Parallel Scheduling of DAGs
Under Memory Constraints

Loris Marchal, Hanna Nagy, Bertrand Simon & Frédéric Vivien

ENS de Lyon, France

IPDPS — Vancouver 2018
Breaking down the title

**DAGs of tasks**
- Describe many applications
- Used by increasingly popular runtime schedulers
  - (XKAAPL, 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

**Simple memory model**

- 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

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

![Diagram of a task graph with vertices A, B, C, D, E, and F, and edges marked with numbers 1, 2, 3, 4, 5, 6, 7, and 8. The number of used memory is 3.](image)
Memory model

Task graph weights

- Vertex $w_i$: estimated task duration
- Edge $m_{ij}$: data size

Simple memory model

- 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

Simple memory model

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

$M_{used} = 9$
Memory model

Task graph weights

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

Simple memory model

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

$M_{used} = 9$
Memory model

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

Simple memory model
- 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 showing task graph weights](attachment:task_graph_weights.png)
Memory model

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

Simple memory model
- 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
- Shared data: output data of A used for both B and C
Computing the maximum memory peak

Two equivalent quantities (in our model)
- Maximum memory peak of any parallel execution
- Maximum weight of a topological cut

Topological cut: \((S, T)\) 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 \(T\)
Computing the maximum memory peak

Two equivalent quantities (in our model)

- Maximum memory peak of any parallel execution
- Maximum weight of a topological cut

Topological cut: \((S, T)\) 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](image-url)

\(M_{used} = 12\)

Topological cut \(\leftrightarrow\) execution state where \(T\) nodes are not started yet
Computing the maximum topological cut

**Literature**

- Minimum cut is polynomial on graphs
- Maximum cut is NP-hard even on DAGs [Lampis et al. 2011]
- 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

\[
\begin{align*}
\min & \sum_{(i,j) \in E} m_{i,j} d_{i,j} \\
\forall (i,j) \in E, \quad & d_{i,j} \geq p_i - p_j \\
& d_{i,j} \geq 0 \\
p_s = 1, \quad & p_t = 0
\end{align*}
\]

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

A classical min-cut LP formulation

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

- 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 \textit{(larger than all edge weights)}
- Idea: use an optimal algorithm for Max-Flow

**Algorithm sketch**

1. Build a \textit{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 $G_{\text{diff}}$ 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., $O(|V|^2|E|)$
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 of a Directed Acyclic Graph (DAG)]

$M_{\text{available}} = 10$
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]

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

<table>
<thead>
<tr>
<th>Definition (PartialSerialization of a DAG G under a memory M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute a set of new edges $E'$ such that:</td>
</tr>
<tr>
<td>- $G' = (V, E \cup E')$ is a DAG</td>
</tr>
<tr>
<td>- $\text{MaxTopologicalCut}(G') \leq M$</td>
</tr>
<tr>
<td>- $\text{CritPath}(G')$ is minimized</td>
</tr>
</tbody>
</table>

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
4. Goto Step 1

### Several heuristic choices for Step 3

- **MinLevels** Minimize \(\text{TopLevel}(u) + \text{BottomLevel}(v)\)
- **RespectOrder**
  - Pre-compute a *good* sequential schedule \(\sigma\)
  - Step 3: select first vertex \(u \in T\), last vertex \(v \in S\) in \(\sigma\)
    - *Always succeeds if memory*(\(\sigma\)) \(\leq M\)
- **MaxSize** Maximize \(\text{Inputs}(u) + \text{Outputs}(v)\)
- **MaxMinSize** Maximize \(\min \{\text{Inputs}(u), \text{Outputs}(v)\}\)
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
4. Goto Step 1

Several heuristic choices for Step 3

- **MinLevels** Minimize \(TopLevel(u) + BottomLevel(v)\)
- **RespectOrder**
  - Pre-compute a good sequential schedule \(\sigma\)
  - Step 3: select first vertex \(u \in T\), last vertex \(v \in S\) in \(\sigma\)
  - Always succeeds if memory(\(\sigma\)) \(\leq M\)
- **MaxSize** Maximize \(Inputs(u) + Outputs(v)\)
- **MaxMinMaxSize** Maximize \(\min\{Inputs(u), Outputs(v)\}\)
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
4. Goto Step 1

Several heuristic choices for Step 3

- **MinLevels** Minimize \(\text{TopLevel}(u) + \text{BottomLevel}(v)\)
- **RespectOrder**
  - Pre-compute a good sequential schedule \(\sigma\)
  - Step 3: select first vertex \(u \in T\), last vertex \(v \in S\) in \(\sigma\)
  - Always succeeds if memory(\(\sigma\)) \(\leq M\)
- **MaxSize** Maximize \(\text{Inputs}(u) + \text{Outputs}(v)\)
- **MaxMinSize** Maximize \(\min\{\text{Inputs}(u), \text{Outputs}(v)\}\)
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
4. Goto Step 1

**Several heuristic choices for Step 3**

- **MinLevels** Minimize \(TopLevel(u) + BottomLevel(v)\)
- **RespectOrder**
  - Pre-compute a *good* sequential schedule \(\sigma\)
  - Step 3: select first vertex \(u \in T\), last vertex \(v \in S\) in \(\sigma\)
  - *Always succeeds if memory* \((\sigma) \leq M\)
- **MaxSize** Maximize \(Inputs(u) + Outputs(v)\)
- **MaxMinSize** Maximize \(\min\{Inputs(u), Outputs(v)\}\)
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
4. Goto Step 1

**Several heuristic choices for Step 3**

- **MinLevels** Minimize \(TopLevel(u) + BottomLevel(v)\)
- **RespectOrder**
  - Pre-compute a good sequential schedule \(\sigma\)
  - Step 3: select first vertex \(u \in T\), last vertex \(v \in S\) in \(\sigma\)

  *Always succeeds if memory(\(\sigma\)) \(\leq M\)*

- **MaxSize** Maximize \(Inputs(u) + Outputs(v)\)
- **MaxMinSize** Maximize \(\min\{Inputs(u), Outputs(v)\}\)
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
4. Goto Step 1

Several heuristic choices for Step 3

- **MinLevels** Minimize \(TopLevel(u) + BottomLevel(v)\)
- **RespectOrder**
  - Pre-compute a good sequential schedule \(\sigma\)
  - Step 3: select first vertex \(u \in T\), last vertex \(v \in S\) in \(\sigma\)
  - Always succeeds if memory \((\sigma) \leq M\)
- **MaxSize** Maximize \(Inputs(u) + Outputs(v)\)
- **MaxMinSize** Maximize \(\min \{Inputs(u), Outputs(v)\}\)
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 → lower is better

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

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

- **y**: $CP / original CP \rightarrow lower is better$

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

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