Minimizing I/Os in Out-of-Core Task Tree Scheduling

Loris Marchal[1] Samuel McCauley[2] **Bertrand Simon**[1]
Frédéric Vivien[1]

1: CNRS, INRIA, ENS Lyon and Univ. Lyon, France.
2: IT University Copenhagen, Denmark.

APDCM – Orlando – May 2017

Motivation

Scientific application workflows

- Described as DAGs: nodes = tasks, edges = dependencies
- Can be a tree (multifrontal sparse matrix factorization)

Focus on the memory needs

- Larger memory footprint: may not fit in main memory
- Resort to storing some files on disk: out-of-core execution
- Expensive disk access delays the execution
- Scheduling choices impact memory usage

Objective: Minimize I/Os while scheduling a tree-shaped workflow

Motivation

Scientific application workflows

- Described as DAGs: nodes = tasks, edges = dependencies
- Can be a tree (multifrontal sparse matrix factorization)

Focus on the memory needs

- Larger memory footprint: may not fit in main memory
- Resort to storing some files on disk: out-of-core execution
- Expensive disk access delays the execution
- Scheduling choices impact memory usage

Objective: Minimize I/Os while scheduling a tree-shaped workflow

Ultimate goal: parallel processing

- Problem: sequential case not well understood yet
- Our contribution: step towards its understanding

Problem modeling

Task tree

- Tree $G = (V, E)$, each node has a single parent, $|V| = n$
- Output file of a node i: size w_i (integer number of slots)
- A node must be executed after all its children

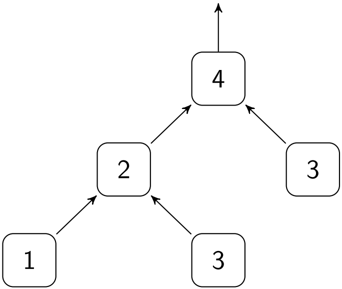
Memory model

- Main memory of size M, infinite disk
- Can move a slot to disk at unit cost: 1 I/O

Memory Management when a node is executed

- Children' output files stored in main memory
- Directly replaced by the node's output file (never coexist)

Example, $M = 5$

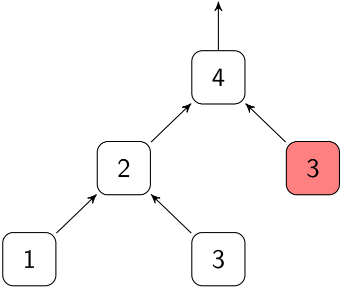


Memory: 0 / 5

Disk: 0

I/Os: 0

Example, $M = 5$

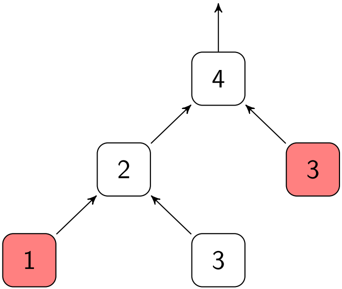


Memory: 3 / 5

Disk: 0

I/Os: 0

Example, $M = 5$

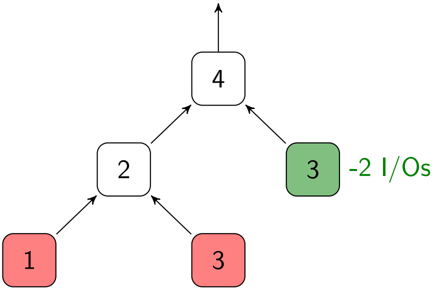


Memory: 4 / 5

Disk: 0

I/Os: 0

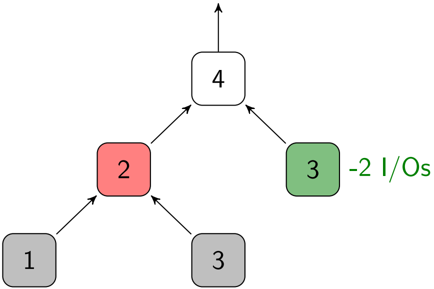
Example, $M = 5$



Memory: 5 / 5

Disk: 2

I/Os: 2

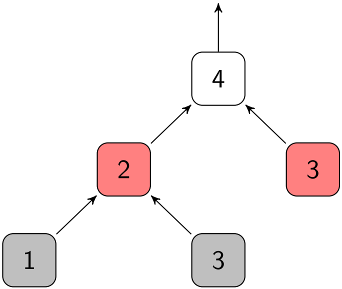


Memory: 3 / 5

Disk: 2

I/Os: 2

Example, $M = 5$

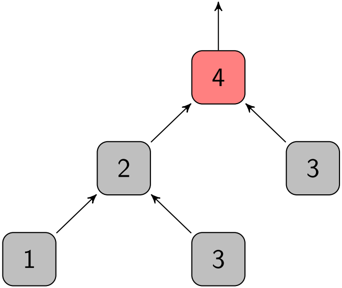


Memory: 5 / 5

Disk: 0

I/Os: 2

Example, $M = 5$



Memory: 4 / 5

Disk: 0

I/Os: 2

Model motivation

Other models used in the literature

- Input and output files coexist
- Additional memory used during execution
- Describe more accurately the reality

Advantages of this model **[Liu 1986, 1987]**

- Simpler theoretic study
- Previous models can be simulated by this one

Description of a solution

Traversal

- Schedule σ: $\sigma(i) = t$ if task i is the t- th executed
- I/O function τ: output file of task i has $\tau(i)$ slots written to disk
- Assume wlog that the data is 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

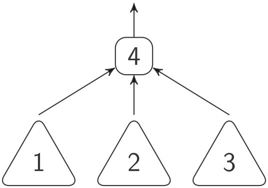
The MinIO problem

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

Lemma: knowing the optimal σ or τ is enough

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)



Related work

Peak memory minimization **[Liu 1986, 1987]**

- Optimal Postorder algorithm: PostOrderMinMem in $\mathcal{O}(n \log n)$
- Optimal algorithm MinMemAlgo in $\mathcal{O}(n^2)$

Minimizing I/Os without splitting files **[Jacquelin et al. 2011]**

- Implies combinatorial choices: NP-complete
- NP-complete even restricted to postorders, or with σ known

Model similar to ours **[Agullo et al. 2010]**

- Optimal Postorder algorithm PostOrderMinIO in $\mathcal{O}(n \log n)$
- Did not consider the general problem

Postorder traversals are optimal on Homogeneous trees

Theorem

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

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

Postorder traversals are optimal on Homogeneous trees

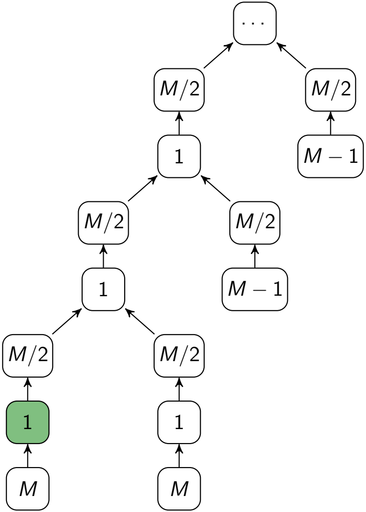
Theorem

Both PostOrderMinMem *and* PostOrderMinIO *minimize I/Os on homogeneous trees (unit file 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. . .

PostOrderMinIO is not competitive



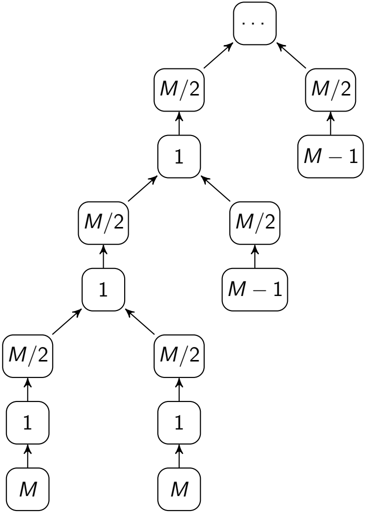
I/O optimal

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

PostOrderMinIO

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

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



I/O optimal

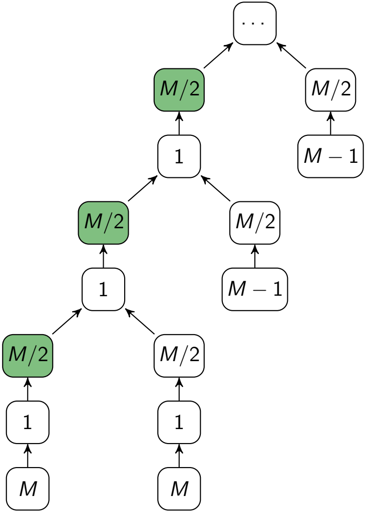
- Peak memory: $M + 1$
- I/Os: 1

POSTORDERMINIO

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

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

PostOrderMinIO is not competitive



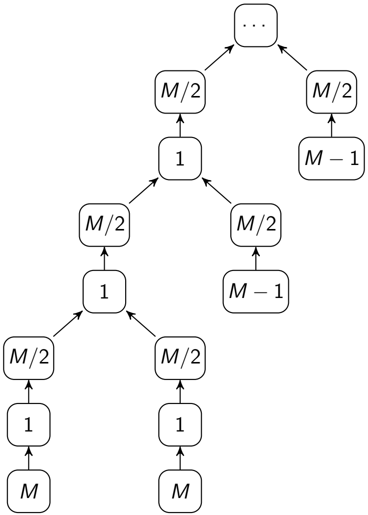
I/O optimal

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

PostOrderMinIO

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

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



I/O optimal

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

PostOrderMinIO

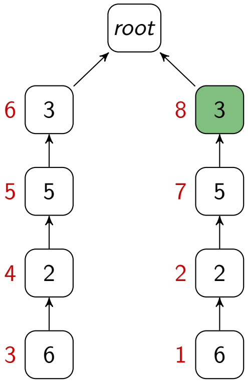
- Peak memory: $\frac{3}{2}M$
- I/Os: $\Theta(|V|M)$

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

Can we rely on MinMemAlgo?

MinMemAlgo is not competitive

$M = 6$



I/O Optimal

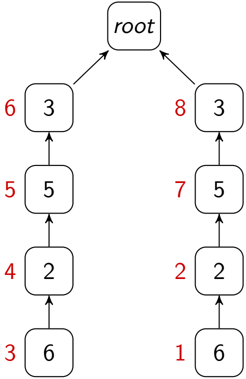
- Peak memory: 9
- I/Os: 3

MinMemAlgo (red labels)

- Peak memory: 8
- I/Os: 4

MinMemAlgo is not competitive

$M = 6$



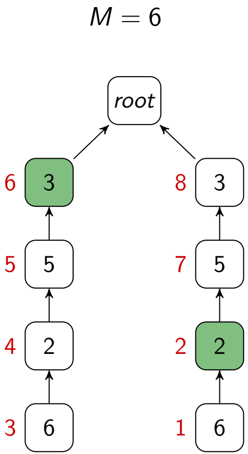
I/O Optimal

- Peak memory: 9
- I/Os: 3

MinMemAlgo (red labels)

- Peak memory: 8
- I/Os: 4

MinMemAlgo is not competitive



I/O Optimal

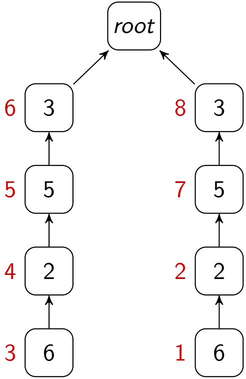
- Peak memory: 9
- I/Os: 3

MinMemAlgo (red labels)

- Peak memory: 8
- I/Os: 4

MinMemAlgo is not competitive

$M = 6$



I/O Optimal

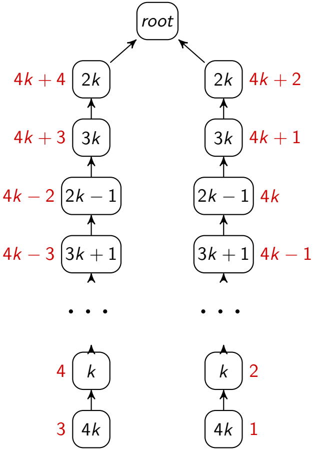
- Peak memory: 9
- I/Os: 3

MinMemAlgo (red labels)

- Peak memory: 8
- I/Os: 4

MinMemAlgo is not competitive

$M = 4k$



I/O Optimal

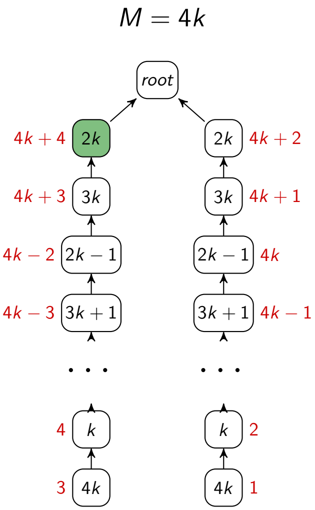
- Peak memory: $6k$
- I/Os: $2k$

MinMemAlgo (red labels)

- Peak memory: $5k$
- I/Os: $> k^2$

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

MinMemAlgo is not competitive



I/O Optimal

- Peak memory: $6k$
- I/Os: $2k$

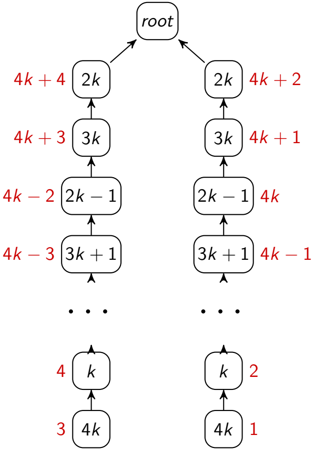
MinMemAlgo (red labels)

- Peak memory: $5k$
- I/Os: $> k^2$

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

MinMemAlgo is not competitive

$M = 4k$



I/O Optimal

- Peak memory: $6k$
- I/Os: $2k$

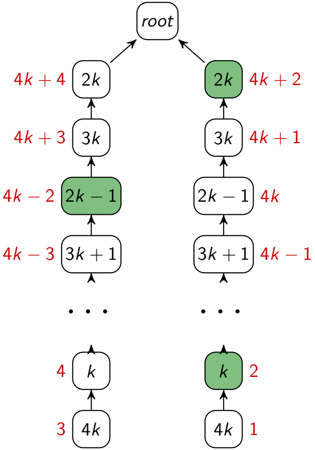
MinMemAlgo (red labels)

- Peak memory: $5k$
- I/Os: $> k^2$

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

MinMemAlgo is not competitive

$M = 4k$



I/O Optimal

- Peak memory: $6k$
- I/Os: $2k$

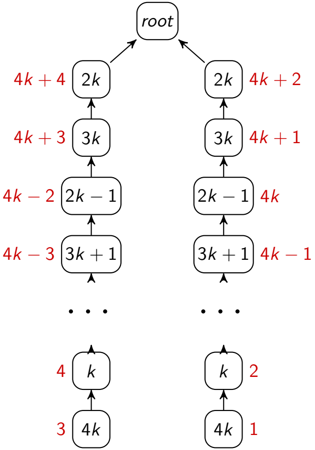
MinMemAlgo (red labels)

- Peak memory: $5k$
- I/Os: $> k^2$

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

MinMemAlgo is not competitive

$M = 4k$



I/O Optimal

- Peak memory: $6k$
- I/Os: $2k$

MinMemAlgo (red labels)

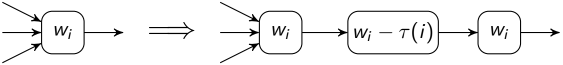
- Peak memory: $5k$
- I/Os: $> k^2$

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

Existing solutions not satisfactory: need for a new heuristic

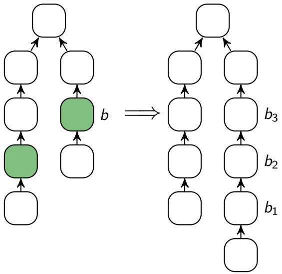
General description

- Underlying concept: run MinMemAlgo several times
- Each run: identify an I/O, then enforce it in the graph



FullRecExpand

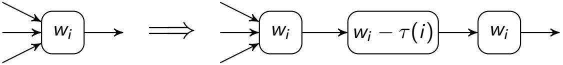
- Recursive calls on the root's children
- While MinMemAlgo needs I/Os:
  - Enforce the I/O that is the latest to be read from disk



New heuristic: FullRecExpand

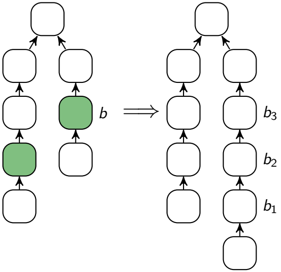
General description

- Underlying concept: run MinMemAlgo several times
- Each run: identify an I/O, then enforce it in the graph



FullRecExpand

- Recursive calls on the root's children
- While MinMemAlgo needs I/Os:
  - Enforce the I/O that is the latest to be read from disk



RecExpand ($\mathcal{O}(n^3)$): ≤ 2 iterations

Experimental setup

Two datasets

- SYNTH: 330 synthetic binary trees of 3000 nodes uniformly drawn, memory weight uniform in $[1; 100]$
- TREES: 330 elimination trees of actual sparse matrices from 2000 to 40000 nodes (University of Florida Sparse Matrix Collection)
- Main memory size (M): mean of
  - Minimum memory for which a solution exists
  - Maximum memory for which I/Os are needed

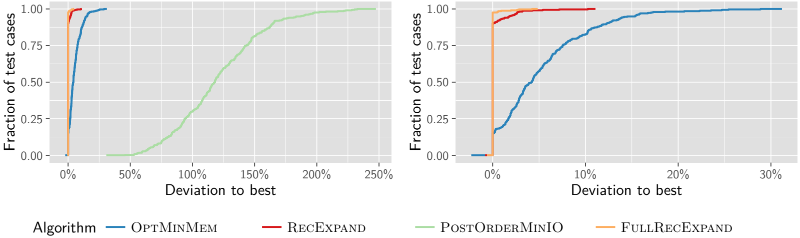
Heuristics

- MinMemAlgo, RecExpand, PostOrderMinIO, FullRecExpand

Performance

- If k I/Os are performed, performance is $1 + \frac{k}{M}$
- Objective: take into account the size of the main memory

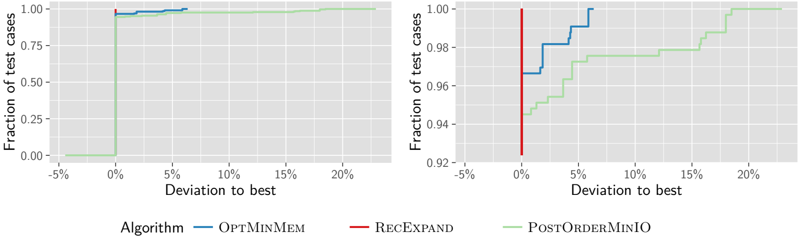
Results on Synth (right graph: zoom)



Analysis ***(Performance profiles: best is top-left)***

- Left: PostOrderMinIO performs poorly ($>$ 100% deviation in 3/4 of the cases)
- Right: RecExpand significantly better than MinMemAlgo:
  - RecExpand best in $\approx$ 90% of the cases
  - MinMemAlgo best in $\approx$ 13% of the cases
- RecExpand is comparable to FullRecExpand

Results on TREES (right graph: zoom)



Analysis ***(best is top-left)***

- Smaller differences (right graph: zoom of the top-left part)
- Most of the graphs have *"easy"* solutions (cannot ensure optimality)
- RECEXPAND is always the best heuristic
- MINMEMALGO outperforms POSTORDERMINIO

Conclusion

The MinIO problem

- Complexity still open, conjectured NP-hard
- Finding σ or τ suffices

Optimal solutions on subclasses

- Optimal postorder algorithm was already known
- PostOrderMinMem optimal for homogeneous trees

Heuristics

- MinMemAlgo performances are not bad
- RecExpand successfully combines the concepts of MinMemAlgo and the memory limit

Perspectives

- Recall: only concerns sequential schedules
- Next step: study I/O efficient parallel schedules (e.g., via memory booking)