Scheduling task-graphs under memory constraints: a short state of the art

Loris Marchal (CNRS, Lyon, France)

SOLHAR scheduling meeting, April 10, 2014
Outline

Tree-Shaped Task graphs, Single Machine, Memory Minimization
  Foundations: Register Allocation & Pebble Game
  Liu’s algorithms

Extensions on Single Machine
  I/O Minimization
  Series-Parallel Task-Graphs

Extensions on Parallel Machines
  Sequential Tasks, Parallel Machine
  Malleable Tasks
  Hybrid Scheduling
Outline

Tree-Shaped Task graphs, Single Machine, Memory Minimization
  Foundations: Register Allocation & Pebble Game
  Liu’s algorithms

Extensions on Single Machine
  I/O Minimization
  Series-Parallel Task-Graphs

Extensions on Parallel Machines
  Sequential Tasks, Parallel Machine
  Malleable Tasks
  Hybrid Scheduling
How to efficiently compute the following arithmetic expression with the minimum number of registers?

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

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 root node using minimum number of pebbles
Foundations: Register Allocation & Pebble Game

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

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

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 root node using minimum number of pebbles
Foundations: Register Allocation & Pebble Game

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

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

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 root node using minimum number of pebbles
Foundations: Register Allocation & Pebble Game

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

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

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 root node using minimum number of pebbles
How to efficiently compute the following arithmetic expression with the minimum number of registers?

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

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 root node using minimum number of pebbles
Foundations: Register Allocation & Pebble Game

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

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

Complexity results

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

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

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 root node using minimum number of pebbles
Tree-Shaped Task Graphs

- In-tree of $n$ nodes
- Output data of size $f_i$
- Execution data of size $n_i$
- Input data of leaf nodes have null size

Memory for node $i$: $\text{MemReq}(i) = \left( \sum_{j \in \text{Children}(i)} f_j \right) + n_i + f_i$

or: $\text{MemReq}(i) = \max \left\{ \sum_{j \in \text{Children}(i)} f_j, f_i \right\}$

Extensively studied by Liu (single machine):
- Best post-order traversal [J. Liu, 1986]
- Best traversal [J. Liu, 1987]
Tree-Shaped Task Graphs

- In-tree of $n$ nodes
- Output data of size $f_i$
- Execution data of size $n_i$
- Input data of leaf nodes have null size

- Memory for node $i$: $\text{MemReq}(i) = \left( \sum_{j \in \text{Children}(i)} f_j \right) + n_i + f_i$

- or: $\text{MemReq}(i) = \max \left\{ \sum_{j \in \text{Children}(i)} f_j, f_i \right\}$

Extensively studied by Liu (single machine):
- Best post-order traversal [J. Liu, 1986]
- Best traversal [J. Liu, 1987]
Tree-Shaped Task Graphs

- In-tree of \( n \) nodes
- Output data of size \( f_i \)
- Execution data of size \( n_i \)
- Input data of leaf nodes have null size

Memory for node \( i \): \( \text{MemReq}(i) = \left( \sum_{j \in \text{Children}(i)} f_j \right) + n_i + f_i \)

or: \( \text{MemReq}(i) = \max \left\{ \sum_{j \in \text{Children}(i)} f_j, f_i \right\} \)

Extensively studied by Liu (single machine):
- Best post-order traversal [J. Liu, 1986]
- Best traversal [J. Liu, 1987]
Tree-Shaped Task Graphs

In-tree of $n$ nodes
Output data of size $f_i$
Execution data of size $n_i$
Input data of leaf nodes have null size

Memory for node $i$: $\text{MemReq}(i) = \left( \sum_{j \in \text{Children}(i)} f_j \right) + n_i + f_i$

or: $\text{MemReq}(i) = \max \left\{ \sum_{j \in \text{Children}(i)} f_j, f_i \right\}$

Extensively studied by Liu (single machine):
- Best post-order traversal [J. Liu, 1986]
- Best traversal [J. Liu, 1987]
Tree-Shaped Task Graphs

- In-tree of $n$ nodes
- Output data of size $f_i$
- Execution data of size $n_i$
- Input data of leaf nodes have null size

Memory for node $i$: 
$$\text{MemReq}(i) = \left( \sum_{j \in \text{Children}(i)} f_j \right) + n_i + f_i$$

or:
$$\text{MemReq}(i) = \max \left\{ \sum_{j \in \text{Children}(i)} f_j, f_i \right\}$$

Extensively studied by Liu (single machine):
- Best post-order traversal [J. Liu, 1986]
- Best traversal [J. Liu, 1987]
Tree-Shaped Task Graphs

- In-tree of $n$ nodes
- Output data of size $f_i$
- Execution data of size $n_i$
- Input data of leaf nodes have null size

Memory for node $i$: $\text{MemReq}(i) = \left( \sum_{j \in \text{Children}(i)} f_j \right) + n_i + f_i$

or: $\text{MemReq}(i) = \max \left\{ \sum_{j \in \text{Children}(i)} f_j, f_i \right\}$

Extensively studied by Liu (single machine):
- Best post-order traversal [J. Liu, 1986]
- Best traversal [J. Liu, 1987]
Liu’s Best Post-Order Traversal for Trees

Post-Order: entirely process one subtree after the other (DFS)

▶ 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:
▶ Post-Order traversals are dominant for unit-weight trees
Liu’s Best Post-Order Traversal for Trees

Post-Order: entirely process one subtree after the other (DFS)

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:

Post-Order traversals are dominant for unit-weight trees
Liu’s Best Post-Order Traversal for Trees

Post-Order: entirely process one subtree after the other (DFS)

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:

Post-Order traversals are dominant for unit-weight trees
Liu’s Best Post-Order Traversal for Trees

Post-Order: entirely process one subtree after the other (DFS)

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:

Post-Order traversals are dominant for unit-weight trees
Post-Order: entirely process one subtree after the other (DFS)

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:

Post-Order traversals are dominant for unit-weight trees
Liu’s Best Post-Order Traversal for Trees

Post-Order: entirely process one subtree after the other (DFS)

- 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 $P_i - f_i$
- Post-Order traversals are dominant for unit-weight trees
Liu’s Best Post-Order Traversal for Trees

Post-Order: entirely process one subtree after the other (DFS)

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

For a given processing order 1, ..., $n$, 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 $P_i - f_i$

Post-Order traversals are dominant for unit-weight trees
Post-Order is not optimal...

Post-Order traversals are arbitrarily bad in the general case

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

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

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

<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...

Post-Order traversals are arbitrarily bad in the general case

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

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

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

<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...

Post-Order traversals are arbitrarily bad in the general case

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

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

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

<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 Increase Compared to Optimal</td>
<td>1%</td>
</tr>
</tbody>
</table>
Post-Order is not optimal...

Post-Order traversals are arbitrarily bad in the general case

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

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

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

<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!

Post-Order traversals are arbitrarily bad in the general case

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

Minimum peak memory:

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

Minimum post-order peak memory:

$$M_{\text{min}} = M + \epsilon + (b - 1)M/b$$

<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>
Optimal General Traversal

- Recursive algorithms
- When merging schedules from several subtrees, divide schedules into sequence of segments
- Each segment has a hill and valley cost (valley=end)
- Sort sequences by decreasing hill − valley value
- Add root, recompute segments (some of them are merged)

segments (hill, valley):

- in C: \([A, B](10, 1), [C](5, 5)\)
- in E: \([D, E](5, 1)\)
- in F: \([A, B](10, 1), [D, E](6, 2), [C, F](6, 6)\)
- in H: \([G, H](8, 4)\)
- whole tree: \([A, B](10, 1), [D, E](6, 2), [G, H](10, 6), [C, F, I](10, 10)\)
Optimal General Traversal

- Recursive algorithms
- When merging schedules from several subtrees, divide schedules into sequence of segments
- Each segment has a hill and valley cost (valley=end)
- Sort sequences by decreasing hill − valley value
- Add root, recompute segments (some of them are merged)

Segments (hill, valley):

- in C: [A, B](10, 1), [C](5, 5)
- in E: [D, E](5, 1)
- in F: [A, B](10, 1), [D, E](6, 2), [C, F](6, 6)
- in H: [G, H](8, 4)
- Whole tree: [A, B](10, 1), [D, E](6, 2), [G, H](10, 6), [C, F, I](10, 10)
Optimal General Traversal

- Recursive algorithms
- When merging schedules from several subtrees, divide schedules into sequence of segments
- Each segment has a *hill* and *valley* cost (valley=end)
- Sort sequences by decreasing *hill* — *valley* value
- Add root, recompute segments (some of them are merged)

segments (hill, valley):
- in C: [A, B](10, 1), [C](5, 5)
- in E: [D, E](5, 1)
  - in F: [A, B](10, 1), [D, E](6, 2), [C, F](6, 6)
  - in H: [G, H](8, 4)
- whole tree: [A, B](10, 1), [D, E](6, 2), [G, H](10, 6), [C, F, I](10, 10)
Optimal General Traversal

- Recursive algorithms
- When merging schedules from several subtrees, divide schedules into sequence of segments
- Each segment has a hill and valley cost (valley=end)
- Sort sequences by decreasing hill − valley value
- Add root, recompute segments (some of them are merged)

segments (hill, valley):
- in C: [A, B](10, 1), [C](5, 5)
- in E: [D, E](5, 1)
- in F: [A, B](10, 1), [D, E](6, 2), [C, F](6, 6)
- in H: [G, H](8, 4)
- whole tree: [A, B](10, 1), [D, E](6, 2), [G, H](10, 6), [C, F, I](10, 10)
Optimal General Traversal

- Recursive algorithms
- When merging schedules from several subtrees, divide schedules into sequence of segments
- Each segment has a hill and valley cost (valley=end)
- Sort sequences by decreasing hill − valley value
- Add root, recompute segments (some of them are merged)

segments (hill, valley):

- in C: [A, B](10, 1), [C](5, 5)
- in E: [D, E](5, 1)
- in F: [A, B](10, 1), [D, E](6, 2), [C, F](6, 6)
- in H: [G, H](8, 4)
- whole tree: [A, B](10, 1), [D, E](6, 2), [G, H](10, 6), [C, F, I](10, 10)
Optimal General Traversal

- Recursive algorithms
- When merging schedules from several subtrees, divide schedules into sequence of segments
- Each segment has a hill and valley cost (valley=end)
- Sort sequences by decreasing hill − valley value
- Add root, recompute segments (some of them are merged)

Segments (hill, valley):
- in C: [A, B](10, 1), [C](5, 5)
- in E: [D, E](5, 1)
- in F: [A, B](10, 1), [D, E](6, 2), [C, F](6, 6)
- in H: [G, H](8, 4)
- Whole tree: [A, B](10, 1), [D, E](6, 2), [G, H](10, 6), [C, F, I](10, 10)
Outline

Tree-Shaped Task graphs, Single Machine, Memory Minimization
   Foundations: Register Allocation & Pebble Game
   Liu’s algorithms

Extensions on Single Machine
   I/O Minimization
   Series-Parallel Task-Graphs

Extensions on Parallel Machines
   Sequential Tasks, Parallel Machine
   Malleable Tasks
   Hybrid Scheduling
I/O Minimization

- Memory is too limited: out-of-core execution
- Data written to disk, and then read
- Goal: given a memory $M$, minimize the total volume of I/O

- Optimal post-order scheme [Agullo, Guermouche, L’Excellent]
  - In a post-order traversal, always write oldest file (to be used the latest)
- If no paging (entire files must be written/read), finding optimal traversal is NP-complete (or optimal post-order)
- Optimal traversal with paging: open problem
Series-Parallel Graphs: Motivation

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

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

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

Diagram:
- Node 1
- Series-Parallel graph $SP_1$ and $SP_2$
Series-Parallel Graphs: Motivation

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

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

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

Tree-Shaped Task graphs, Single Machine, Memory Minimization
  Foundations: Register Allocation & Pebble Game
  Liu’s algorithms

Extensions on Single Machine
  I/O Minimization
  Series-Parallel Task-Graphs

Extensions on Parallel Machines
  Sequential Tasks, Parallel Machine
  Malleable Tasks
  Hybrid Scheduling
Sequential Tasks, Parallel Machine: Complexity

- $p$ uniform processors
- Shared memory of size $M$
- Task $i$ has execution times $p_i$
- Parallel processing of nodes $\Rightarrow$ larger memory
- Trade-off time vs. memory

Complexity results:
- NP-completeness of the bi-objective problem in a homogeneous model (pebble game model)
- No constant factor approximation for both makespan and memory
Sequential Tasks, Parallel Machine: Heuristics

Simple heuristics:

- \texttt{ParInnerFirst}: Post-Order in Parallel
- \texttt{ParDeepestFirst}: Approach Optimal Makespan
- \texttt{ParSubtrees}: Coarse-Grain Parallelism

Memory-bounded heuristics:

- Strong assumptions on the tree (reduction tree)
- Memory condition checked when processing a new leaf
- $2M$ memory guarantee for list-scheduling heuristics (given feasible memory $M$)
- Complex memory-booking scheme
Malleable Tasks

See Bertrand’s presentation
Hybrid Scheduling

- Hybrid system with two memories (CPU+GPU)
- Tasks typed with a specific memory (strong affinities)
- Minimizing both peak memories is NP-complete
- Optimal post-order is polynomial

- Generalization: DAG
- Tasks with two different processing times
- Adaptation of the HEFT heuristic