Memory-aware schedules for tree-shaped workflows

Mathias Jacquelin, Loris Marchal, Yves Robert and Bora Uçar

CNRS & École Normale Supérieure de Lyon, France

INRIA ROMA project-team
LIP (ENS-Lyon, CNRS, INRIA)
École Normale Supérieure de Lyon, France

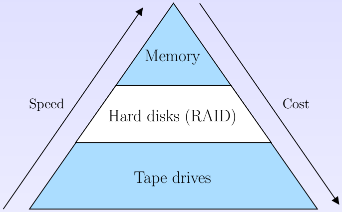
Workshop in Aussois,
Aussois, June 3, 2010.

Outline

Motivation: BlueWaters

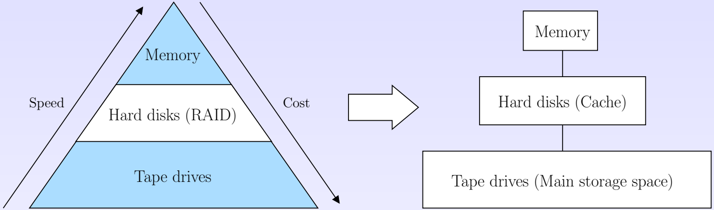
- Sustained Petaflops/s for general applications.
- Hierarchical storage system
  - Hard disk drives used as "cache"
  - Tape drives used as actual permanent storage media
- Objective: Design an efficient disk management policy

Motivation: BlueWaters



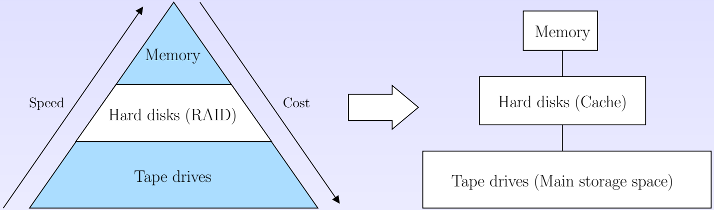
- Sustained Petaflops/s for general applications.
- Hierarchical storage system
  - Hard disk drives used as ''cache''
  - Tape drives used as actual permanent storage media
- Objective: Design an efficient disk management policy

Motivation: BlueWaters



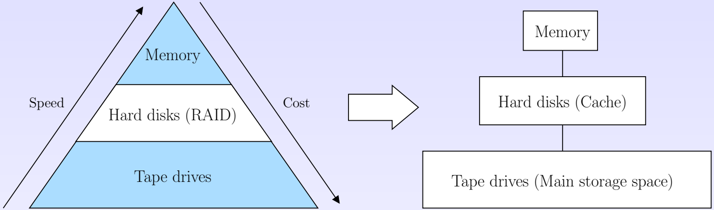
- Sustained Petaflops/s for general applications.
- Hierarchical storage system
  - Hard disk drives used as ''cache''
  - Tape drives used as actual permanent storage media
- Objective: Design an efficient disk management policy

Motivation: BlueWaters



- Sustained Petaflops/s for general applications.
- Hierarchical storage system
  - Hard disk drives used as ''cache''
  - Tape drives used as actual permanent storage media
- Objective: Design an efficient disk management policy

Motivation: BlueWaters



- Sustained Petaflops/s for general applications.
- Hierarchical storage system
  - Hard disk drives used as ''cache''
  - Tape drives used as actual permanent storage media
- Objective: Design an efficient disk management policy

MinIO and MinMemory problems

MinIO

Given the size M of the main memory, determine the minimum I/O volume that is required to execute the application.

MinIO and MinMemory problems

MinIO

Given the size M of the main memory, determine the minimum I/O volume that is required to execute the application.

Today :

MinMemory

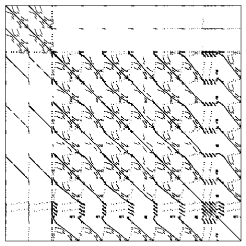
Determine the minimum amount of main memory that is required to execute the application without any access to secondary memory

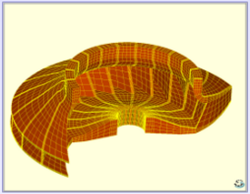
Application model

- Application modeled by DAGs
  - File sizes on edges
  - Computation memory overhead on nodes
- Homogeneous MinMemory (a.k.a. pebble game) on DAGs is NP-complete (Sethi'73).

Refine analysis for tree-shaped workflows

Motivation: Sparse Matrices and linear algebra

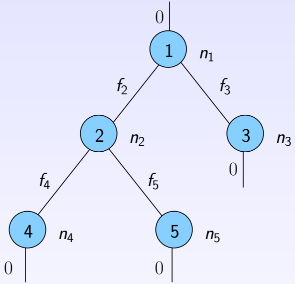




- Large peak memory requirements
- Memory usage becomes a bottleneck

- Objective: Minimize the amount of required memory, and minimize the IO-volume for out-of-core computations

Introduction: tree-shaped workflows



- p nodes.
- Input file of size f_i.
- Execution file of size n_i.
- Root input file of null size.
- Leaf nodes produce files of null size.
- Memory required for node i:

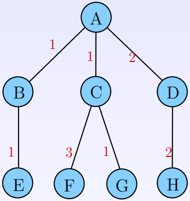
$$MemReq(i) = f_i + n_i + \sum_{j \in Children(i)} f_j$$

Introduction: model emulation

- Data is overwritten

Overwritten data model

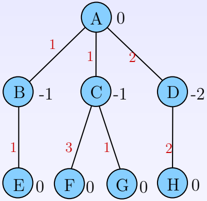
$\max(f_i, \sum_{j \in Children(i)} f_j)$



$\Rightarrow$

Our model

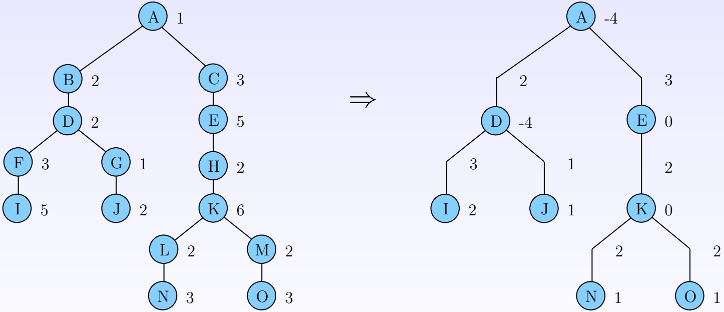
$f_i + n_i + \sum_{j \in Children(i)} f_j$



Introduction: model emulation

- Data is overwritten
- Liu's model (no weight on edges)

Liu's model	Our model
$\max(n_i, \sum_{j \in Children(i)} n_j)$	$f_i + n_i + \sum_{j \in Children(i)} f_j$



Outline

The MinMemory problem

MinMemory

Given a tree $\mathcal{T}$ with p nodes, determine the minimum amount of memory M such that there exists a schedule $\sigma(\mathcal{T}, p, M)$.

MinMemory has polynomial-time complexity (Liu'87).

Post-order traversal in the general case

Best bottom-up post-order traversal (Liu'86)

Best post-order traversal is obtained by sorting, at each level, subtrees in non increasing order of $\max_{i \in \text{subtree}}(MemReq(i)) - f_{\text{subroot}}$.

Post-order traversal in the general case

Best bottom-up post-order traversal (Liu'86)

Best post-order traversal is obtained by sorting, at each level, subtrees in non increasing order of $\max_{i \in \text{subtree}} (MemReq(i)) - f_{\text{subroot}}$.

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.

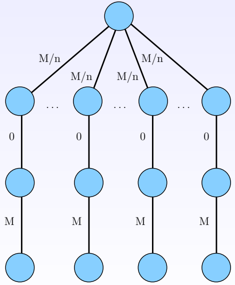
Post-order traversal in the general case

Best bottom-up post-order traversal (Liu'86)

Best post-order traversal is obtained by sorting, at each level, subtrees in non increasing order of $\max_{i \in \text{subtree}} (MemReq(i)) - f_{\text{subroot}}$.

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 memory

$$M_{\text{min}} = M$$

- Any post-order traversal

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

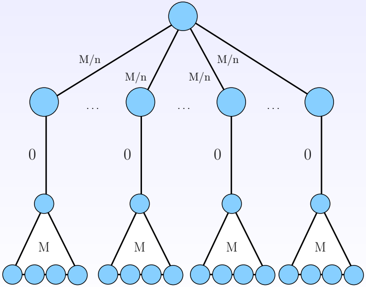
Post-order traversal in the general case

Best bottom-up post-order traversal (Liu'86)

Best post-order traversal is obtained by sorting, at each level, subtrees in non increasing order of $\max_{i \in \text{subtree}} (MemReq(i)) - f_{\text{subroot}}$.

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 memory

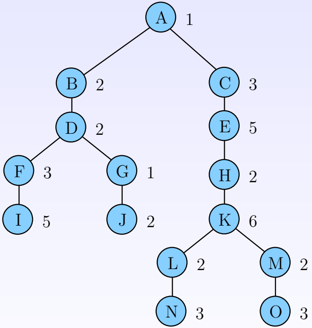
$$M_{\text{min}} = M$$

- Any post-order traversal

$$M_{\text{min}} = M + 2(n-1)M/n$$

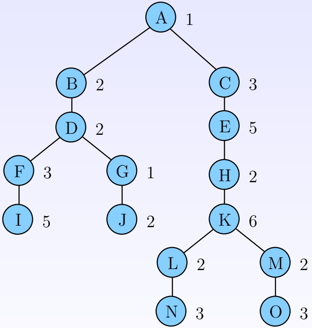
Liu's optimal algorithm (Liu'87)

- Extends the original pebble game on trees (Tarjan'80).
- Rationale: Recursive bottom-up algorithm combining schedules of subtrees.
- Combination based on the notion of Hill-Valley Segments.



Liu's optimal algorithm (Liu'87)

- Extends the original pebble game on trees (Tarjan'80).
- Rationale: Recursive bottom-up algorithm combining schedules of subtrees.
- Combination based on the notion of Hill-Valley Segments.



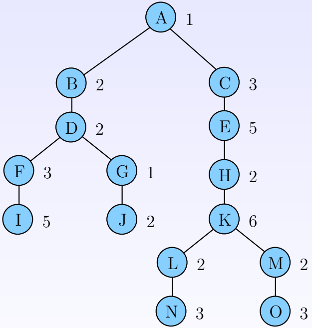
T[A]
- Schedule : {N, L, O, M, K, H, I, F, J, G, D, B E, C, A}
- Cost sequence : {E(7), A(1)}
- Valley segments : {(NA = 6)}

T[B]
- Schedule : {I, F, J, G, D, B}
- Cost sequence : {J(5), B(2)}
- Valley segments : {(IB = 3)}

T[C]
- Schedule : {N, L, O, M, K, H, E, C}
- Cost sequence : {K(6), H(2), E(5), C(3)}
- Valley segments : {(NH = 4) , (HC = 2)}

Liu's optimal algorithm (Liu'87)

- Extends the original pebble game on trees (Tarjan'80).
- Rationale: Recursive bottom-up algorithm combining schedules of subtrees.
- Combination based on the notion of Hill-Valley Segments.



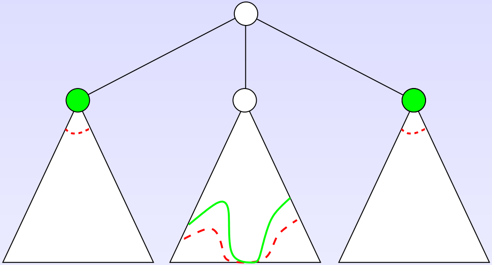
T[A]
- Schedule : {N, L, O, M, K, H, I, F, J, G, D, B E, C, A}
- Cost sequence : {E(7), A(1)}
- Valley segments : {(NA = 6)}

T[B]
- Schedule : {I, F, J, G, D, B}
- Cost sequence : {J(5), B(2)}
- Valley segments : {(IB = 3)}

T[C]
- Schedule : {N, L, O, M, K, H, E, C}
- Cost sequence : {K(6), H(2), E(5), C(3)}
- Valley segments : {(NH = 4) , (HC = 2)}

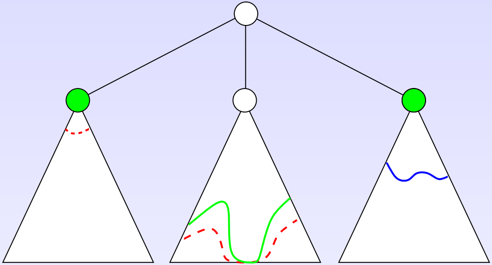
- Complexity : $O(N^2)$

MinMem optimal algorithm



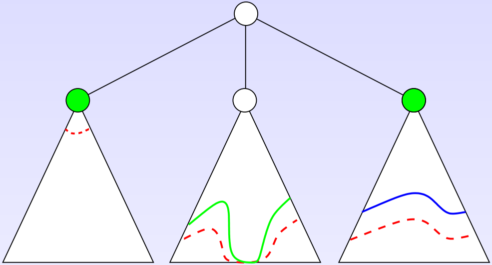
- Based on the *Explore* subroutine
  - Rationale: recursively computes the minimum cut of the tree reachable with a memory of size M in a top-down approach.
  - Returns:
    - Cut of the subtree
    - Schedule of the subtree to the cut
    - Next smallest memory peak (impassable hill)
- Gradually increases the memory available for exploration using previous cut as a shortcut

MinMem optimal algorithm



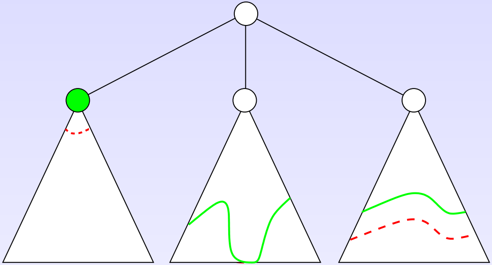
- Based on the *Explore* subroutine
  - Rationale: recursively computes the minimum cut of the tree reachable with a memory of size M in a top-down approach.
  - Returns:
    - Cut of the subtree
    - Schedule of the subtree to the cut
    - Next smallest memory peak (impassable hill)
- Gradually increases the memory available for exploration using previous cut as a shortcut

MinMem optimal algorithm



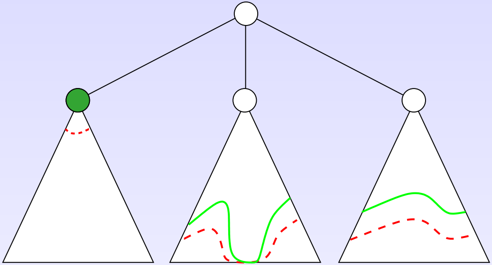
- Based on the *Explore* subroutine
  - Rationale: recursively computes the minimum cut of the tree reachable with a memory of size M in a top-down approach.
  - Returns:
    - Cut of the subtree
    - Schedule of the subtree to the cut
    - Next smallest memory peak (impassable hill)
- Gradually increases the memory available for exploration using previous cut as a shortcut

MinMem optimal algorithm



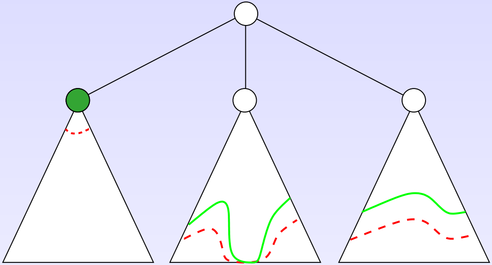
- Based on the *Explore* subroutine
  - Rationale: recursively computes the minimum cut of the tree reachable with a memory of size M in a top-down approach.
  - Returns:
    - Cut of the subtree
    - Schedule of the subtree to the cut
    - Next smallest memory peak (impassable hill)
- Gradually increases the memory available for exploration using previous cut as a shortcut

MinMem optimal algorithm



- Based on the *Explore* subroutine
  - Rationale: recursively computes the minimum cut of the tree reachable with a memory of size M in a top-down approach.
  - Returns:
    - Cut of the subtree
    - Schedule of the subtree to the cut
    - Next smallest memory peak (impassable hill)
- Gradually increases the memory available for exploration using previous cut as a shortcut

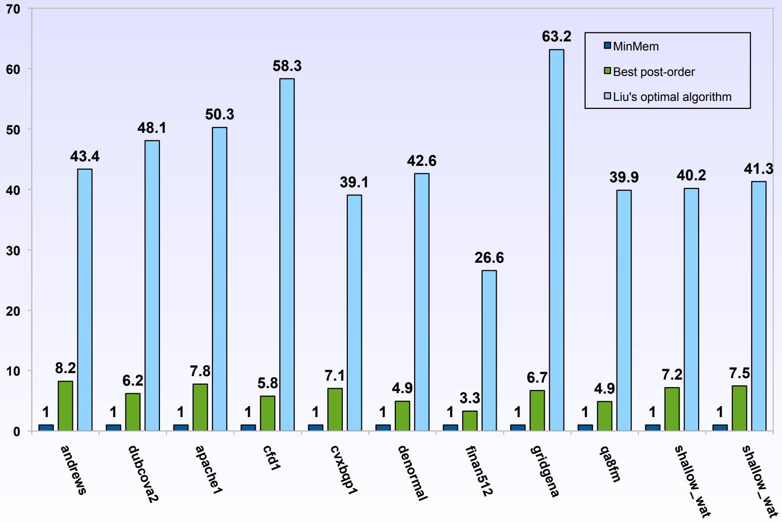
MinMem optimal algorithm



- Based on the *Explore* subroutine
  - Rationale: recursively computes the minimum cut of the tree reachable with a memory of size M in a top-down approach.
  - Returns:
    - Cut of the subtree
    - Schedule of the subtree to the cut
    - Next smallest memory peak (impassable hill)
- Gradually increases the memory available for exploration using previous cut as a shortcut
- Complexity : $O(N^2)$

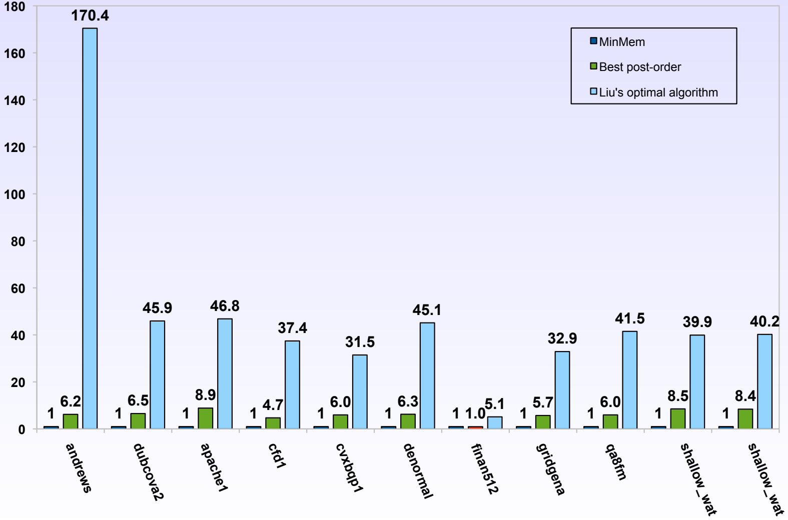
Performance of *MinMem*

Sparse Cholesky assembly trees
Runtimes / *MinMem* using AMD ordering



Performance of *MinMem*

Sparse Cholesky assembly trees
Runtimes / *MinMem* using METIS ordering



Performance of *MinMem*

Memory required to process randomly weighted assembly trees

- Keep assembly tree structure.
- Randomly set the weights.

Matrix	Optimal	Post-Order	Improvement
tandem_dual METIS agg =1	36229.20	46811.40	0.29
onera_dual METIS agg =1	32537.20	41873.00	0.29
poisson3Db METIS agg =1	12809.60	17562.60	0.37
poisson3Db METIS agg =4	6812.40	8505.40	0.25
poisson3Db METIS agg =16	4389.40	5157.40	0.17
poisson3Db AMD agg =1	14346.80	18703.60	0.30
poisson3Db AMD agg =4	7448.20	8471.00	0.14
poisson3Db AMD agg =16	3504.60	3993.40	0.14

Outline

Conclusion and perspectives

Contributions

- Post-order traversal can be arbitrarily bad.
- In preliminary experiments, Post-order almost always optimal.
- New optimal top-down algorithm for MINMEMORY.
- Experimental performance evaluation of Liu, Post-order and *MinMem* using real-life sparse Cholesky factorization assembly trees.

Ongoing work

- Benchmark all algorithms on an extensive collection of sparse assembly trees.
- Investigate MINIO problem.

Future work

- Extend to DAGs using heuristics.
- Write efficient data replacement policies based on MINIO, in the context of BlueWaters.