Scheduling tree-shaped task graphs to minimize memory and makespan

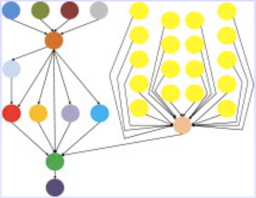
Lionel Eyraud-Dubois (INRIA, Bordeaux, France),
Loris Marchal (CNRS, Lyon, France),
Oliver Sinnen (Univ. Auckland, New Zealand),
Frédéric Vivien (INRIA, Lyon, France)

CCDSC 2014
September 5, 2014

Introduction

Task graph scheduling

- Application modeled as a graph
- Map tasks on processors and schedule them
- Usual performance metric: makespan (time)



Today: focus on memory

- Workflows with large temporary data
- Bad evolution of perf. for computation vs. communication: $1/\text{Flops} \ll 1/\text{bandwidth} \ll \text{latency}$
- Gap between processing power and communication cost increasing exponentially

	annual improvements
Flops rate	59%
mem. bandwidth	26%
mem. latency	5%

- Avoid communications
- Restrict to in-core memory (out-of-core is expensive)

Focus on Task Trees

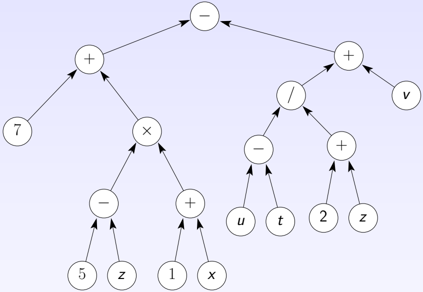
Motivation:

- Arise in multifrontal sparse matrix factorization
- Assembly/Elimination tree: application task graph is a tree
- Large temporary data
- Memory usage becomes a bottleneck

Related Work: 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:

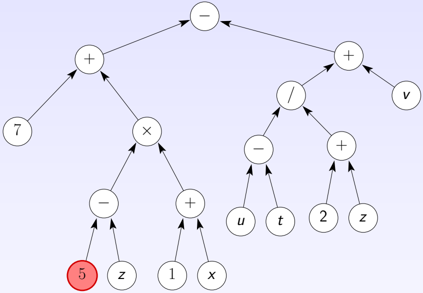
- Inputs can be pebbled anytime
- If all ancestors are pebbled, a node can be pebbled
- A pebble may be removed anytime

Objective: pebble root node using minimum number of pebbles

Related Work: 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:

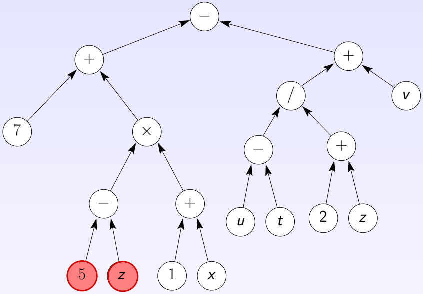
- Inputs can be pebbled anytime
- If all ancestors are pebbled, a node can be pebbled
- A pebble may be removed anytime

Objective: pebble root node using minimum number of pebbles

Related Work: 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:

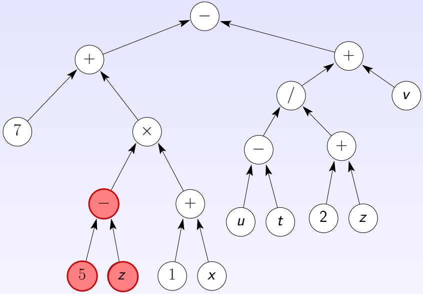
- Inputs can be pebbled anytime
- If all ancestors are pebbled, a node can be pebbled
- A pebble may be removed anytime

Objective: pebble root node using minimum number of pebbles

Related Work: 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:

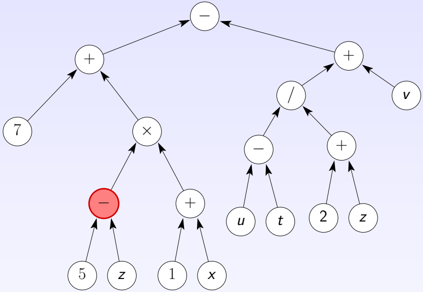
- Inputs can be pebbled anytime
- If all ancestors are pebbled, a node can be pebbled
- A pebble may be removed anytime

Objective: pebble root node using minimum number of pebbles

Related Work: 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

Related Work: Register Allocation & Pebble Game

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

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

Complexity results

Problem on trees:

- Polynomial algorithm [Sethi & Ullman, 1970]

General problem on DAGs (common subexpressions):

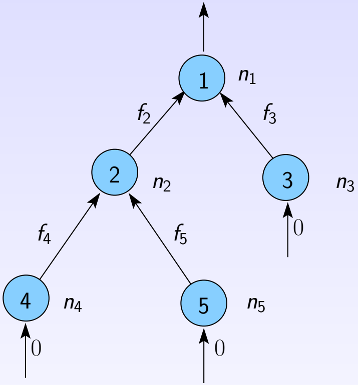
- P-Space complete [Gilbert, Lengauer & Tarjan, 1980]
- Without re-computation: NP-complete [Sethi, 1973]

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

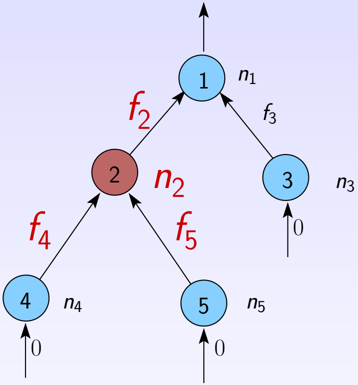
Notations: Tree-Shaped Task Graphs



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

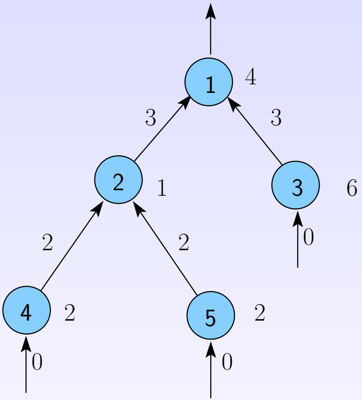
Notations: Tree-Shaped Task Graphs



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

Impact of Schedule on Memory Peak

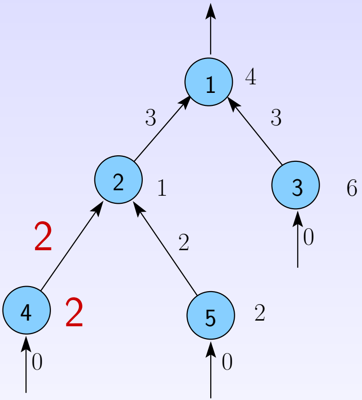


Peak memory so far:

Two existing optimal sequential schedules:

- Best traversal [J. Liu, 1987]
- Best post-order traversal [J. Liu, 1986]

Impact of Schedule on Memory Peak

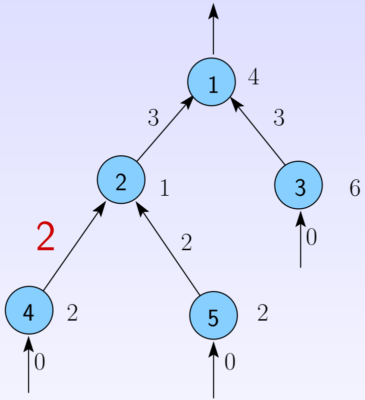


Peak memory so far: 4

Two existing optimal sequential schedules:

- Best traversal [J. Liu, 1987]
- Best post-order traversal [J. Liu, 1986]

Impact of Schedule on Memory Peak

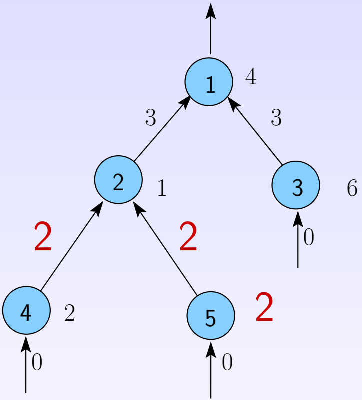


Peak memory so far: 4

Two existing optimal sequential schedules:

- Best traversal [J. Liu, 1987]
- Best post-order traversal [J. Liu, 1986]

Impact of Schedule on Memory Peak

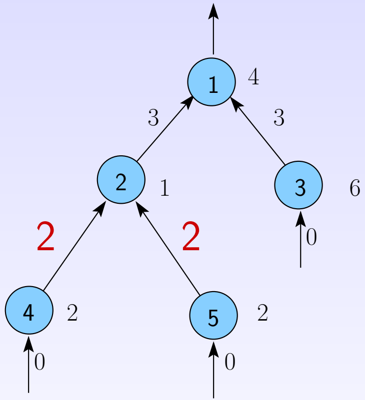


Peak memory so far: 6

Two existing optimal sequential schedules:

- Best traversal [J. Liu, 1987]
- Best post-order traversal [J. Liu, 1986]

Impact of Schedule on Memory Peak

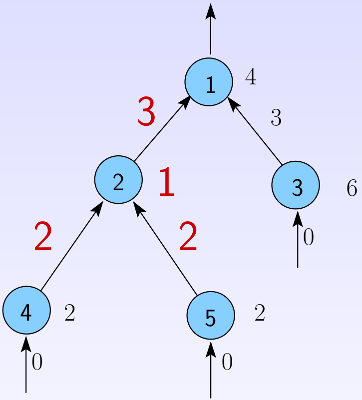


Peak memory so far: 6

Two existing optimal sequential schedules:

- Best traversal [J. Liu, 1987]
- Best post-order traversal [J. Liu, 1986]

Impact of Schedule on Memory Peak

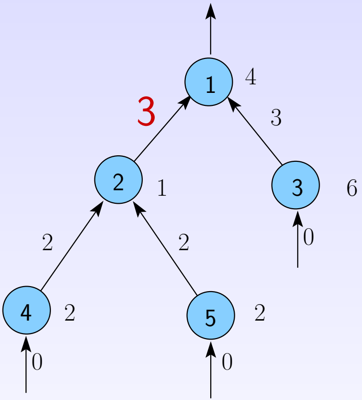


Peak memory so far: 8

Two existing optimal sequential schedules:

- Best traversal [J. Liu, 1987]
- Best post-order traversal [J. Liu, 1986]

Impact of Schedule on Memory Peak

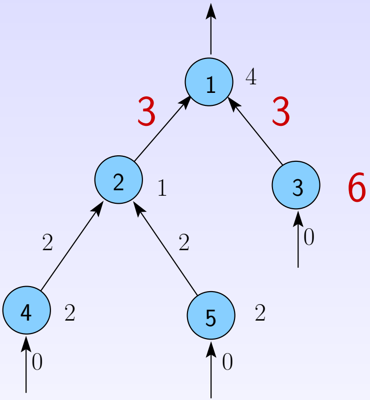


Peak memory so far: 8

Two existing optimal sequential schedules:

- Best traversal [J. Liu, 1987]
- Best post-order traversal [J. Liu, 1986]

Impact of Schedule on Memory Peak

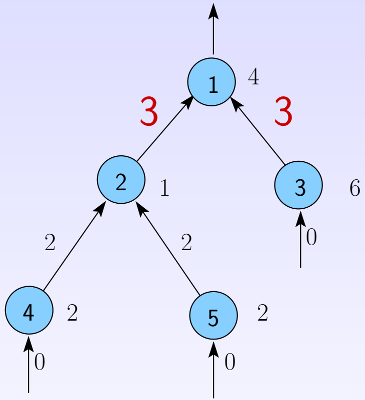


Peak memory so far: 12

Two existing optimal sequential schedules:

- Best traversal [J. Liu, 1987]
- Best post-order traversal [J. Liu, 1986]

Impact of Schedule on Memory Peak

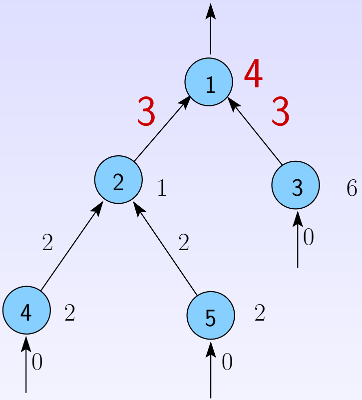


Peak memory so far: 12

Two existing optimal sequential schedules:

- Best traversal [J. Liu, 1987]
- Best post-order traversal [J. Liu, 1986]

Impact of Schedule on Memory Peak

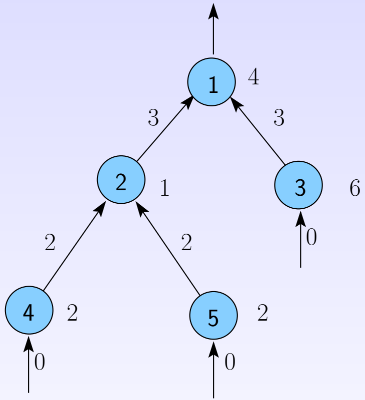


Peak memory so far: 12

Two existing optimal sequential schedules:

- Best traversal [J. Liu, 1987]
- Best post-order traversal [J. Liu, 1986]

Impact of Schedule on Memory Peak

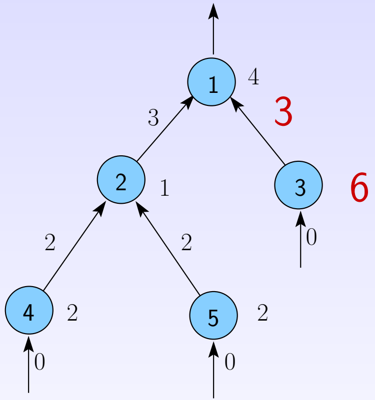


Peak memory so far:

Two existing optimal sequential schedules:

- Best traversal [J. Liu, 1987]
- Best post-order traversal [J. Liu, 1986]

Impact of Schedule on Memory Peak

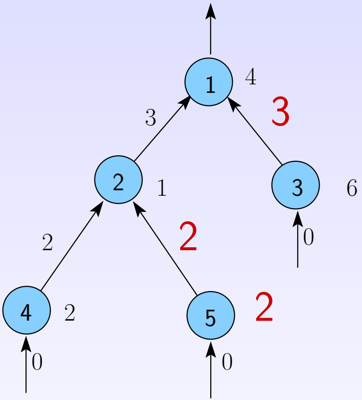


Peak memory so far: 9

Two existing optimal sequential schedules:

- Best traversal [J. Liu, 1987]
- Best post-order traversal [J. Liu, 1986]

Impact of Schedule on Memory Peak

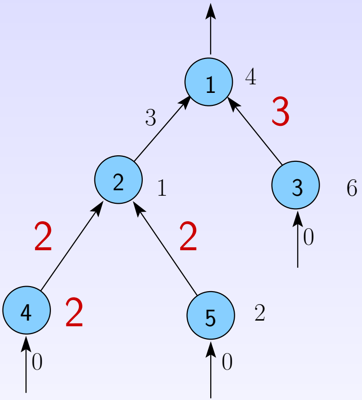


Peak memory so far: 9

Two existing optimal sequential schedules:

- Best traversal [J. Liu, 1987]
- Best post-order traversal [J. Liu, 1986]

Impact of Schedule on Memory Peak

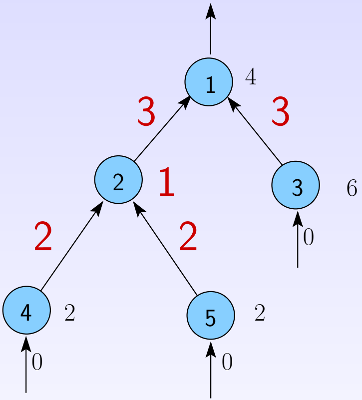


Peak memory so far: 9

Two existing optimal sequential schedules:

- Best traversal [J. Liu, 1987]
- Best post-order traversal [J. Liu, 1986]

Impact of Schedule on Memory Peak

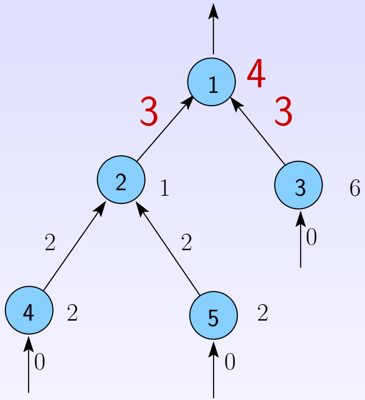


Peak memory so far: 11

Two existing optimal sequential schedules:

- Best traversal [J. Liu, 1987]
- Best post-order traversal [J. Liu, 1986]

Impact of Schedule on Memory Peak

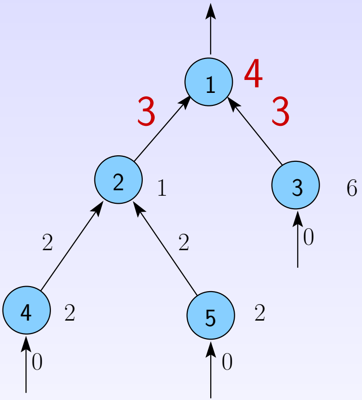


Peak memory so far: 11

Two existing optimal sequential schedules:

- Best traversal [J. Liu, 1987]
- Best post-order traversal [J. Liu, 1986]

Impact of Schedule on Memory Peak



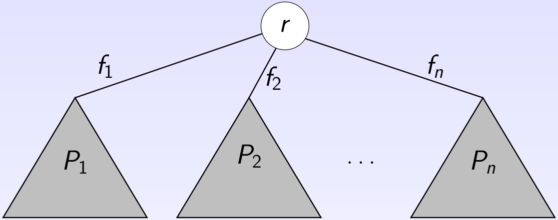
Peak memory so far: 11 (which is better than 12)

Two existing optimal sequential schedules:

- Best traversal [J. Liu, 1987]
- Best post-order traversal [J. Liu, 1986]

Post-Order Traversal for Trees

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



Post-Order traversals are arbitrarily bad in the general case

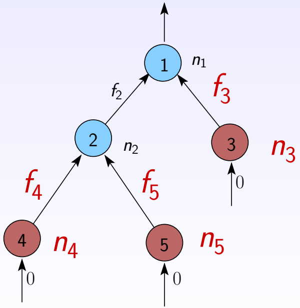
There is no constant k such that the best post-order traversal is a k-approximation.

In practice post-order have very good performance

Outline

Model for Parallel Tree Processing

- p identical 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



NP-Completeness in the Pebble Game Model

Background:

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

Space-Time Tradeoff

Not possible to get a guarantee on both memory and time simultaneously:

Theorem 1

There is no algorithm that is both an α-approximation for makespan minimization and a β-approximation for memory peak minimization when scheduling tree-shaped task graphs.

For a fixed number of processors:

Theorem 2

For any $\alpha(p)$-approximation for makespan and $\beta(p)$-approximation for memory peak with $p \geq 2$ processors,

$$\alpha(p)\beta(p) \;\geq\; \frac{2p}{\lceil \log(p) \rceil + 2}.$$

Outline

InnerFirst: Post-Order in Parallel

Motivation:

- Post-Order behavior: process inner nodes ASAP
- Parallel version: give priority to inner nodes
- Naturally limits the number of concurrent subtrees
- Intuitively good to keep memory low

Implementation as a list-scheduling heuristic

- Put ready nodes in a queue (higher priority for inner nodes)
- Schedule them whenever a processor is ready
- Initially, sort leaf nodes using best sequential post-order

Performance:

- $(2 - 1/p)$-approximation for makespan
- Unbounded ratio for memory
- $O(n \log n)$ complexity

DeepestFirst: Approach Optimal Makespan

DeepestFirst:

- Compute critical path values for all tasks
- List-scheduling based on critical path values

Performance:

- Known as a good heuristic for makespan minimization
- No guarantee (or intuition) on memory behavior
- $O(n \log n)$ complexity

Subtrees: Coarse-Grain Parallelism

Motivation:

- Divide the tree in p large subtrees + small set of other nodes
- Each processor works on its own subtree
- Locally, use memory-optimal sequential algorithm
- Process all remaining nodes sequentially
- Optimization: if more than p subtrees when spliting, load-balance subtrees on processors

Performance:

- $O(n \log n)$ complexity
- p-approximation algorithm for memory

How to Cope with Limited Memory?

Motivation:

- Work with a given quantity of memory
- Optimize makespan under this constraint

Stronger assumptions:

- Reduction tree: $\sum_{j \in Children(i)} f_j \geq f_i$
- No extra memory cost for task execution

Assumptions not verified, but enforced by adding fictitious nodes

How to Cope with Limited Memory?

Motivation:

- Work with a given quantity of memory
- Optimize makespan under this constraint

Stronger assumptions:

- Reduction tree: $\displaystyle\sum_{j \in Children(i)} f_j \geq f_i$
- No extra memory cost for task execution

Assumptions not verified, but enforced by adding fictitious nodes

How to Cope with Limited Memory?

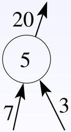
Motivation:

- Work with a given quantity of memory
- Optimize makespan under this constraint

Stronger assumptions:

- Reduction tree: $\sum_{j \in Children(i)} f_j \geq f_i$
- No extra memory cost for task execution

Assumptions not verified, but enforced by adding fictitious nodes



How to Cope with Limited Memory?

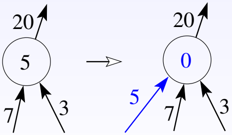
Motivation:

- Work with a given quantity of memory
- Optimize makespan under this constraint

Stronger assumptions:

- Reduction tree: $\sum_{j \in Children(i)} f_j \geq f_i$
- No extra memory cost for task execution

Assumptions not verified, but enforced by adding fictitious nodes



How to Cope with Limited Memory?

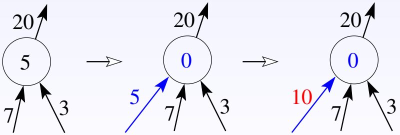
Motivation:

- Work with a given quantity of memory
- Optimize makespan under this constraint

Stronger assumptions:

- Reduction tree: $\sum_{j \in Children(i)} f_j \geq f_i$
- No extra memory cost for task execution

Assumptions not verified, but enforced by adding fictitious nodes



Memory-Bounded Heuristics: Simple Way

First idea: restrain List-Scheduling heuristics (InnerFirst and DeepestFirst)

- Choose a feasible amount $\frac{M}{2}$ of memory
- Check that memory $\leq \frac{M}{2}$ when starting a new leaf
- Guarantee: Memory used at most M

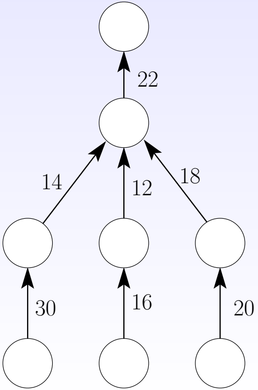
Proof ideas:

- Reduction tree: memory reduced by processing inner nodes
- During the processing: at most twice the input memory

Memory-Bounded Heuristics: Complex Way

Second idea: complex memory booking scheme

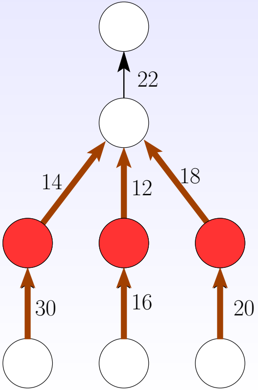
- Book memory for parent nodes, ensure they can be processed later
- Test for memory (booked+used) when starting a leaf
- Never exceeds a given memory M



Memory-Bounded Heuristics: Complex Way

Second idea: complex memory booking scheme

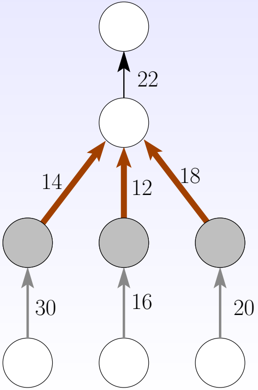
- Book memory for parent nodes, ensure they can be processed later
- Test for memory (booked+used) when starting a leaf
- Never exceeds a given memory M



Memory-Bounded Heuristics: Complex Way

Second idea: complex memory booking scheme

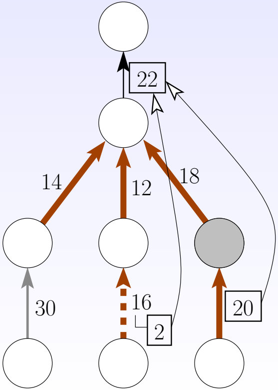
- Book memory for parent nodes, ensure they can be processed later
- Test for memory (booked+used) when starting a leaf
- Never exceeds a given memory M



Memory-Bounded Heuristics: Complex Way

Second idea: complex memory booking scheme

- Book memory for parent nodes, ensure they can be processed later
- Test for memory (booked+used) when starting a leaf
- Never exceeds a given memory M



Outline

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

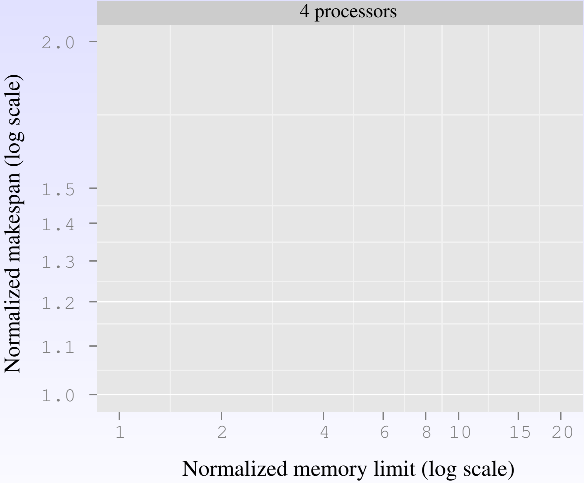
- 2, 4, 8, 16 or 32 processors

Results

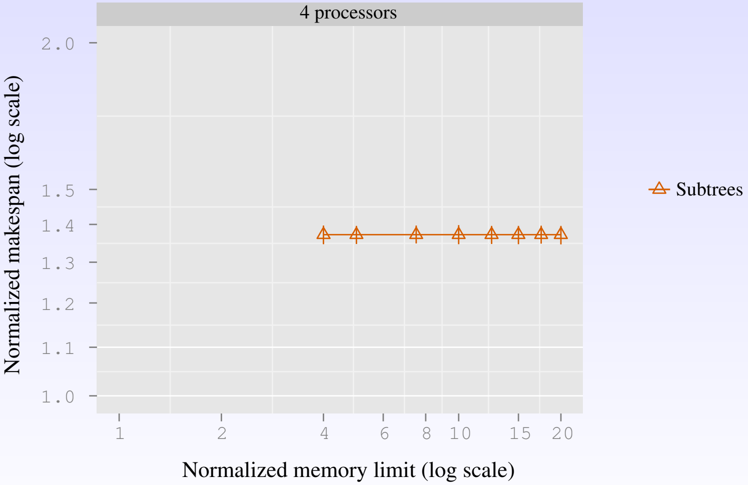
Heuristic	Best memory	Avg. normalized memory needed	Best makespan	Avg. normalized makespan
SUBTREES	81.1 %	2.33	0.2 %	1.35
SUBTREESOPTIM	49.9 %	2.45	1.1 %	1.29
INNERFIRST	19.1 %	3.77	37.2 %	1.03
DEEPESTFIRST	3.0 %	4.26	95.7 %	1.00

- Memory normalized with optimal sequential memory
- Makespan normalized with best makespan

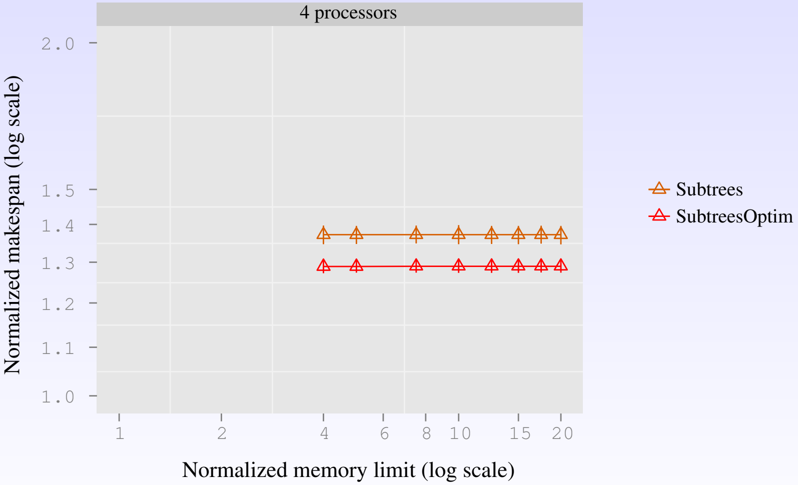
Memory-Aware Heuristics: Makespan vs. Memory



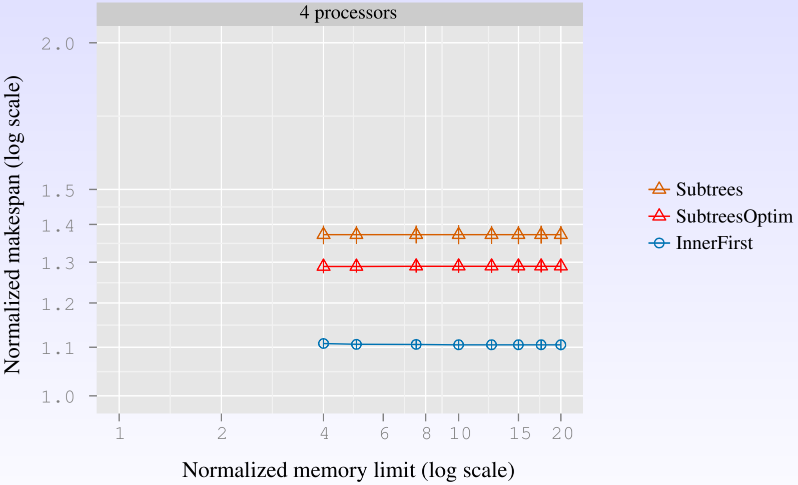
Memory-Aware Heuristics: Makespan vs. Memory



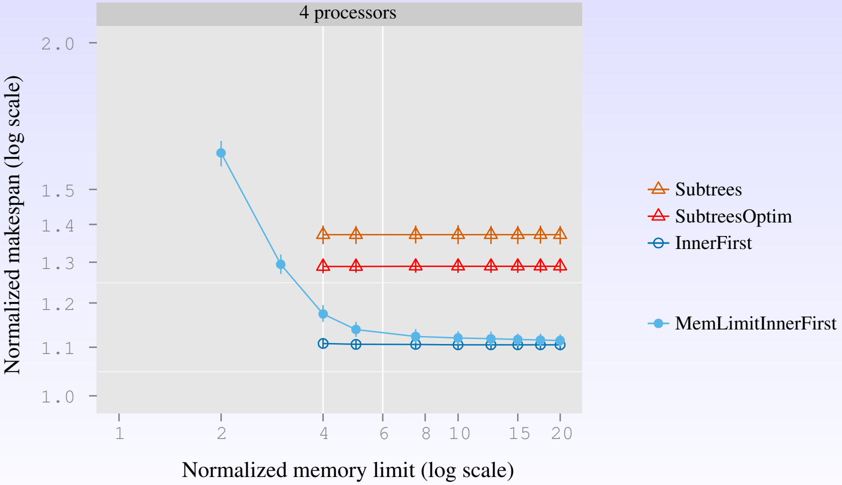
Memory-Aware Heuristics: Makespan vs. Memory



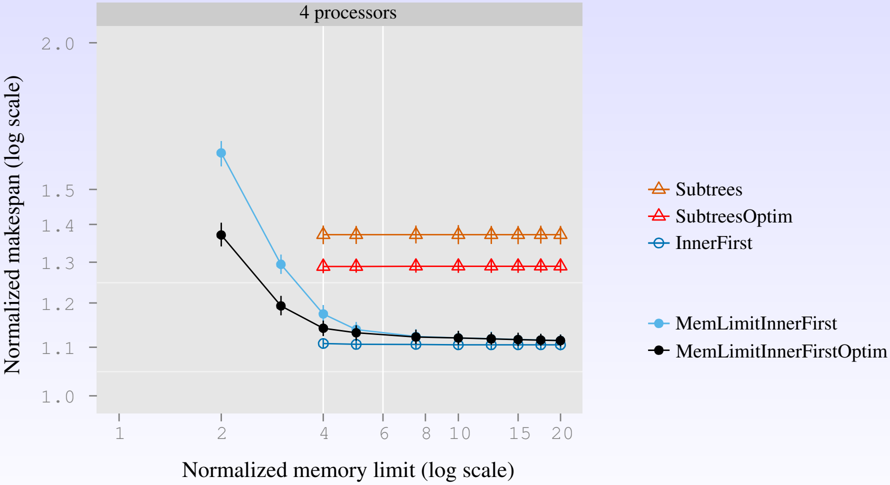
Memory-Aware Heuristics: Makespan vs. Memory



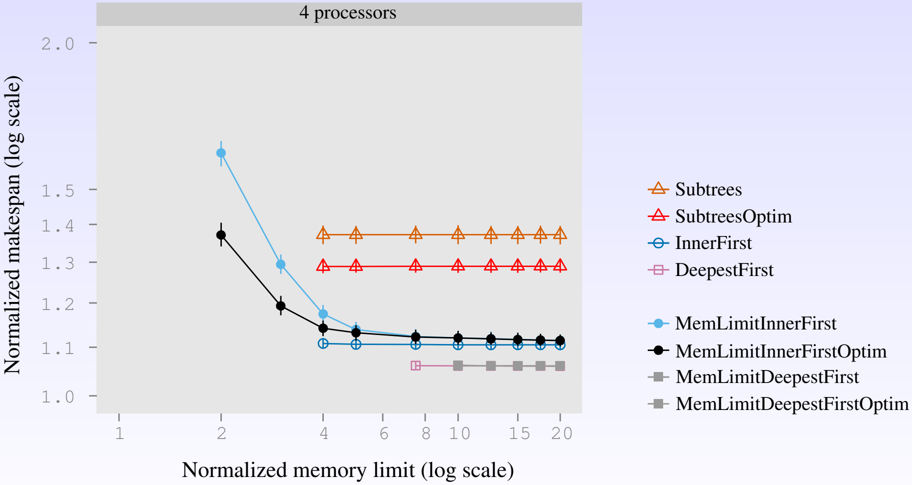
Memory-Aware Heuristics: Makespan vs. Memory



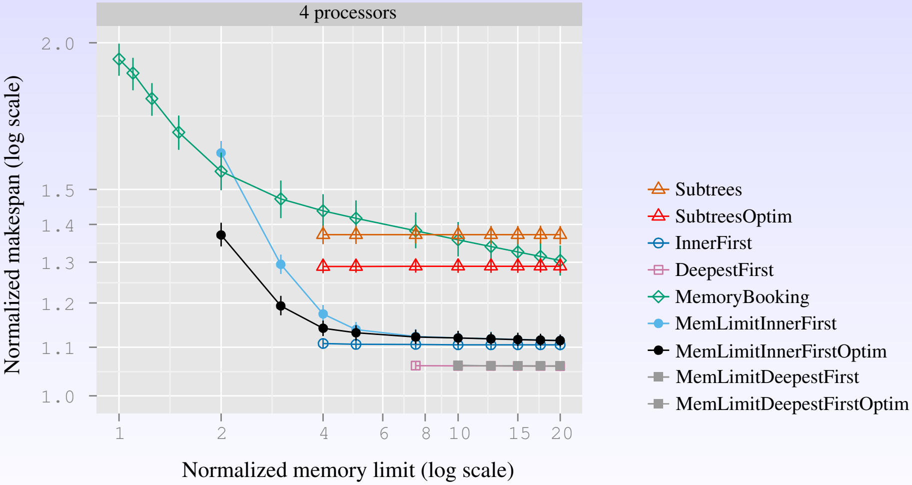
Memory-Aware Heuristics: Makespan vs. Memory



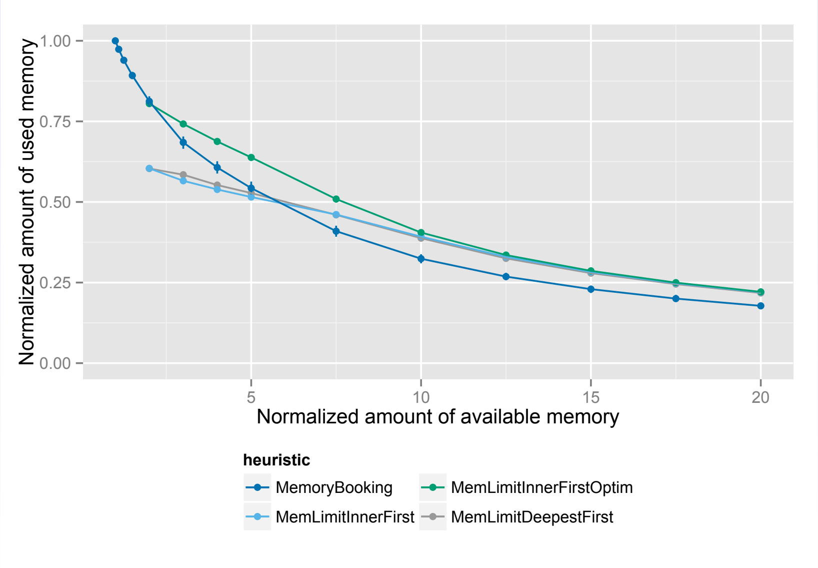
Memory-Aware Heuristics: Makespan vs. Memory



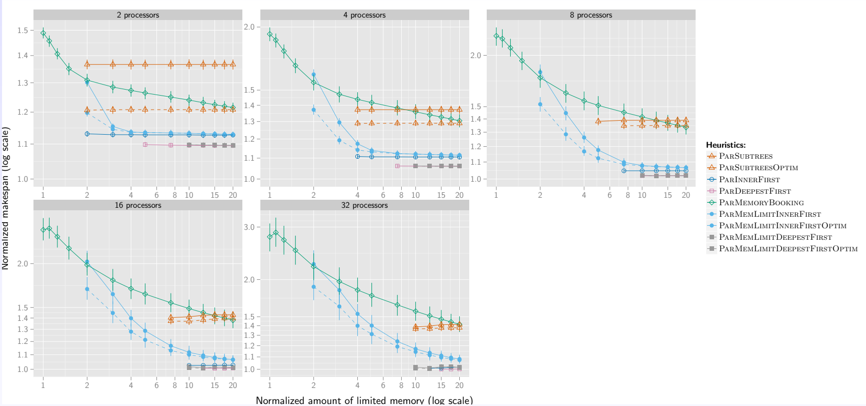
Memory-Aware Heuristics: Makespan vs. Memory



Memory-Aware Heuristics: Memory Usage



Memory-Aware Heuristics: Makespan vs. memory



Outline

Summary and Perspectives

- Complexity study of parallel tree traversals
- Simple heuristics
- Memory-bounded heuristics
- Simulations on real elimination trees

Future work:

- Consider distributed memory
- Extend results to other class of regular graphs ($2D$ grids, etc.)
- Minimize I/O volume for out-of-core execution
- Consider parallel (malleable) tasks

What does the fox really want?

A break!