Part 3: Memory-Aware DAG Scheduling

Minimize Memory for Trees

Minimize Memory for SP-Graphs

Minimize I/Os for Trees

Shared Memory of Parallel Processing

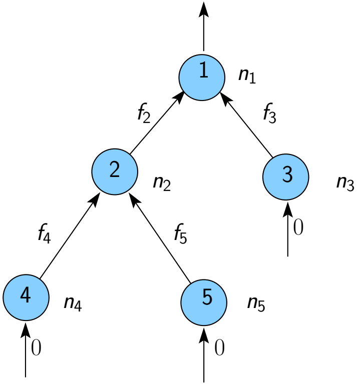
Introduction

- Directed Acyclic Graphs: express task dependencies
  - nodes: computational tasks
  - edges: dependencies
    (data = output of a task = input of another task)
- Formalism proposed long ago in scheduling
- Back into fashion thanks to task based runtimes

Special focus on task trees:

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

Notations: Tree-Shaped Task Graphs



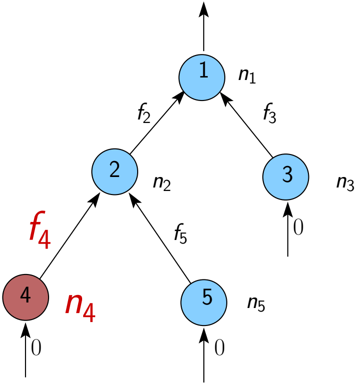
- In-tree of n nodes
- Output data of size f_i
- Execution data of size n_i
- Input data of leaf nodes have null size

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

Two equivalent problems (reverse schedules):

- Bottom-up processing
- Top-down processing

Notations: Tree-Shaped Task Graphs



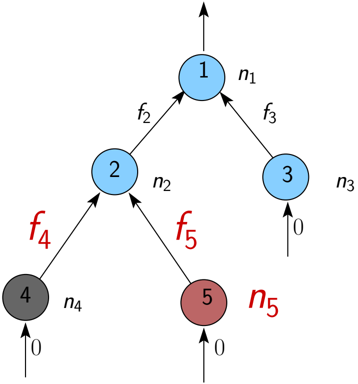
- In-tree of n nodes
- Output data of size f_i
- Execution data of size n_i
- Input data of leaf nodes have null size

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

Two equivalent problems (reverse schedules):

- Bottom-up processing
- Top-down processing

Notations: Tree-Shaped Task Graphs



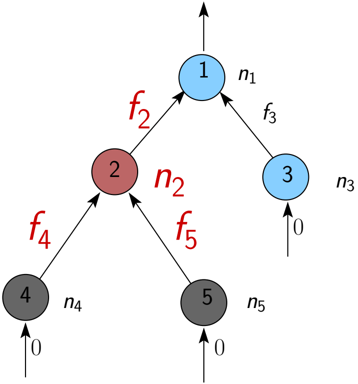
- In-tree of n nodes
- Output data of size f_i
- Execution data of size n_i
- Input data of leaf nodes have null size

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

Two equivalent problems (reverse schedules):

- Bottom-up processing
- Top-down processing

Notations: Tree-Shaped Task Graphs



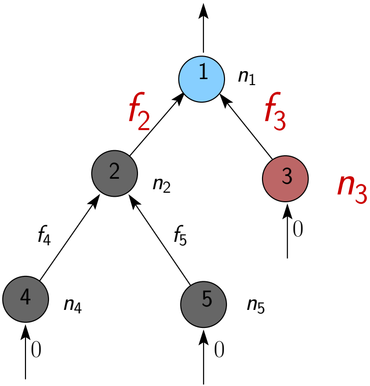
- In-tree of n nodes
- Output data of size f_i
- Execution data of size n_i
- Input data of leaf nodes have null size

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

Two equivalent problems (reverse schedules):

- Bottom-up processing
- Top-down processing

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

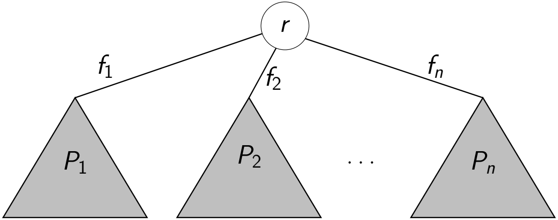
- Memory for node i: $MemReq(i) = \left(\sum_{j \in Children(i)} f_j\right) + n_i + f_i$

Two equivalent problems (reverse schedules):

- Bottom-up processing
- Top-down processing

Liu's Best Post-Order Traversal for Trees

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



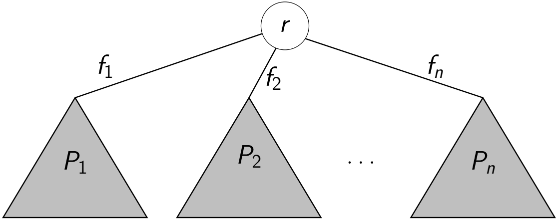
- For each subtree T_i: peak memory P_i, residual memory f_i
- For a given processing order $1, \ldots, n$, the peak memory is:

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

- Optimal order:

Liu's Best Post-Order Traversal for Trees

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



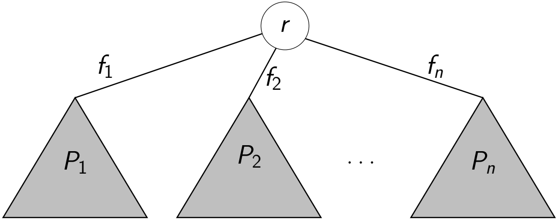
- For each subtree T_i: peak memory P_i, residual memory f_i
- For a given processing order $1, \dots, n$, the peak memory is:

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

- Optimal order:

Liu's Best Post-Order Traversal for Trees

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



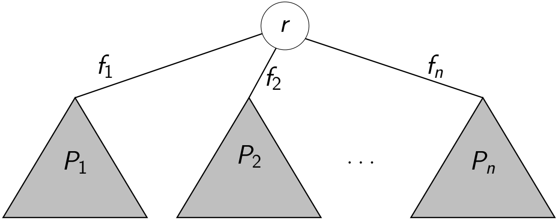
- For each subtree T_i: peak memory P_i, residual memory f_i
- For a given processing order $1, \ldots, n$, the peak memory is:

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

- Optimal order:

Liu's Best Post-Order Traversal for Trees

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



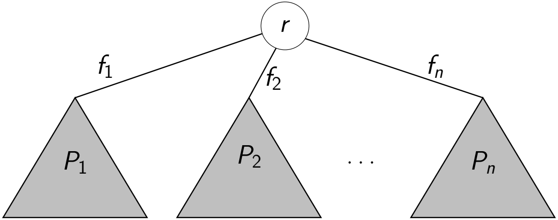
- For each subtree T_i: peak memory P_i, residual memory f_i
- For a given processing order $1, \dots, n$, the peak memory is:

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

- Optimal order:

Liu's Best Post-Order Traversal for Trees

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



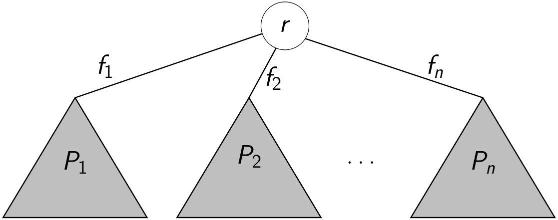
- For each subtree T_i: peak memory P_i, residual memory f_i
- For a given processing order $1, \ldots, n$, the peak memory is:

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

- Optimal order:

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 $1, \dots, n$, the peak memory is:

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

- Optimal order: non-increasing $P_i - f_i$

Proof for best post-order

Theorem (Best Post-Order).

The best post-order traversal is obtain by processing subtrees in non-increasing order $P_i - f_i$.

Proof:

- Consider an optimal traversal which does not respect the order:
  - subtree j is processed right before subtree k
  - $P_k - f_k \geq P_j - f_j$

	peak when j, then k	peak when k, then j
during first subtree	$mem\_before + P_j$	$mem\_before + P_k$
during second subtree	$mem\_before + f_j + P_k$	$mem\_before + f_k + P_j$

- $f_k + P_j \leq f_j + P_k$
- Transform the schedule step by step without increasing the memory.

Proof for best post-order

Theorem (Best Post-Order).

The best post-order traversal is obtain by processing subtrees in non-increasing order $P_i - f_i$.

Proof:

- Consider an optimal traversal which does not respect the order:
  - subtree j is processed right before subtree k
  - $P_k - f_k \geq P_j - f_j$

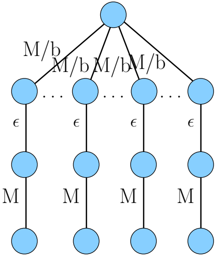
	peak when j, then k	peak when k, then j
during first subtree	$mem\_before + P_j$	$mem\_before + P_k$
during second subtree	$mem\_before + f_j + P_k$	$mem\_before + f_k + P_j$

- $f_k + P_j \leq f_j + P_k$
- Transform the schedule step by step without increasing the memory.

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.



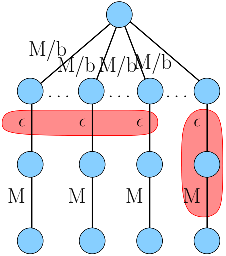
- Minimum peak memory: $M_{\min} = M + \epsilon + (b-1)\epsilon$
- Minimum post-order peak memory: $M_{\min} = M + \epsilon + (b-1)M/b$

	actual assembly trees	random trees
Non optimal traversals	4.2%	61%
Maximum increase compared to optimal	18%	22%
Average increased compared to optimal	1%	12%

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.



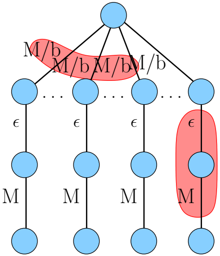
- Minimum peak memory: $M_{\min} = M + \epsilon + (b-1)\epsilon$
- Minimum post-order peak memory: $M_{\min} = M + \epsilon + (b-1)M/b$

	actual assembly trees	random trees
Non optimal traversals	4.2%	61%
Maximum increase compared to optimal	18%	22%
Average increased compared to optimal	**1%**	12%

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.



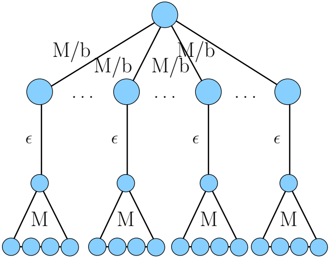
- Minimum peak memory:
  $M_{\text{min}} = M + \epsilon + (b-1)\epsilon$
- Minimum post-order peak memory:
  $M_{\text{min}} = M + \epsilon + (b-1)M/b$

	actual assembly trees	random trees
Non optimal traversals	4.2%	61%
Maximum increase compared to optimal	18%	22%
Average increased compared to optimal	**1%**	12%

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.



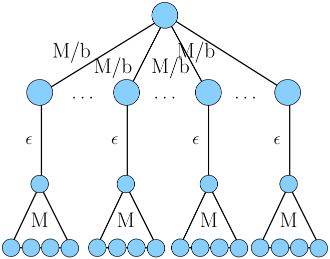
- Minimum peak memory: $M_{\min} = M + \epsilon + 2(b-1)\epsilon$
- Minimum post-order peak memory: $M_{\min} = M + \epsilon + 2(b-1)M/b$

	actual assembly trees	random trees
Non optimal traversals	4.2%	61%
Maximum increase compared to optimal	18%	22%
Average increased compared to optimal	**1%**	12%

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

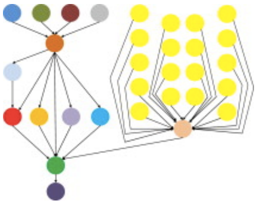
	actual assembly trees	random trees
Non optimal traversals	4.2%	61%
Maximum increase compared to optimal	18%	22%
Average increased compared to optimal	**1%**	12%

Liu's optimal traversal – sketch

- Recursive algorithm: at each step, merge the optimal ordering of each subtree (sequence)
- Sequence: divided into segments:
  - H_1: maximum over the whole sequence (hill)
  - V_1: minimum after H_1 (valley)
  - H_2: maximum after H_1
  - V_2: minimum after H_2
  - ...
  - The valleys V_is are the boundaries of the segments
- Combine the sequences by non-increasing $H - V$
- Complex proof based on a partial order on the cost-sequences: $(H_1, V_1, H_2, V_2, \ldots, H_r, V_r) \prec (H'_1, V'_1, H'_2, V'_2, \ldots, H'_{r'}, V'_{r'})$ if for each $1 \leq i \leq r$, there exists $1 \leq j \leq r'$ with $H_i \leq H'_j$ and $V_i \leq V'_j$.

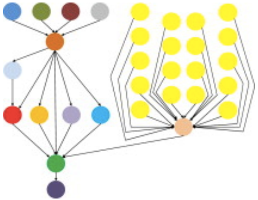
Series-Parallel Graphs: Motivation

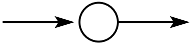
- Not all scientific workflows are trees
- But most workflows exhibit some regularity
- Large class of workflows: Series-Parallel graphs



Series-Parallel Graphs: Motivation

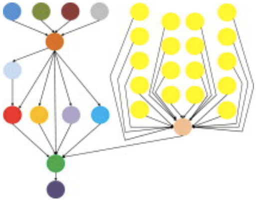
- Not all scientific workflows are trees
- But most workflows exhibit some regularity
- Large class of workflows: Series-Parallel graphs

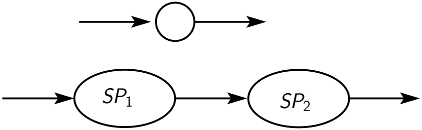




Series-Parallel Graphs: Motivation

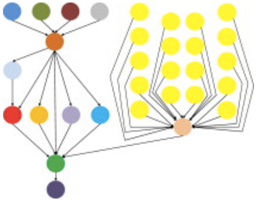
- Not all scientific workflows are trees
- But most workflows exhibit some regularity
- Large class of workflows: Series-Parallel graphs

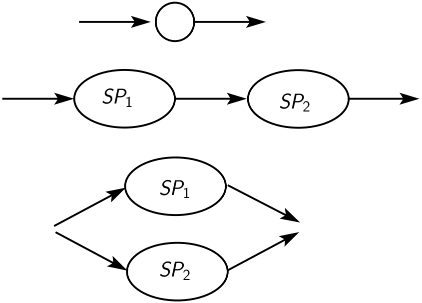




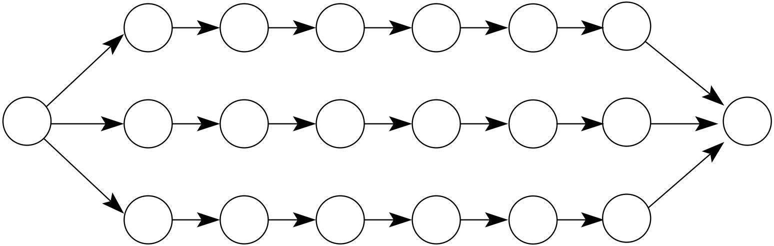
Series-Parallel Graphs: Motivation

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





First Step: Parallel-Chain Graphs



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

Theorem

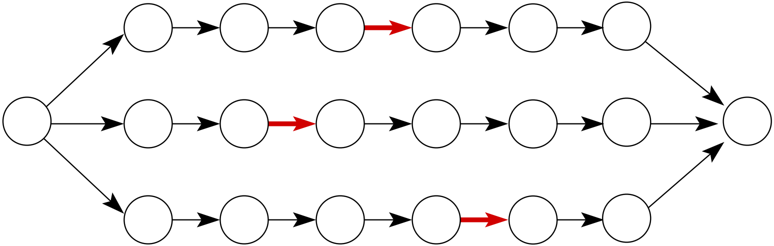
There exists a schedule with minimal memory which synchronises at $e_1, \ldots, e_B$.

Algorithm:

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

(Technicality: set weights of e_i to zero)

First Step: Parallel-Chain Graphs



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

Theorem

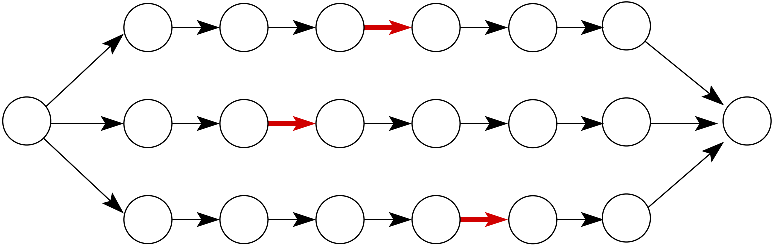
There exists a schedule with minimal memory which synchronises at $e_1, \ldots, e_B$.

Algorithm:

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

(Technicality: set weights of e_i to zero)

First Step: Parallel-Chain Graphs



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

Theorem

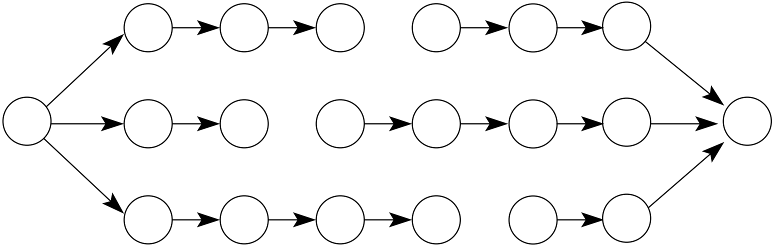
There exists a schedule with minimal memory which synchronises at $e_1, \ldots, e_B$.

Algorithm:

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

(Technicality: set weights of e_i to zero)

First Step: Parallel-Chain 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

(Technicality: set weights of e_i to zero)

General Series-Parallel Graphs

Principle:

- Follow the recursive definition of the SP-graph
- Compute both optimal schedule and minimal cut
- Replace subgraphs by chains of nodes (based on opt. sched.)

For sequential composition:

- Select minimal cut
- Concatenate schedules

For parallel composition (as for Parallel-Chains):

- Merge cuts
- On the left part, use algo. for out-trees for merging schedules
- On the right, use algo. for in-trees for merging schedules

Simple algorithm vs. very complex proof of optimality

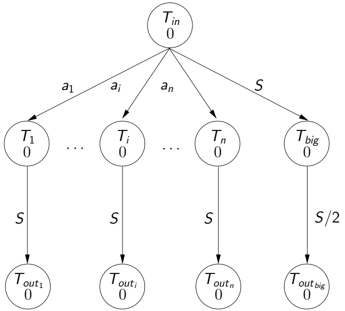
Minimizing I/Os for Trees

Problem:

- Amount of available memory M is to small to compute the whole tree
- Some data needs to be written to disk, and read back later
- Objective: minimize the amount of I/Os (total volume)

Theorem.

When data must be fully written to disk, deciding which data to write to disk is NP-complete.



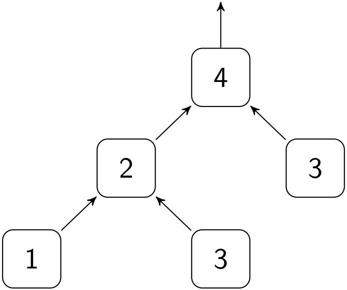
$M = 2S$

Minimizing I/O for Trees – with Paging

With paging:

- Partial data may be written to disk
- Simpler model: memory weight only on edges
  output of $i = w_i$ (original model by Liu)
- When processing a node, max(input, output) is needed
- I/O cost metric: volume of data written to disk

Example with $M = 5$:



Memory: 0 / 5

Disk: 0

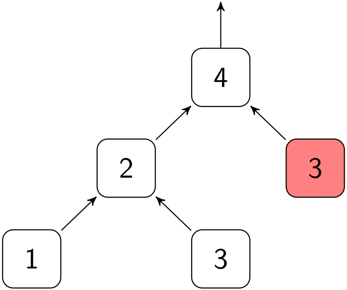
I/Os: 0

Minimizing I/O for Trees – with Paging

With paging:

- Partial data may be written to disk
- Simpler model: memory weight only on edges
  output of $i = w_i$ (original model by Liu)
- When processing a node, max(input, output) is needed
- I/O cost metric: volume of data written to disk

Example with $M = 5$:



Memory: 3 / 5

Disk: 0

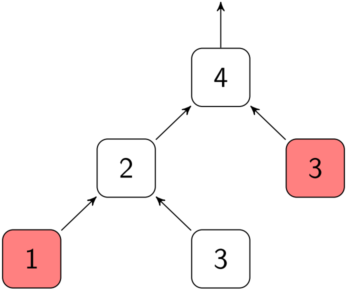
I/Os: 0

Minimizing I/O for Trees – with Paging

With paging:

- Partial data may be written to disk
- Simpler model: memory weight only on edges
  output of $i = w_i$ (original model by Liu)
- When processing a node, max(input, output) is needed
- I/O cost metric: volume of data written to disk

Example with $M = 5$:



Memory: 4 / 5

Disk: 0

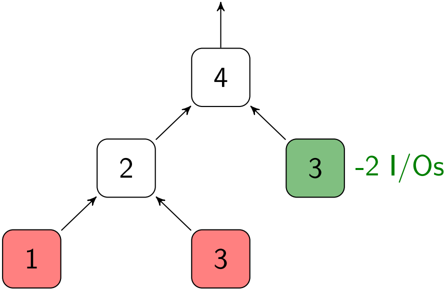
I/Os: 0

Minimizing I/O for Trees – with Paging

With paging:

- Partial data may be written to disk
- Simpler model: memory weight only on edges
  output of $i = w_i$ (original model by Liu)
- When processing a node, max(input, output) is needed
- I/O cost metric: volume of data written to disk

Example with $M = 5$:



Memory: 5 / 5

Disk: 2

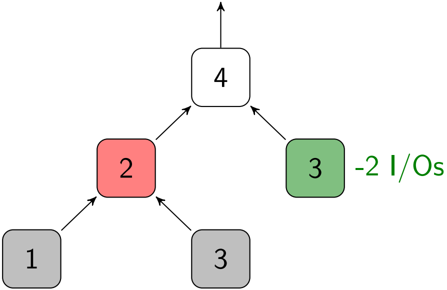
I/Os: 2

Minimizing I/O for Trees – with Paging

With paging:

- Partial data may be written to disk
- Simpler model: memory weight only on edges
  output of $i = w_i$ (original model by Liu)
- When processing a node, max(input, output) is needed
- I/O cost metric: volume of data written to disk

Example with $M = 5$:



Memory: 3 / 5

Disk: 2

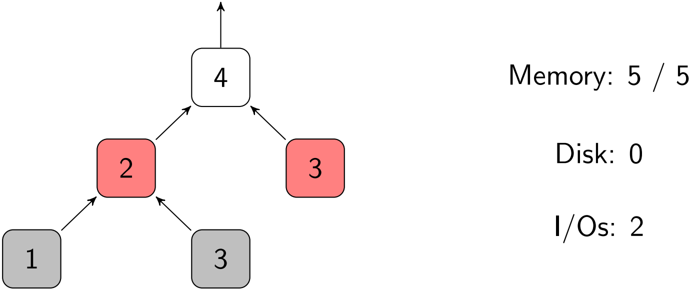
I/Os: 2

Minimizing I/O for Trees – with Paging

With paging:

- Partial data may be written to disk
- Simpler model: memory weight only on edges
  output of $i = w_i$ (original model by Liu)
- When processing a node, max(input, output) is needed
- I/O cost metric: volume of data written to disk

Example with $M = 5$:

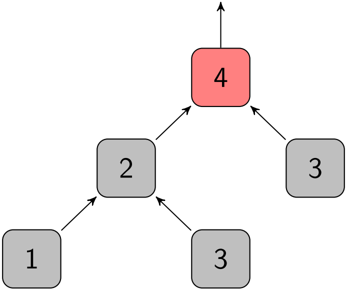


Minimizing I/O for Trees – with Paging

With paging:

- Partial data may be written to disk
- Simpler model: memory weight only on edges
  output of $i = w_i$ (original model by Liu)
- When processing a node, max(input, output) is needed
- I/O cost metric: volume of data written to disk

Example with $M = 5$:



Memory: 4 / 5

Disk: 0

I/Os: 2

Description of a solution

Traversal

- Schedule σ: $\sigma(i) = t$ if task i is the t- th executed
- I/O function τ: output data of task i has $\tau(i)$ slots written to disk
- W.l.o.g. data written to disk ASAP and read ALAP

Validity of a traversal

- Schedule respects precedences
- I/Os consistent: $\tau(i) \leq w_i$
- The main memory (size M) is never exceeded, $\forall i \in V$:

$$\left(\sum_{\substack{(k,p)\in E \\ \sigma(k)<\sigma(i)<\sigma(p)}} (w_k - \tau(k)) \right) + \max\left(w_i, \sum_{(j,i)\in E} w_j \right) \leq M$$

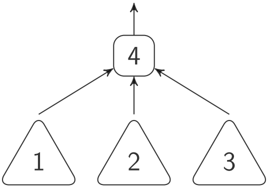
Objective

The MinIO problem

Given a tree G and a memory limit M, find a valid traversal that minimizes the total amount of I/Os ($= \sum \tau(i)$).

An interesting subclass: postorder traversals

- Fully process a subtree before starting a new one
- Completely characterized by the execution order of subtrees
- Widely used in sparse matrix softwares (e.g., MUMPS, QR-MUMPS)



Preliminary results

Let (σ, τ) be an optimal traversal for MinIO of a given instance

Lemma (Schedule is enough).

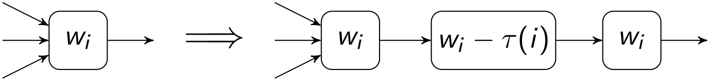
Given σ: the *Furthest In the Future* I/O policy minimizes I/Os.

Lemma (I/O function is enough).

Given τ: a valid traversal (σ', τ) can be computed in polynomial time.

Proof.

Expand each node following:

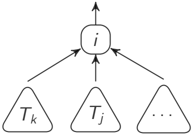


Then minimize the memory peak. □

Postorder algorithms [Liu 1986, Agullo et al. 2010]

- When executing T_i : order of execution of children of i
- First compute the storage requirement of subtree T_i:

$$S_i = \max\left(w_i\ ,\ \max_{j \in Chil(i)}\left(S_j + \sum_{\substack{k \in Chil(i)\\ \sigma(k) < \sigma(j)}} w_k\right)\right)$$



- Memory really used: $A_i = \min(S_i, M)$
- For a given order σ, the volume of I/O is given by:

$$V_i = \max\left(0, \max_{j \in Chil(i)}\left(A_j + \sum_{\substack{k \in Chil(i)\\ \sigma(k) < \sigma(j)}} w_k\right) - M\right) + \sum_{j \in Chil(i)} V_j$$

Best Postorder for Minimizing I/Os

For a given order σ, the volume of I/O is given by:

$$V_i = \max\left(0, \max_{j \in Chil(i)}\left(A_j + \sum_{\substack{k \in Chil(i) \\ \sigma(k) < \sigma(j)}} w_k\right) - M\right) + \sum_{j \in Chil(i)} V_j$$

Theorem.

Given a set of values (x_i, y_i), the minimum of $\max(x_i + \sum_{j<i} y_j)$ is obtained by sorting the sequence by decreasing $x_i - y_i$.

Corollary

The postorder traversal that minimizes I/Os sorts the subtree by decreasing $A_j - w_j$.

Minimizing I/Os for Homogeneous Trees

Theorem.

Both PostOrderMinMem and PostOrderMinIO minimize I/Os on homogeneous trees (unit sizes).

Note: PostOrderMinMem does not rely on M so is optimal for any memory size and several memory layers (cache-oblivious)

But PostOrderMinIO is not competitive on heterogeneous trees:

- Cases when PostOrderMinIO needs I/O why optimal traversal does not
- Even in when the optimal traversal requires I/Os...

Minimizing I/Os for Homogeneous Trees

Theorem.

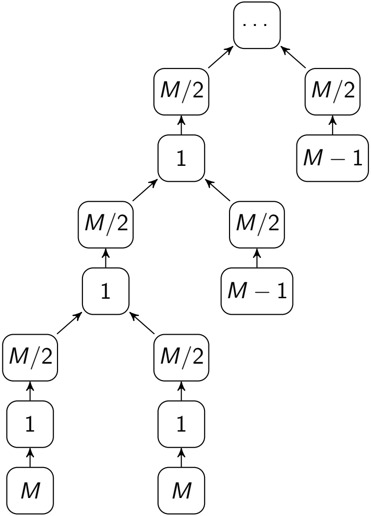
Both PostOrderMinMem and PostOrderMinIO minimize I/Os on homogeneous trees (unit sizes).

Note: PostOrderMinMem does not rely on M so is optimal for any memory size and several memory layers (cache-oblivious)

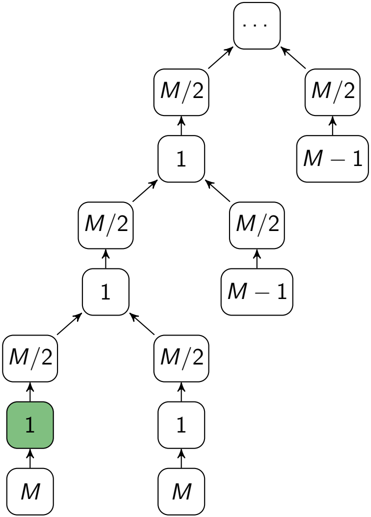
But PostOrderMinIO is not competitive on heterogeneous trees:

- Cases when PostOrderMinIO needs I/O why optimal traversal does not
- Even in when the optimal traversal requires I/Os...

PostOrderMinIO is not competitive



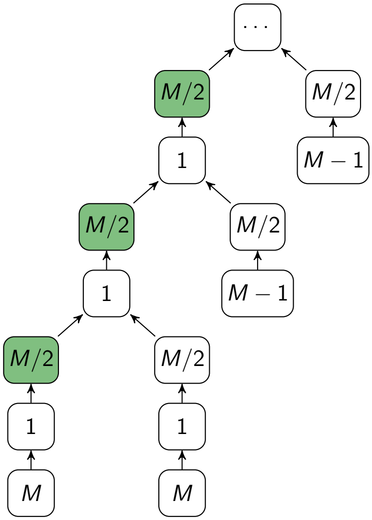
PostOrderMinIO is not competitive



I/O optimal

- Peak memory: $M + 1$
- I/Os: 1

PostOrderMinIO is not competitive



I/O optimal

- Peak memory: $M + 1$
- I/Os: 1

PostOrderMinIO

- Peak memory: $\frac{3}{2}M$
- I/Os: $\Theta(|V|M)$

Competitive ratio: $\Omega(|V|M)$

MinIO for Trees – Summary

- PostOrder algorithms optimal for homogeneous trees
- No known competitive algorithms for heterogeneous trees
- Heterogeneous trees: still an open problem!

Part 3: Memory-Aware DAG Scheduling

Minimize Memory for Trees

Minimize Memory for SP-Graphs

Minimize I/Os for Trees

Shared Memory of Parallel Processing