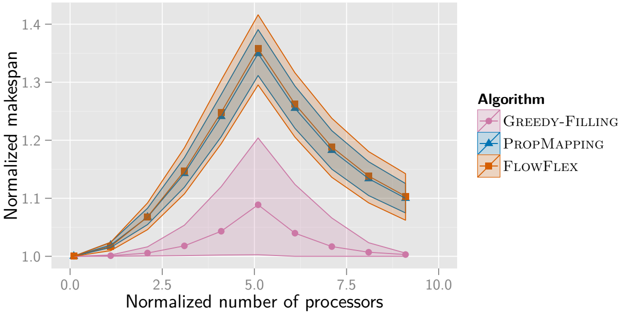
Results on SYNTH-PROP



- Y: Makespan normalized by the lower bound $LB = \max(CP, \frac{W}{p})$
- X: Number of processors normalized by:

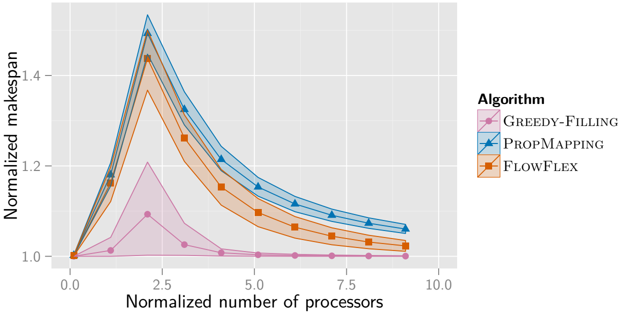
$$parallelism = \frac{\text{makespan with all } \delta_i = 1 \text{ and } p = \infty}{\text{makespan with all } \delta_i = 1 \text{ and } p = 1}$$

Results on SYNTH-PROP



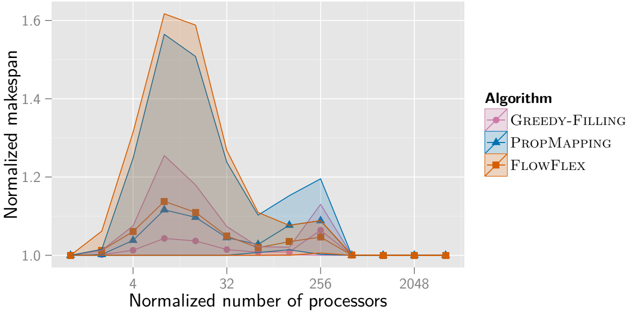
- Plot: mean + ribbon with 90% of the results
- Small/large number of processors: similar results (simpler problem)
- Greedy-Filling:
  - $\approx$ 25% of gain
  - $<$ 20% from the lower bound

Results on SYNTH-RAND



- Similar results with random thresholds
- Larger gaps between GREEDY-FILLING and the others
- Maximum gap happens for smaller platforms

Results on TREES



- Shape of the results depends a lot on the matrix
- Here: one matrix with different ordering and amalgamation parameters
- GREEDY-FILLING (almost always) better than both others
- Smaller maximum gain (around 15%)

Outline

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

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

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-approx. algorithms) ☺

Model extension

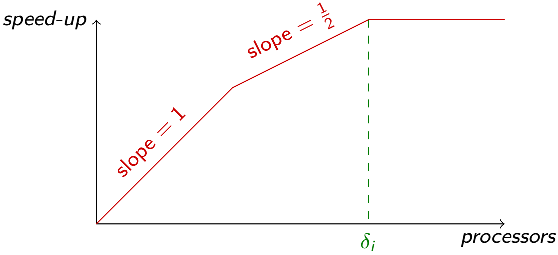
Model extension

Previous model:

- Two phases: perfect parallelism or constant speed-up

New model:

- Additional phases, phase i with $\textit{speed-up}(p) = \alpha_i \cdot p$



New approximation ratio: $\dfrac{1}{\min_i \alpha_i}\left(2 - \dfrac{\delta_{\min}}{p}\right)$

(Approximation ratio certainly false: PhD student on vacation and had forgotten to commit new result in svn.)