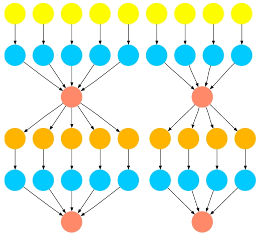
Efficient scheduling of DAGs under bounded memory

Loris Marchal
Joint work with Bertrand Simon & Frédéric Vivien

Scheduling for Large Scale Systems Workshop 2019 Bordeaux

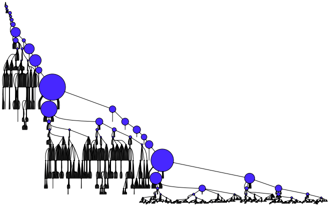
Modeling scientific applications as task graphs

- Scientific applications divided into rather independent modules (tasks)
- Tasks linked through data dependencies
- Directed Acyclic Graph of tasks



Multifrontal sparse matrix factorization over runtimes

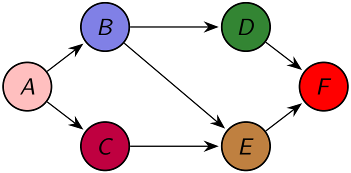
SOLHAR



- Task graph: tree (with dependencies towards the root)
- Large temporary data
- Memory becomes a bottleneck
- Schedule trees with limited memory

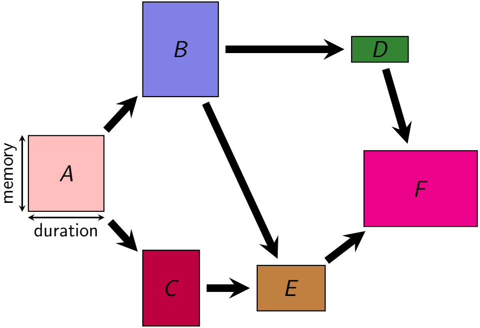
Task graph scheduling and memory

- Consider a simple task graph



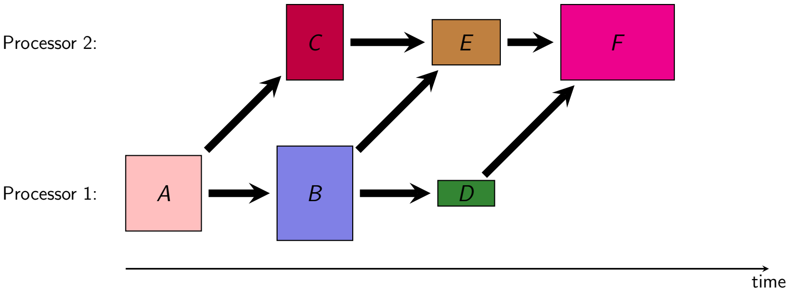
Task graph scheduling and memory

- Consider a simple task graph
- Tasks have durations and memory demands



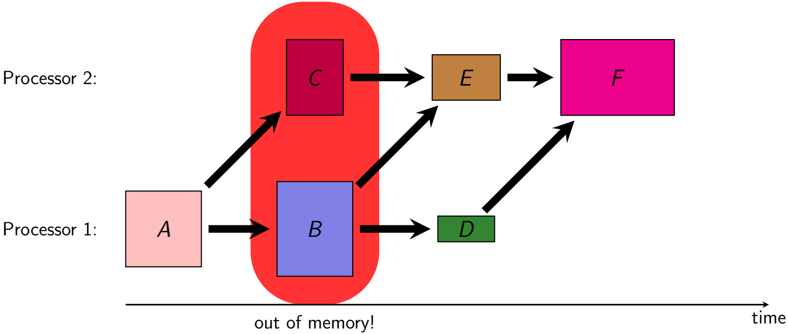
Task graph scheduling and memory

- Consider a simple task graph
- Tasks have durations and memory demands



Task graph scheduling and memory

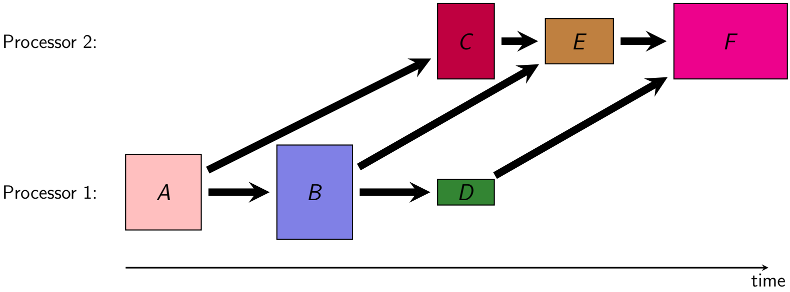
- Consider a simple task graph
- Tasks have durations and memory demands



- Peak memory: maximum memory usage

Task graph scheduling and memory

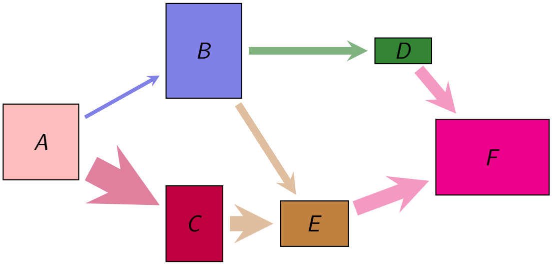
- Consider a simple task graph
- Tasks have durations and memory demands



- Peak memory: maximum memory usage
- Trade-off between peak memory and makespan

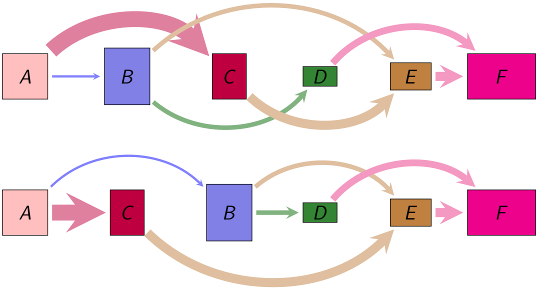
Going back to sequential processing

- Temporary data require memory
- Scheduling influences the peak memory



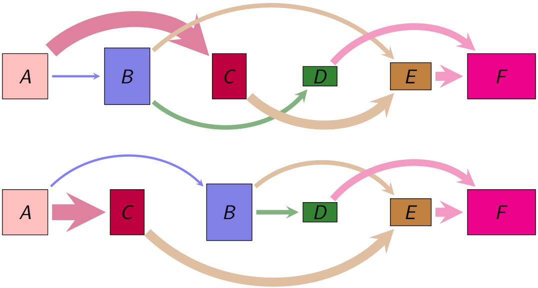
Going back to sequential processing

- Temporary data require memory
- Scheduling influences the peak memory



Going back to sequential processing

- Temporary data require memory
- Scheduling influences the peak memory

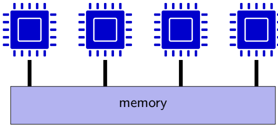


Known results on sequential memory-aware scheduling:

- Optimal algorithm for trees [Liu, 1897], SP-graphs [Kayaaslan et al., 2018]
- General graphs with unit size weights: pebble game [Sethi and Ullman, 1973], PSPACE complete [Gilbert et al., 1980]

Today's focus

- Schedule general graphs
- On a shared-memory platform



First option: design good static scheduler:

- NP-complete, non-approximable
- Cannot react to unpredicted changes in the platform or inaccuracies in task timings

Second option:

- Limit memory consumption of any dynamic scheduler
  Target: runtime systems
- Without impacting too much parallelism

Outline

Outline

Memory model

Task graphs with:

- Vertex weights (w_i): task (estimated) durations
- Edge weights ($m_{i,j}$): data sizes

Memory model

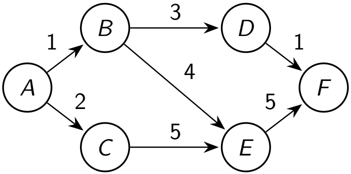
Task graphs with:

- Vertex weights (w_i): task (estimated) durations
- Edge weights ($m_{i,j}$): data sizes

Simple memory model: at the beginning of a task

- Inputs are freed (instantaneously)
- Outputs are allocated

At the end of a task: outputs stay in memory



Memory model

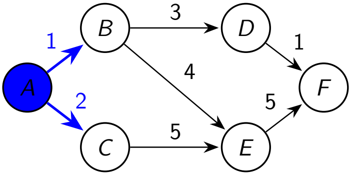
Task graphs with:

- Vertex weights (w_i): task (estimated) durations
- Edge weights ($m_{i,j}$): data sizes

Simple memory model: at the beginning of a task

- Inputs are freed (instantaneously)
- Outputs are allocated

At the end of a task: outputs stay in memory



Memory model

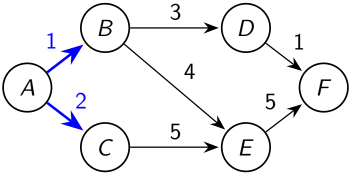
Task graphs with:

- Vertex weights (w_i): task (estimated) durations
- Edge weights ($m_{i,j}$): data sizes

Simple memory model: at the beginning of a task

- Inputs are freed (instantaneously)
- Outputs are allocated

At the end of a task: outputs stay in memory



Memory model

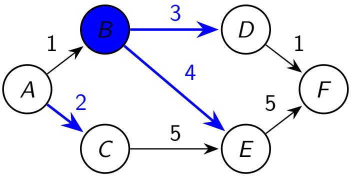
Task graphs with:

- Vertex weights (w_i): task (estimated) durations
- Edge weights ($m_{i,j}$): data sizes

Simple memory model: at the beginning of a task

- Inputs are freed (instantaneously)
- Outputs are allocated

At the end of a task: outputs stay in memory



Memory model

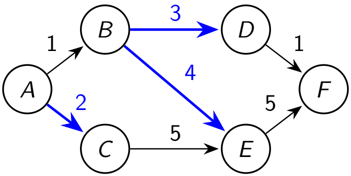
Task graphs with:

- Vertex weights (w_i): task (estimated) durations
- Edge weights ($m_{i,j}$): data sizes

Simple memory model: at the beginning of a task

- Inputs are freed (instantaneously)
- Outputs are allocated

At the end of a task: outputs stay in memory



Memory model

Task graphs with:

- Vertex weights (w_i): task (estimated) durations
- Edge weights ($m_{i,j}$): data sizes

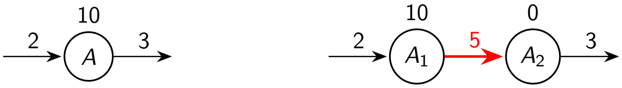
Simple memory model: at the beginning of a task

- Inputs are freed (instantaneously)
- Outputs are allocated

At the end of a task: outputs stay in memory

Emulation of other memory behaviours:

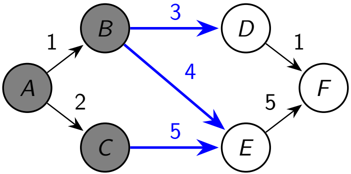
- Inputs + outputs allocated during task: duplicate nodes



Computing the maximum memory peak

Topological cut: (S, T) with:

- S include the source node, T include the target node
- No edge from T to S
- Weight of the cut = weight of all edges from S to T



Any topological cut corresponds to a possible state when all node in S are completed or being processed.

Two equivalent questions (in our model):

- What is the maximum memory of any parallel execution?
- What is the topological cut with maximum weight?

Computing the maximum topological cut

Literature:

- Lots of studies of various cuts in non-directed graphs ([Diaz,2000] on Graph Layout Problems)
- Minimum cut is polynomial on both directed/non-directed graphs
- Maximum cut NP-complete on both directed/non-directed graphs ([Karp 1972] for non-directed, [Lampis 2011] for directed ones)
- Not much for topological cuts

Theorem.

Computing the maximum topological cut of a DAG can be done in polynomial time.

Maximum topological cut – using LP

- Consider one classical LP formulation for finding a minimum cut:

$$\min \sum_{(i,j)\in E} m_{i,j} d_{i,j}$$

$$\forall (i,j) \in E, \quad d_{i,j} \geq p_i - p_j$$

$$\forall (i,j) \in E, \quad d_{i,j} \geq 0$$

$$p_s = 1, \;\; p_t = 0$$

Maximum topological cut – using LP

- Consider one classical LP formulation for finding a minimum cut:

$$\min \sum_{(i,j)\in E} m_{i,j} d_{i,j}$$
$$\forall (i,j) \in E, \quad d_{i,j} \geq p_i - p_j$$
$$\forall (i,j) \in E, \quad d_{i,j} \geq 0$$
$$p_s = 1, \; p_t = 0$$

- Integer solution $\Leftrightarrow$ topological cut

Maximum topological cut – using LP

- Consider one classical LP formulation for finding a minimum cut:

$$\max \sum_{(i,j)\in E} m_{i,j} d_{i,j}$$
$$\forall (i,j) \in E, \quad d_{i,j} = p_i - p_j$$
$$\forall (i,j) \in E, \quad d_{i,j} \geq 0$$
$$p_s = 1, \quad p_t = 0$$

- Integer solution $\Leftrightarrow$ topological cut
- Then change the optimization direction (min $\rightarrow$ max)

Maximum topological cut – using LP

- Consider one classical LP formulation for finding a minimum cut:

$$\max \sum_{(i,j)\in E} m_{i,j} d_{i,j}$$
$$\forall (i,j) \in E, \quad d_{i,j} = p_i - p_j$$
$$\forall (i,j) \in E, \quad d_{i,j} \geq 0$$
$$p_s = 1, \;\; p_t = 0$$

- Integer solution $\Leftrightarrow$ topological cut
- Then change the optimization direction (min $\rightarrow$ max)
- Draw w uniformly in $]0, 1[$, define the cut such that $S_w = \{i \mid p_i > w\}, \quad T_w = \{i \mid p_i \leq w\}$
- Expected cost of this cut $= M^*$ (opt. rational solution)
- All cuts with random w have the same cost M^*

Maximum topological cut – direct algorithm

- Dual problem: Min-Flow *(larger than all edge weights)*
- Idea: use an optimal algorithm for Max-Flow

Algorithm sketch

1. Build a large flow F on the graph G
2. Consider G^{diff} with edge weights $F_{i,j} - m_{i,j}$
3. Compute a maximum flow *maxdiff* in G^{diff}
4. $F - maxdiff$ is a minimum flow in G
5. Residual graph $\rightarrow$ maximum topological cut

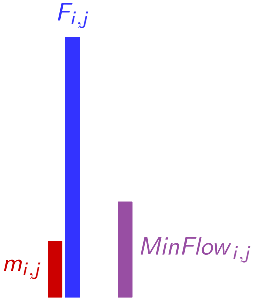
Complexity: same as maximum flow, e.g., $O(|V|^2|E|)$

Maximum topological cut – direct algorithm

- Dual problem: Min-Flow *(larger than all edge weights)*
- Idea: use an optimal algorithm for Max-Flow

Algorithm sketch

1. Build a large flow F on the graph G
2. Consider G^{diff} with edge weights $F_{i,j} - m_{i,j}$
3. Compute a maximum flow $maxdiff$ in G^{diff}
4. $F - maxdiff$ is a minimum flow in G
5. Residual graph → maximum topological cut



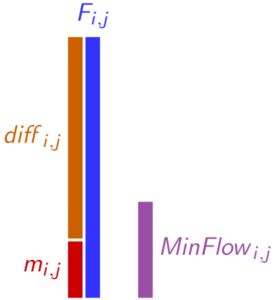
Complexity: same as maximum flow, e.g., $O(|V|^2|E|)$

Maximum topological cut – direct algorithm

- Dual problem: Min-Flow *(larger than all edge weights)*
- Idea: use an optimal algorithm for Max-Flow

Algorithm sketch

1. Build a large flow F on the graph G
2. Consider G^{diff} with edge weights $F_{i,j} - m_{i,j}$
3. Compute a maximum flow *maxdiff* in G^{diff}
4. $F - maxdiff$ is a minimum flow in G
5. Residual graph → maximum topological cut



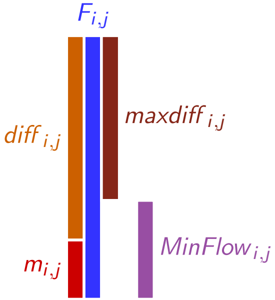
Complexity: same as maximum flow, e.g., $O(|V|^2|E|)$

Maximum topological cut – direct algorithm

- Dual problem: Min-Flow *(larger than all edge weights)*
- Idea: use an optimal algorithm for Max-Flow

Algorithm sketch

1. Build a large flow F on the graph G
2. Consider G^{diff} with edge weights $F_{i,j} - m_{i,j}$
3. Compute a maximum flow $maxdiff$ in G^{diff}
4. $F - maxdiff$ is a minimum flow in G
5. Residual graph → maximum topological cut



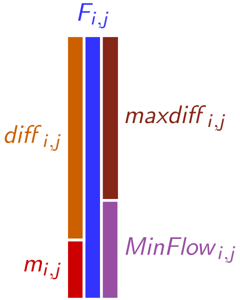
Complexity: same as maximum flow, e.g., $O(|V|^2|E|)$

Maximum topological cut – direct algorithm

- Dual problem: Min-Flow *(larger than all edge weights)*
- Idea: use an optimal algorithm for Max-Flow

Algorithm sketch

1. Build a large flow F on the graph G
2. Consider G^{diff} with edge weights $F_{i,j} - m_{i,j}$
3. Compute a maximum flow *maxdiff* in G^{diff}
4. $F - maxdiff$ is a minimum flow in G
5. Residual graph $\rightarrow$ maximum topological cut



Complexity: same as maximum flow, e.g., $O(|V|^2|E|)$

Summary

Predict the maximal memory of any dynamic scheduling

$$\Leftrightarrow$$

Compute the maximal topological cut

Two algorithms:

- Linear program + rounding
- Direct algorithm based on MaxFlow/MinCut

Outline

Coping with limiting memory

Problem:

- Limited available memory M
- Allow use of dynamic schedulers
- Avoid running out of memory
- Keep high level of parallelism (as much as possible)

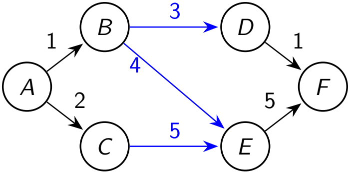
Coping with limiting memory

Problem:

- Limited available memory M
- Allow use of dynamic schedulers
- Avoid running out of memory
- Keep high level of parallelism (as much as possible)

Our solution:

- Add edges to guarantee that any parallel execution stays below M
  fictitious dependencies to reduce maximum memory
- Minimize the obtained critical path



$M = 10$

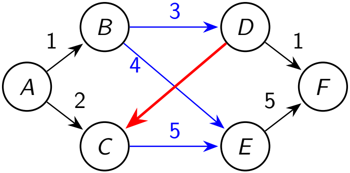
Coping with limiting memory

Problem:

- Limited available memory M
- Allow use of dynamic schedulers
- Avoid running out of memory
- Keep high level of parallelism (as much as possible)

Our solution:

- Add edges to guarantee that any parallel execution stays below M
  fictitious dependencies to reduce maximum memory
- Minimize the obtained critical path



$M = 10$

Problem definition and complexity

Definition (PartialSerialization).

Given a DAG $G = (V, E)$ and a bound M, find a set of new edges E' such that $G' = (V, E \cup E')$ is a DAG, $MaxMem(G') \leq M$ and $CritPath(G')$ is minimized.

Theorem.

PartialSerialization is NP-hard in the stronge sense.

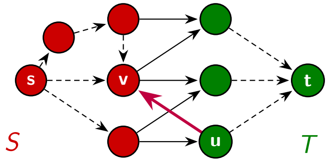
NB: stays NP-hard if we are given a sequential schedule σ of G which uses at most a memory M.

Heuristic solutions for PartialSerialization

Framework:

(inspired by [Sbîrlea et al. 2014])

1. Compute a max. top. cut (S, T)
2. If weight $\leq M$: succeeds
3. Add edge (u, v) with $u \in T,\ v \in S$ without creating cycles; or fail
4. Goto Step 1



Several heuristic choices for Step 3:

MinLevels does not create a large critical path

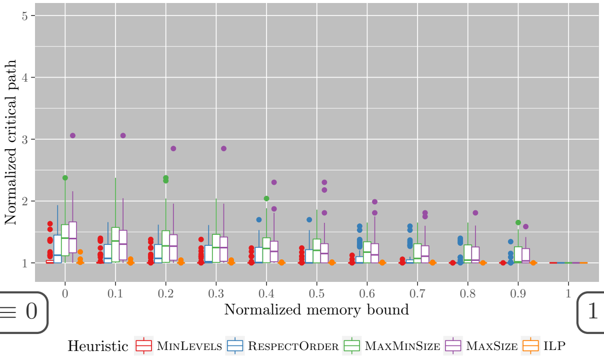
RespectOrder follows a precomputed memory-efficient schedule, always succeeds

MaxSize targets nodes dealing with large data

MaxMinSize variant of MaxSize

Simulations: dense random graphs (25, 50, 100 nodes)

lower is better



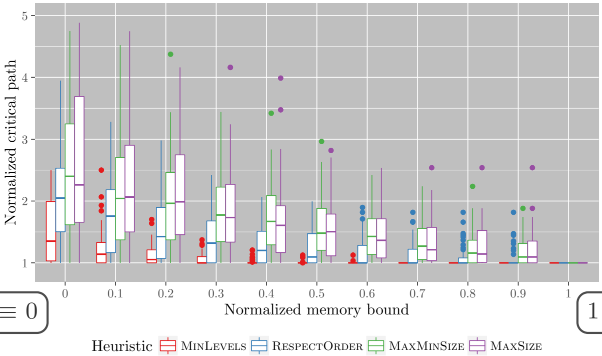
DFS memory $\equiv 0$

$1 \equiv$ MaxTopCut

- *x: memory (0 = DFS, 1 = MaxTopCut)*
  median ratio MaxTopCut / DFS ≈ 1.3
- *y: CP / original CP → lower is better*
- MinLevels performs best

Simulations: sparse random graphs (25, 50, 100 nodes)

lower is better



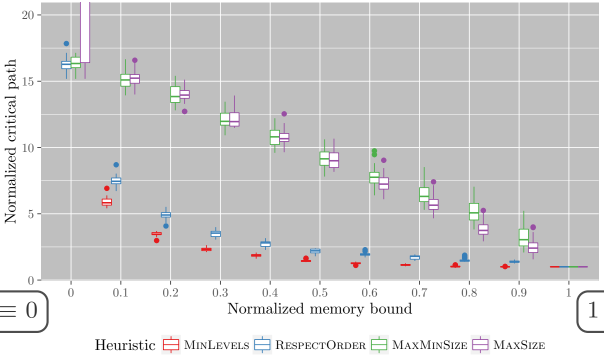
DFS memory $\equiv 0$

$1 \equiv$ MaxTopCut

- *x: memory (0 = DFS, 1 = MaxTopCut)*
  median ratio MaxTopCut / DFS ≈ 2
- *y: CP / original CP $\rightarrow$ lower is better*
- MinLevels performs best, but might fail

Simulations – Pegasus workflows (LIGO 100 nodes)

lower is better



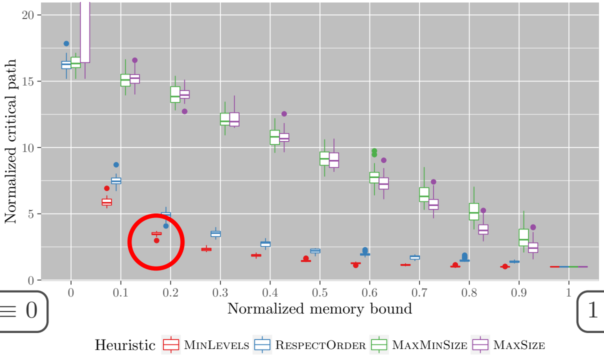
DFS memory $\equiv 0$

$1 \equiv$ MaxTopCut

- *Median ratio MaxTopCut / DFS $\approx$ 20*
- MinLevels performs best, RespectOrder always succeeds

Simulations – Pegasus workflows (LIGO 100 nodes)

lower is better



DFS memory $\equiv 0$

$1 \equiv$ MaxTopCut

- *Median ratio MaxTopCut / DFS $\approx$ 20*
- MinLevels performs best, RespectOrder always succeeds
- Memory divided by 5 for CP multiplied by 3

Outline

Conclusion

- Dynamic scheduling with bounded memory
  - Explicit algo. to compute maximum memory
- Adding fictitious dependencies to limit memory usage
  - Changing the graph to allow various scheduling strategies
  - Critical path as a performance metric
  - Several heuristics (+ ILP)

Perspectives:

- Reduce heuristic complexity to cope with large graphs
- Include knowledge on the dynamic scheduler
- Other performance metric?
- Approximations algorithms??