Task-graph scheduling to minimize memory

Loris Marchal

Joint work with Henri Casanova, Mathias Jacquelin, Thomas Lambert, Yves Robert, Oliver Sinnen & Frédéric Vivien.

NCST 2012 Fréjus
Outline

Motivation and previous work

Parallel tree processing

Series-Parallel graphs

Summary and Perspectives
Outline

Motivation and previous work

Parallel tree processing

Series-Parallel graphs

Summary and Perspectives
Related Work: Register allocation

How to efficiently compute the following arithmetic expression with the minimum number of registers?

\[ 7 + (1 + x)(5 - z) - \left( \frac{(1 + x)}{(u - t)} + 2z \right) / v \]

Pebble-game rules:
- Inputs can be pebbled anytime
- If all ancestors are pebbled, a node can be pebbled
- A pebble may be removed anytime

Objective: pebble all outputs using minimum number of pebbles
**Related Work: Register allocation**

How to efficiently compute the following arithmetic expression with the minimum number of registers?

\[ 7 + (1 + x)(5 - z) - \frac{(1 + x)}{(u - t) + 2z} / v \]

**Pebble-game rules:**

- Inputs can be pebbled anytime
- If all ancestors are pebbled, a node can be pebbled
- A pebble may be removed anytime

**Objective:** pebble all outputs using minimum number of pebbles
Related Work: Register allocation

How to efficiently compute the following arithmetic expression with the minimum number of registers?

\[ 7 + (1 + x)(5 - z) - ((1 + x)/(u - t) + 2z)/v \]

Pebble-game rules:
- Inputs can be pebbled anytime
- If all ancestors are pebbled, a node can be pebbled
- A pebble may be removed anytime

Objective: pebble all outputs using minimum number of pebbles
Related Work: Register allocation

How to efficiently compute the following arithmetic expression with the minimum number of registers?

\[ 7 + (1 + x)(5 - z) - \frac{((1 + x)/(u - t) + 2z)}{v} \]

Pebble-game rules:

- Inputs can be pebbled anytime
- If all ancestors are pebbled, a node can be pebbled
- A pebble may be removed anytime

Objective: pebble all outputs using minimum number of pebbles
Related Work: Register allocation

How to efficiently compute the following arithmetic expression with the minimum number of registers?

\[ 7 + (1 + x)(5 - z) - \left(\frac{(1 + x)}{(u - t)} + 2z\right)/v \]

Pebble-game rules:
- Inputs can be pebbled anytime
- If all ancestors are pebbled, a node can be pebbled
- A pebble may be removed anytime

Objective: pebble all outputs using minimum number of pebbles
Related Work: Register allocation

How to efficiently compute the following arithmetic expression with the minimum number of registers?

\[
7 + (1 + x)(5 - z) - ((1 + x)/(u - t) + 2z)/v
\]

Pebble-game rules:
- Inputs can be pebbled anytime
- If all ancestors are pebbled, a node can be pebbled
- A pebble may be removed anytime

Objective: pebble all outputs using minimum number of pebbles
Related Work: Register allocation

How to efficiently compute the following arithmetic expression with the minimum number of registers?

\[ 7 + (1 + x)(5 - z) - ((1 + x)/(u - t) + 2z)/v \]

Pebble-game rules:
- Inputs can be pebbled anytime
- If all ancestors are pebbled, a node can be pebbled
- A pebble may be removed anytime

Objective: pebble all outputs using minimum number of pebbles
How to efficiently compute the following arithmetic expression with the minimum number of registers?

\[ 7 + (1 + x)(5 - z) - \frac{(1 + x)}{(u - t) + 2z}/v \]

Pebble-game rules:
- Inputs can be pebbled anytime
- If all ancestors are pebbled, a node can be pebbled
- A pebble may be removed anytime

Objective: pebble all outputs using minimum number of pebbles
How to efficiently compute the following arithmetic expression with the minimum number of registers?

$$7 + (1 + x)(5 - z) - ((1 + x)/(u - t) + 2z)/v$$

Pebble-game rules:
- Inputs can be pebbled anytime
- If all ancestors are pebbled, a node can be pebbled
- A pebble may be removed anytime

Objective: pebble all outputs using minimum number of pebbles
Related Work: Register allocation

How to efficiently compute the following arithmetic expression with the minimum number of registers?

\[ 7 + (1 + x)(5 - z) - ((1 + x)/(u - t) + 2z)/v \]

Complexity results

General problem on DAGs:
- P-Space complete [Gilbert, Lengauer & Tarjan, 1980]
- Without re-computation: NP-complete [Sethi, 1973]

Problem on trees:
- Polynomial algorithm [Sethi & Ullman, 1970]

Pebble-game rules:
- Inputs can be pebbled anytime
- If all ancestors are pebbled, a node can be pebbled
- A pebble may be removed anytime

Objective: pebble all outputs using minimum number of pebbles
New motivation: scientific computing

- Workflows with large data files
- Bad evolution of performance for computation vs. communication: $1/	ext{Flops} \ll 1/	ext{bandwidth} \ll \text{latency}$

- Gap between processing power and communication cost increasing exponentially

<table>
<thead>
<tr>
<th></th>
<th>annual improvements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flops rate</td>
<td>59%</td>
</tr>
<tr>
<td>mem. bandwidth</td>
<td>26%</td>
</tr>
<tr>
<td>mem. latency</td>
<td>5%</td>
</tr>
</tbody>
</table>

- Avoid communications
- Restrict to in-core memory (out-of-core is expensive)
New motivation: scientific computing

- Workflows with large data files
- Bad evolution of performance for computation vs. communication:
  \[ \frac{1}{\text{Flops}} \ll \frac{1}{\text{bandwidth}} \ll \text{latency} \]
- Gap between processing power and communication cost increasing exponentially

<table>
<thead>
<tr>
<th></th>
<th>annual improvements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flops rate</td>
<td>59%</td>
</tr>
<tr>
<td>mem. bandwidth</td>
<td>26%</td>
</tr>
<tr>
<td>mem. latency</td>
<td>5%</td>
</tr>
</tbody>
</table>

- Avoid communications
- Restrict to in-core memory (out-of-core is expensive)
Existing algorithms on trees

- Context: multifrontal sparse factorization
- Assembly tree: the DAG of the application is a tree
- Large tree with large input files

Two existing algorithms:
- Best post-order traversal [J. Liu, 1986]
- Best traversal [J. Liu, 1987]
Introduction: tree-shaped workflows

- In-tree of $n$ nodes
- Output file of size $f_i$
- Execution file of size $n_i$
- Input files of leaf nodes have null size
- Memory required for node $i$:

$$\text{MemReq}(i) = \sum_{j \in \text{Children}(i)} f_j + n_i + f_i$$

NB: top-down schedule = mirror of bottom-up schedule
Introduction: tree-shaped workflows

- In-tree of $n$ nodes
- Output file of size $f_i$
- Execution file of size $n_i$
- Input files of leaf nodes have null size
- Memory required for node $i$:

$$\text{MemReq}(i) = \sum_{j \in \text{Children}(i)} f_j + n_i + f_i$$

NB: top-down schedule = mirror of bottom-up schedule
Introduction: tree-shaped workflows

- In-tree of $n$ nodes
- Output file of size $f_i$
- Execution file of size $n_i$
- Input files of leaf nodes have null size
- Memory required for node $i$:
  \[ MemReq(i) = \sum_{j \in \text{Children}(i)} f_j + n_i + f_i \]

NB: top-down schedule = mirror of bottom-up schedule
Introduction: tree-shaped workflows

- In-tree of $n$ nodes
- Output file of size $f_i$
- Execution file of size $n_i$
- Input files of leaf nodes have null size
- Memory required for node $i$:

$$\text{MemReq}(i) = \sum_{j \in \text{Children}(i)} f_j + n_i + f_i$$

NB: top-down schedule = mirror of bottom-up schedule
Introduction: tree-shaped workflows

- In-tree of $n$ nodes
- Output file of size $f_i$
- Execution file of size $n_i$
- Input files of leaf nodes have null size
- Memory required for node $i$:

$$\text{MemReq}(i) = \sum_{j \in \text{Children}(i)} f_j + n_i + f_i$$

NB: top-down schedule = mirror of bottom-up schedule
Liu’s best post-order traversal for trees

- For each subtree $T_i$: peak memory $P_i$, residual memory $f_i$

- For a given processing order $1, \ldots, n$, the peak memory is:

$$\max\{P_1, f_1 + P_2, f_1 + f_2 + P_3, \ldots, \sum_{i<n} f_i + P_n, \sum_{i<n} f_i + n_r + f_r\}$$

- Optimal order:

- Postorder traversals are dominant when:

$$\sum_{j \in \text{Children}(i)} f_j \geq f_i$$
Liu’s best post-order traversal for trees

For each subtree $T_i$: peak memory $P_i$, residual memory $f_i$

For a given processing order $1, \ldots, n$, the peak memory is:

$$\max\{P_1, f_1 + P_2, f_1 + f_2 + P_3, \ldots, \sum_{i<n} f_i + P_n, \sum f_i + n_r + f_r\}$$

Optimal order:

Postorder traversals are dominant when:

$$\sum_{j \in \text{Children}(i)} f_j \geq f_i$$
Liu’s best post-order traversal for trees

- For each subtree $T_i$: peak memory $P_i$, residual memory $f_i$
- For a given processing order $1, \ldots, n$, the peak memory is:

$$\max\{P_1, f_1 + P_2, f_1 + f_2 + P_3, \ldots, \sum_{i<n} f_i + P_n, \sum f_i + n_r + f_r\}$$

- Optimal order:

- Postorder traversals are dominant when:

$$\sum_{j \in \text{Children}(i)} f_j \geq f_i$$
Liu’s best post-order traversal for trees

- For each subtree $T_i$: peak memory $P_i$, residual memory $f_i$
- For a given processing order $1, \ldots, n$, the peak memory is:
  \[
  \max\{P_1, f_1 + P_2, f_1 + f_2 + P_3, \ldots, \sum_{i<n} f_i + P_n, \sum f_i + n_r + f_r\}
  \]

- Optimal order:
- Postorder traversals are dominant when:
  \[
  \sum_{j \in \text{Children}(i)} f_j \geq f_i
  \]
Liu’s best post-order traversal for trees

- For each subtree $T_i$: peak memory $P_i$, residual memory $f_i$
- For a given processing order $1, \ldots, n$, the peak memory is:
  \[
  \max\{P_1, f_1 + P_2, f_1 + f_2 + P_3, \ldots, \sum_{i<n} f_i + P_n, \sum f_i + n_r + f_r\}
  \]
- Optimal order:
- Postorder traversals are dominant when:
  \[
  \sum_{j \in \text{Children}(i)} f_j \geq f_i
  \]
Liu’s best post-order traversal for trees

- For each subtree $T_i$: peak memory $P_i$, residual memory $f_i$
- For a given processing order $1, \ldots, n$, the peak memory is:

$$\max\{P_1, f_1 + P_2, f_1 + f_2 + P_3, \ldots, \sum_{i<n} f_i + P_n, \sum f_i + n_r + f_r\}$$

- Optimal order: non-increasing $P_i - f_i$
- Postorder traversals are dominant when:

$$\sum_{j \in \text{Children}(i)} f_j \geq f_i$$
Liu’s best post-order traversal for trees

- For each subtree $T_i$: peak memory $P_i$, residual memory $f_i$
- For a given processing order $1, \ldots, n$, the peak memory is:
  \[
  \max\{P_1, f_1 + P_2, f_1 + f_2 + P_3, \ldots, \sum_{i<n} f_i + P_n, \sum_{i<n} f_i + n_r + f_r\}
  \]
- Optimal order: non-increasing $P_i - f_i$
- Postorder traversals are dominant when:
  \[
  \sum_{j \in \text{Children}(i)} f_j \geq f_i
  \]
Post-Order is not optimal...

Postorder traversals are arbitrarily bad in the general case

There is no constant $k$ such that the best postorder traversal is a $k$-approximation.

Minimum peak memory:
$$M_{\text{min}} = M + \epsilon + \epsilon (b-1)$$

Minimum postorder peak memory traversal:
$$M_{\text{min}} = M + \epsilon + \epsilon (b-1)M/b$$

<table>
<thead>
<tr>
<th>actual assembly trees</th>
<th>random trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non optimal traversals</td>
<td>4.2%</td>
</tr>
<tr>
<td>Maximum increase compared to optimal</td>
<td>18%</td>
</tr>
<tr>
<td>Average increased compared to optimal</td>
<td>1%</td>
</tr>
</tbody>
</table>
Post-Order is not optimal...

Postorder traversals are arbitrarily bad in the general case.

There is no constant $k$ such that the best postorder traversal is a $k$-approximation.

- Minimum peak memory:
  $$M_{\text{min}} = M + \epsilon + (b-1)\epsilon$$

- Minimum postorder peak memory traversal:
  $$M_{\text{min}} = M + \epsilon + (b-1)M/b$$

<table>
<thead>
<tr>
<th></th>
<th>actual assembly trees</th>
<th>random trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non optimal traversals</td>
<td>4.2%</td>
<td>61%</td>
</tr>
<tr>
<td>Maximum increase compared to optimal</td>
<td>18%</td>
<td>22%</td>
</tr>
<tr>
<td>Average increased compared to optimal</td>
<td>1%</td>
<td>12%</td>
</tr>
</tbody>
</table>
Post-Order is not optimal...

Postorder traversals are arbitrarily bad in the general case

There is no constant $k$ such that the best postorder traversal is a $k$-approximation.

- Minimum peak memory:
  $M_{\text{min}} = M + \epsilon + (b - 1)\epsilon$

- Minimum postorder peak memory traversal:
  $M_{\text{min}} = M + \epsilon + (b-1)M/b$

<table>
<thead>
<tr>
<th></th>
<th>actual assembly trees</th>
<th>random trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non optimal traversals</td>
<td>4.2%</td>
<td>61%</td>
</tr>
<tr>
<td>Maximum increase compared to optimal</td>
<td>18%</td>
<td>22%</td>
</tr>
<tr>
<td>Average increased compared to optimal</td>
<td>1%</td>
<td>12%</td>
</tr>
</tbody>
</table>
Post-Order is not optimal...

Postorder traversals are arbitrarily bad in the general case

There is no constant $k$ such that the best postorder traversal is a $k$-approximation.

- Minimum peak memory:
  $$M_{\text{min}} = M + \epsilon + 2(b-1)\epsilon$$

- Minimum postorder peak memory traversal:
  $$M_{\text{min}} = M + \epsilon + 2(b-1)M/b$$

<table>
<thead>
<tr>
<th></th>
<th>actual assembly trees</th>
<th>random trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non optimal traversals</td>
<td>4.2%</td>
<td>61%</td>
</tr>
<tr>
<td>Maximum increase compared to optimal</td>
<td>18%</td>
<td>22%</td>
</tr>
<tr>
<td>Average increased compared to optimal</td>
<td>1%</td>
<td>12%</td>
</tr>
</tbody>
</table>
Post-Order is not optimal...but almost!

Postorder traversals are arbitrarily bad in the general case

There is no constant $k$ such that the best postorder traversal is a $k$-approximation.

- Minimum peak memory:
  $$M_{\text{min}} = M + \epsilon + (b - 1)\epsilon$$

- Minimum postorder peak memory traversal:
  $$M_{\text{min}} = M + \epsilon + (b - 1)M/b$$

<table>
<thead>
<tr>
<th></th>
<th>actual assembly trees</th>
<th>random trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non optimal traversals</td>
<td>4.2%</td>
<td>61%</td>
</tr>
<tr>
<td>Maximum increase compared to optimal</td>
<td>18%</td>
<td>22%</td>
</tr>
<tr>
<td>Average increased compared to optimal</td>
<td><strong>1%</strong></td>
<td>12%</td>
</tr>
</tbody>
</table>
Outline

Motivation and previous work

Parallel tree processing

Series-Parallel graphs

Summary and Perspectives
Parallel tree processing

- $p$ identical processors
- Node $i$ has execution times $p_i$
- Parallel processing of nodes $\Rightarrow$ larger memory
- Trade-off time vs. memory
NP-completeness in the pebble game model

Background:

- Makespan minimization NP-complete for trees ($P|\text{trees}|C_{\text{max}}$)
- Polynomial when unit-weight tasks ($P|p_i = 1, \text{trees}|C_{\text{max}}$)
- Pebble game polynomial on trees

Pebble game model:

- unit execution time: $p_i = 1$
- unit memory costs: $n_i = 0, f_i = 1$
  (pebble edges, equivalent to pebble game for trees)

Theorem
Deciding whether a tree can be scheduled using at most $B$ pebbles in at most $C$ steps is NP-complete.
NP-completeness – proof

Reduction from 3-Partition:

Schedule the tree using:

- $p = 3mB$ processors,
- at most $B = 3m \times B + 3m$ pebbles,
- at most $C = 2m + 1$ steps.
Joint minimization of both objectives

No zenith approximation:

Theorem
There is no algorithm that is both an $\alpha$-approximation for makespan minimization and a $\beta$-approximation for memory peak minimization when scheduling tree-shaped task graphs.

(proof sketch on next slide)
No zenith approx. – proof

- $n$ identical subtrees, largest in-degree is $\delta$
- $M_{seq} = \delta + n$; $C^*_{max} \geq \delta + 2$ (critical path = height + 1)

To achieve $\alpha C^*_{max} = \alpha(\delta + 2)$ each $cp^i_k$ node needs to finish at $\alpha(\delta + 2) - 1$

- Calculate number of edges in each subtree, each edge present during a least two steps
- Calculate average memory with $\alpha(\delta + 2) - 2$ steps $\Rightarrow$ lower bound $lb$
- By setting $\delta = n^2$, we show that $lb$ on memory is greater than $2\beta$ for any $\beta$ we choose $\Rightarrow$ contradiction
Approximability overview with fixed $p$

Inexistence of solutions which are $\alpha$-approximation for the makespan and $\beta$-approximation for the memory, with fixed number of processors.

- A: $(\alpha, \frac{\sqrt{p+1}}{2\alpha} + \frac{1}{\alpha^2})$
- B: $(1, p - 1)$
- C: $(\alpha, 1)$
Inexistence of solutions which are $\alpha$-approximation for the makespan and $\beta$-approximation for the memory, with fixed number of processors.

- A: $\left(\alpha, \frac{\sqrt{p+1}}{2\alpha} + \frac{1}{\alpha^2}\right)$
- B: $(1, p - 1)$
- C: $(\alpha, 1)$
Approximability overview with fixed $p$

Inexistence of solutions which are $\alpha$-approximation for the makespan and $\beta$-approximation for the memory, with **fixed** number of processors.

\[
\begin{align*}
\alpha (C_{\text{max}}) & \quad \beta (\text{Mem}) \\
\infty & \quad \infty \\
p & \quad p \\
p - 1 & \quad p - 1 \\
\end{align*}
\]

- A: \((\alpha, \frac{\sqrt{p+1}}{2\alpha} + \frac{1}{\alpha^2})\)
- B: \((1, p - 1)\)
- C: \((\alpha, 1)\)
Heuristics for weighted trees – 1/2

List-scheduling heuristics:
▶ Put ready nodes in a queue (sorted with some criterion)
▶ Schedule them whenever a processor is ready

Leaf nodes sorted using best sequential postorder

Two list-scheduling heuristics:
▶ Deepest-First (longest critical path, makespan oriented)
▶ Inner-First (memory oriented, sort of parallel postorder)

Performance:
▶ $(2 - 1/p)$-approximation for makespan
▶ Unbounded ratio for memory
Heuristics for weighted trees 2/2

Another memory-oriented heuristic:

- Split tree into subtrees
- Process $p$ subtrees in parallel
- Process remaining nodes sequentially

$$C_{\text{max}} = \max_{p \text{ largest subtrees}} p(T_i) + \sum_{\text{remaining nodes } j} p_j$$

Optimal subtree splitting (for makespan):

- Start with a single subtree (the tree)
- Split largest subtree until it is a single leaf node
- Store solution at each step
- Take the solution with minimal makespan

Memory guarantee:

- $p$-approximation algorithm

Optimization:

- Simple load-balancing of all subtrees to the processors
Heuristics for weighted trees 2/2

Another memory-oriented heuristic:
- Split tree into subtrees
- Process $p$ subtrees in parallel
- Process remaining nodes sequentially

$$C_{\text{max}} = \max_{p \text{ largest subtrees}} p(T_i) + \sum_{\text{remaining nodes } j} p_j$$

Optimal subtree splitting (for makespan):
- Start with a single subtree (the tree)
- Split largest subtree until it is a single leaf node
- Store solution at each step
- Take the solution with minimal makespan

Memory guarantee:
- $p$-approximation algorithm

Optimization:
- Simple load-balancing of all subtrees to the processors
Heuristics for weighted trees 2/2

Another memory-oriented heuristic:
- Split tree into subtrees
- Process $p$ subtrees in parallel
- Process remaining nodes sequentially

$$C_{\text{max}} = \max_{\text{$p$ largest subtrees}} p(T_i) + \sum_{\text{remaining nodes } j} p_j$$

Optimal subtree splitting (for makespan):
- Start with a single subtree (the tree)
- Split largest subtree until it is a single leaf node
- Store solution at each step
- Take the solution with minimal makespan

Memory guarantee:
- $p$-approximation algorithm

Optimization:
- Simple load-balancing of all subtrees to the processors
Another memory-oriented heuristic:

- Split tree into subtrees
- Process $p$ subtrees in parallel
- Process remaining nodes sequentially

$$C_{\text{max}} = \max_{p \text{ largest subtrees}} p(T_i) + \sum_{\text{remaining nodes } j} p_j$$

Optimal subtree splitting (for makespan):

- Start with a single subtree (the tree)
- Split largest subtree until it is a single leaf node
- Store solution at each step
- Take the solution with minimal makespan

Memory guarantee:

- $p$-approximation algorithm

Optimization:

- Simple load-balancing of all subtrees to the processors
Experimental testbed

- 76 assembly trees of a set of sparse matrices from University of Florida Sparse Collection
- Metis and AMD ordering
- 1, 2, 4, or 16 relaxed amalgamation per node
- 608 trees with:
  - number of nodes: 2,000 to 1,000,000
  - depth: 12 to 70,000
  - maximum degree: 2 to 175,000
Results

- Memory lower bound: best sequential postorder
- Makespan lower bound: \( \max \left\{ \frac{W}{p}, W_{\text{critical path}} \right\} \)
Outline

Motivation and previous work

Parallel tree processing

Series-Parallel graphs

Summary and Perspectives
Series-Parallel graphs: Motivation

- Not all scientific workflows are trees
- But most workflows exhibit some regularity
- Large class of workflows: Series-Parallel graphs

For now: only sequential processing
Series-Parallel graphs: Motivation

- Not all scientific workflows are trees
- But most workflows exhibit some regularity
- Large class of workflows: Series-Parallel graphs

For now: only sequential processing
Series-Parallel graphs: Motivation

- Not all scientific workflows are trees
- But most workflows exhibit some regularity
- Large class of workflows: Series-Parallel graphs

For now: only sequential processing
Series-Parallel graphs: Motivation

- Not all scientific workflows are trees
- But most workflows exhibit some regularity
- Large class of workflows: Series-Parallel graphs

For now: only sequential processing
Series-Parallel graphs: Motivation

- Not all scientific workflows are trees
- But most workflows exhibit some regularity
- Large class of workflows: Series-Parallel graphs

For now: only sequential processing
First step: fork-join graphs

Select edges with minimal weight on each branch: $e_1, \ldots, e_B$

**Theorem**

There exists a schedule with minimal memory which synchronises at $e_1, \ldots, e_B$.

**Algorithm:**

1. Apply optimal algorithm for out-trees on the left part
2. Apply optimal algorithm for in-trees on the right part
First step: fork-join graphs

Select edges with minimal weight on each branch: $e_1, \ldots, e_B$

**Theorem**
There exists a schedule with minimal memory which synchronises at $e_1, \ldots, e_B$.

**Algorithm:**
1. Apply optimal algorithm for out-trees on the left part
2. Apply optimal algorithm for in-trees on the right part
First step: fork-join graphs

Select edges with minimal weight on each branch: \( e_1, \ldots, e_B \)

**Theorem**

There exists a schedule with minimal memory which synchronises at \( e_1, \ldots, e_B \).

**Algorithm:**

1. Apply optimal algorithm for out-trees on the left part
2. Apply optimal algorithm for in-trees on the right part
First step: fork-join graphs

Select edges with minimal weight on each branch: $e_1, \ldots, e_B$

**Theorem**
There exists a schedule with minimal memory which synchronises at $e_1, \ldots, e_B$.

**Algorithm:**
1. Apply optimal algorithm for out-trees on the left part
2. Apply optimal algorithm for in-trees on the right part
Recursive algorithm:

- Apply fork-join algorithm starting with innermost parallel composition
- Replace parallel composition with sequential schedule

Good candidate for optimal algorithm:

- Always optimal in brute-force simulations
- Sketch of proof, adapted from Liu
Outline

Motivation and previous work

Parallel tree processing

Series-Parallel graphs

Summary and Perspectives
Summary and Perspectives

- Comprehensive study of tree-shaped task graphs (postorder, optimal sequential, complexity and heuristics for parallel processing)
- Adaptation to Series-Parallel graphs

Future work:
- Design memory-bounded heuristics for parallel tree processing
- Extend results to other class of regular graphs (2D grids, etc.)
- Minimize I/O volume for out-of-core execution

Thank you!
Summary and Perspectives

- Comprehensive study of tree-shaped task graphs (postorder, optimal sequential, complexity and heuristics for parallel processing)
- Adaptation to Series-Parallel graphs

Future work:
- Design memory-bounded heuristics for parallel tree processing
- Extend results to other class of regular graphs (2D grids, etc.)
- Minimize I/O volume for out-of-core execution

Thank you!