Parallel Scheduling of DAGs Under Memory Constraints

Loris Marchal, Bertrand Simon & Frédéric Vivien

Universität Bremen, Germany & ENS de Lyon, France

MAPSP, Renesse — 2019
Breaking down the title

**DAGs of tasks**

- Describe many applications
- Used by increasingly popular runtime schedulers
  
  \(XKAAP\), \(StarPU\), \(StarSs\), \(ParSEC\), . . .

**Parallel scheduling**

- Many tasks executed concurrently

**Limited available memory (shared-memory platform)**

- Simple breadth-first traversal may go out-of-memory

**Objective**

- Prevent dynamic schedulers from exceeding memory
Outline

1. Model and maximum parallel memory
   - Memory model
   - Maximum parallel memory/maximal topological cut

2. Efficient scheduling with bounded memory
   - Problem definition
   - Complexity
   - Heuristics

3. Simulation results

4. Conclusion
Memory model

**Task graph weights**

- Vertex $w_i$: estimated task duration
- Edge $m_{i,j}$: data size
Memory model

Task graph weights
- Vertex $w_i$: estimated task duration
- Edge $m_{i,j}$: data size

Memory behavior
- Task starts: free inputs (instantaneously) allocate outputs
- Task ends: outputs stay in memory

$M_{used} = 0$
Memory model

Task graph weights
- Vertex $w_i$: estimated task duration
- Edge $m_{i,j}$: data size

Memory behavior
- Task starts: free inputs (instantaneously)
  allocate outputs
- Task ends: outputs stay in memory

$M_{used} = 3$
**Memory model**

**Task graph weights**
- Vertex $w_i$: estimated task duration
- Edge $m_{i,j}$: data size

**Memory behavior**
- Task starts: free inputs (instantaneously) allocate outputs
- Task ends: outputs stay in memory

![Task graph diagram](image)

$M_{used} = 3$
Memory model

**Task graph weights**
- Vertex $w_i$: estimated task duration
- Edge $m_{i,j}$: data size

**Memory behavior**
- Task starts: free inputs (instantaneously) allocate outputs
- Task ends: outputs stay in memory

![Task graph diagram]

$M_{used} = 9$
Memory model

**Task graph weights**
- Vertex $w_i$: estimated task duration
- Edge $m_{i,j}$: data size

**Memory behavior**
- Task starts: free inputs (instantaneously) allocate outputs
- Task ends: outputs stay in memory

![Task graph diagram]

$M_{used} = 9$
Memory model

**Task graph weights**
- Vertex $w_i$: estimated task duration
- Edge $m_{i,j}$: data size

**Memory behavior**
- Task starts: free inputs (instantaneously) allocate outputs
- Task ends: outputs stay in memory

**Emulation of other memory behaviours**
- Inputs not freed, additional execution memory: duplicate nodes

![Diagram](attachment:memory_model_diagram.png)
Maximum memory peak equivalent

**Topological cut = partition of the vertices** \((S, T)\) with

- Source \(s \in S\) and sink \(t \in T\)
- No edge from \(T\) to \(S\)
- Weight of the cut = sum of all edge weights from \(S\) to \(T\)
Maximum memory peak equivalent

Topological cut = partition of the vertices \((S, T)\) with

- Source \(s \in S\) and sink \(t \in T\)
- No edge from \(T\) to \(S\)
- Weight of the cut = sum of all edge weights from \(S\) to \(T\)

\[ M_{used} = 12 \]

Topological cut ⇔ execution state where \(T\) nodes are not started yet
Maximum memory peak equivalent

**Topological cut** = partition of the vertices \((S, T)\) with

- Source \(s \in S\) and sink \(t \in T\)
- No edge from \(T\) to \(S\)
- Weight of the cut = sum of all edge weights from \(S\) to \(T\)

**Diagram:**

- \(M_{used} = 12\)

```
S ---- 2 ---- C ---- 5 ---- E ---- 5 ---- T
     |         |         |         |
     1        3        1
     |         |         |
     A ---- 1 ---- B ---- 4 ---- D
```

**Topological cut** ←→ execution state where \(T\) nodes are not started yet

**Equivalence in our model between:**

- Maximum memory peak (any parallel execution)
- Maximum weight of a topological cut
Computing the maximum topological cut

**Literature**

- Minimum cut is polynomial on graphs
- Maximum cut is NP-hard even on DAGs
- Not much for *topological* cuts

**Theorem**

*Computing the maximum topological cut on a DAG is polynomial.*
Maximum topological cut – using LP

A classical min-cut LP formulation

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

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

\[ d_{i,j} \geq 0 \]

\[ p_s = 1, \quad p_t = 0 \]

Any graph: integer solution \(\Longleftrightarrow\) cut
Maximum topological cut – using LP

A classical min-cut LP formulation

\[
\max \sum_{(i,j) \in E} m_{i,j}d_{i,j}
\]

\[
\forall (i,j) \in E, \quad d_{i,j} = p_i - p_j
\]

\[
d_{i,j} \geq 0
\]

\[
p_s = 1, \quad p_t = 0
\]

Any graph: integer solution ⇔ cut

Modify LP: ‘min’ → ‘max’ ; ‘≥’ → ‘=’
Maximum topological cut – using LP

A classical min-cut LP formulation

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

- Any graph: integer solution \( \iff \) cut
- Modify LP: ‘\( \min \)’ \( \rightarrow \) ‘\( \max \)’ ; ‘\( \geq \)’ \( \rightarrow \) ‘\( = \)’

In a DAG, any (non-integer) optimal solution \( \implies \) max. top. cut

- Any rounding of the \( p_i \)’s works (large \( \in \) \( S \), small \( \in \) \( T \))
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 $G^{\text{diff}}$ with edge weights $F_{i,j} - m_{i,j}$
3. Compute a maximum flow $\text{maxdiff}$ in $G^{\text{diff}}$
4. $F - \text{maxdiff}$ 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 $G^{\text{diff}}$ with edge weights $F_{i,j} - m_{i,j}$
3. Compute a maximum flow $\text{maxdiff}$ in $G^{\text{diff}}$
4. $F - \text{maxdiff}$ 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 $\rightarrow$ 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 $G_{\text{diff}}$ with edge weights $F_{i,j} - m_{i,j}$
3. Compute a maximum flow $\text{maxdiff}$ in $G_{\text{diff}}$
4. $F - \text{maxdiff}$ 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 $G^{\text{diff}}$ with edge weights $F_{i,j} - m_{i,j}$
3. Compute a maximum flow $\text{maxdiff}$ in $G^{\text{diff}}$
4. $F - \text{maxdiff}$ is a minimum flow in $G$
5. Residual graph $\rightarrow$ maximum topological cut

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

1 Model and maximum parallel memory
   - Memory model
   - Maximum parallel memory/maximal topological cut

2 Efficient scheduling with bounded memory
   - Problem definition
   - Complexity
   - Heuristics

3 Simulation results

4 Conclusion
Coping with limited memory

Problem

- Allow use of dynamic schedulers
- Limited available memory $M$
- Keep high level of parallelism
Coping with limited memory

**Problem**
- Allow use of dynamic schedulers
- Limited available memory $M$
- Keep high level of parallelism

**Our solution**
- Add *edges* to guarantee that any parallel execution stays below $M$
- Minimize the obtained *critical path*

![Diagram with nodes and edges]

$M_{\text{available}} = 10$
Coping with limited memory

Problem

- Allow use of dynamic schedulers
- Limited available memory $M$
- Keep high level of parallelism

Our solution

- Add edges to guarantee that any parallel execution stays below $M$
- Minimize the obtained critical path

![Graph diagram with nodes A, B, C, D, E, F and arrows indicating dependencies and weights]

$M_{\text{available}} = 10$
### Problem definition and complexity

**Definition (PARTIALSERIALIZATION of a DAG $G$ under a memory $M$)**

Compute a set of new edges $E'$ such that:
- $G' = (V, E \cup E')$ is a DAG
- $\text{MaxTopologicalCut}(G') \leq M$
- $\text{CritPath}(G')$ is minimized

**Theorem (Sethi 1975)**

*Computing a schedule that minimizes the memory usage is NP-hard.*

$\Rightarrow$ finding a DAG $G'$ with $\text{MaxTopologicalCut}(G') \leq M$ is NP-hard

**Theorem**

*PARTIALSERIALIZATION is NP-hard given a memory-efficient schedule.*

Optimal solution computable by an ILP (builds transitive closure)
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
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
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
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
Heuristic solutions for **PARTIAL SERIALIZATION**

### 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
Outline

1. Model and maximum parallel memory
   - Memory model
   - Maximum parallel memory/maximal topological cut

2. Efficient scheduling with bounded memory
   - Problem definition
   - Complexity
   - Heuristics

3. Simulation results

4. Conclusion
Dense DAGGEN random graphs (25, 50, and 100 nodes)

> x: memory \((0 = DFS, 1 = MaxTopCut)\)
> 
> median ratio \(MaxTopCut / DFS \approx 1.3\)

> y: \(CP / original\ CP \rightarrow lower\ is\ better\)

> MinLevels performs best
Sparse DAGGEN random graphs (25, 50, and 100 nodes)

- **Heuristic**: MinLevels, RespectOrder, MaxMinSize, MaxSize

- **DFS memory**: $0$
- **MaxTopCut**: $1$

- **$x$: memory** ($0 = \text{DFS}, 1 = \text{MaxTopCut}$)
  - median ratio $\text{MaxTopCut} / \text{DFS} \approx 2$

- **$y$: critical path** / original CP → lower is better

- **MinLevels** performs best, but might fail
Simulations – Pegasus workflows (LIGO 100 nodes)

- DFS memory $\equiv 0$
- $1 \equiv \text{MaxTopCut}$

- Median ratio $\text{MaxTopCut} / \text{DFS} \approx 20$
- $\text{MinLevels}$ performs best, $\text{RespectOrder}$ always succeeds
- Memory divided by 5 for CP multiplied by 3
Simulations – Pegasus workflows (LIGO 100 nodes)

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

DFS memory $\equiv 0$

$1 \equiv \text{MaxTopCut}$
Outline

1 Model and maximum parallel memory
   - Memory model
   - Maximum parallel memory/maximal topological cut

2 Efficient scheduling with bounded memory
   - Problem definition
   - Complexity
   - Heuristics

3 Simulation results

4 Conclusion
Conclusion

Memory model proposed

- Simple but expressive
- Explicit algorithm to compute maximum memory

Prevent dynamic schedulers from exceeding memory

- Adding fictitious dependences to limit memory usage
- Critical path as a performance metric
- Several heuristics (+ ILP)

Perspectives

- Reduce heuristic complexity
- Adapt performance metric to a platform
- Consider more clever schedulers
- Distributed memory