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

Loris Marchal1  Bertrand Simon1  Oliver Sinnen2  Frédéric Vivien1
1: LIP, CNRS, INRIA, ENS de Lyon and Univ. de Lyon.  2: University of Auckland, NZ.
Contact: bertrand.simon@ens-lyon.fr

## Objectsives
- 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., $G = T \cup G_1 \cup G_2$

We aim at:
- Guaranteeing widely used algorithms
- Designing better scheduling algorithms

## Related work
- Non-increasing speed-up and work
  - SP-graphs: 2.65-approximation [Lepre 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 [Ballin et al. 2013] named FLOWFLEX (see experimental setup)
  - [Kell et al. 2015]: $time = \frac{q_i}{p_i} + (p_i - 1)c_i$
  - 2-approximation for $p = 3$, open for $p \geq 4$

## Experimental setup
- Third algorithm for comparison: FLOWFLEX
  - 2-approximation from [Ballin 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, $\delta_i = \alpha \times w_i$

## Why malleable task trees suffice?
- **Coarse-grain picture**
  - Each task: partial factorization, graph of smaller sub-tasks

## NP-Completeness of the problem
- Complexity depending on the model
  - Malleability + perfect parallelism $\Rightarrow P$
  - Adding thresholds $\Rightarrow$ NP-complete
  - Arguably complex proof [Drózdowski and Kubiak 1999]

## Behavior of coarse-grain tasks
- Parallel and malleable
- Speed-up model $\rightarrow$ trade-off between:
  - Accuracy: fits well the data
  - Tractability: guaranteed algorithms

## Validation of GREEDY-FILLING
- Results on SYNTH
  - 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

## Results on TREES
- 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%)

## Previous work: Prasanna & Musicians model
- Focus on two quantities
  - speed-up($p$) $= \frac{time(1 \ proc)}{time(p \ proc)}$ = work($p$) = $p \cdot time(p \ proc)$

## 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 $G_1$, proportional to its weight $w_1$
- Algorithm on graph $G$ with $q$ processors:
  - **PROP MAPPING** ($G, q$)
    - If $G = G_1 \parallel \ldots \parallel G_k$ then
      - $PROP\ MAPPING(G_1, q)$
    - Call $PROP\ MAPPING(G_i, \frac{q}{w_i} \times q)$ for each $G_i$

## The widely used PROP MAPPING
- Then schedule each task on $p_i$ processors as soon as it is ready

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

## Theorem: PROP MAPPING 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

```
<table>
<thead>
<tr>
<th>Initial profile</th>
<th>Task insertion</th>
<th>Final profile</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

## Theorem: GREEDY-FILLING is a 2-approximation.

## A simple yet practical model
- Parallel malleable tasks
  - Perfect parallelism up to a threshold:
    - speed-up $= (\frac{1}{\alpha} + 1) \times \frac{w_i}{p_i}$
  - Total work: $w_i$ --- Threshold: $\alpha$
  - Rational allocation for free (McNaughton’s wrap-around rule)

## Conclusion
- On the algorithms
  - PROP MAPPING: 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)