Task Graph Scheduling on Modern Computing Platforms

Bertrand SIMON

ENS de Lyon, France

Bremen — May 2018
3rd-year PhD student in ENS Lyon, ROMA team

- Advisors: Loris Marchal & Frédéric Vivien
- Defending on July 4th

Education and previous experience

- Licence & Master in CS at ENS Lyon
- 2015: 5-month research visit at the Stony Brook University, NY in the team of Michael Bender
Outline

1. PhD thesis overview

2. Online scheduling of DAGs on hybrid platforms

3. Parallel scheduling of DAGs under memory constraints
Focus on three main challenges

- Exploiting task parallelism
- Using efficiently heterogeneous processors
- Coping with a limited memory
Exploiting task parallelism

Main difficulty

- Cope with two conflicting types of parallelism

Context

- Literature: few speedup assumptions $\rightarrow$ complex algorithms
- Linear algebra: similar tasks so similar speedup
design of low-complexity algorithms

<table>
<thead>
<tr>
<th>Guermouche, Marchal, Simon, Vivien</th>
<th>EuroPar 2016</th>
</tr>
</thead>
<tbody>
<tr>
<td>Existing speedup function [Prasanna &amp; Musicus 1996]</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Marchal, Simon, Sinnen, Vivien</th>
<th>TPDS 2018</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design a tunable two-threshold roofline speedup function</td>
<td></td>
</tr>
<tr>
<td>High accuracy on extensive benchmarks for linear algebra kernels</td>
<td></td>
</tr>
</tbody>
</table>
Using efficiently heterogeneous processors

**Setting**
- Two types of processors (CPUs and GPUs)
- Online: remainder of graph unknown

**Main difficulty**
- Decide which tasks should be accelerated on rare GPUs

---

**Canon, Marchal, Simon, Vivien EuroPar 2018**
- Online DAG scheduling: lower bounds and competitive algorithms

— First focus of this talk
Coping with a limited available memory

First setting: some executions fit in memory

Marchal, Nagy, Simon, Vivien  
Prevent dynamic schedulers from exceeding memory  
IPDPS 2018

— Second focus of this talk

Second setting: insufficient memory, I/Os necessary

- I/O minimization: NP-hard on DAGs
  NP-hard on trees with unsplittable files

Marchal, McCauley, Simon, Vivien  
Minimize I/Os in task trees with splittable files; complexity open  
IPDPS Workshops 2017
**Additional projects: external memory data structures**

**Complexity = I/O number**

**Main difficulty**
- Group elements to optimize I/Os

**Bender, Chowdury, Conway, Farach-Colton, Ganapathi, Johnson, McCauley, Simon, Singh**  
*LATIN 2016*

- Minimize the I/O complexity of computing prime tables via sieves

**Bender, Berry, Johnson, Kroeger, McCauley, Phillips, Simon, Singh, Zage**  
*PODS 2016*

Design two history-independent randomized data structures
- Extend skip-lists (binary search tree variant) to external memory
- History-independent PMA (sorted elements maintained in an array)
Outline

1. PhD thesis overview

2. Online scheduling of DAGs on hybrid platforms
   - Lower bounds
   - Competitive algorithms
   - Simulations results

3. Parallel scheduling of DAGs under memory constraints
   - Model and maximum parallel memory
   - Efficient scheduling with bounded memory
   - Simulation results
Online scheduling of DAGs on hybrid platforms

Hybrid Platforms
- Many CPUs + few accelerators (GPUs, Xeon Phi, ...)

Task Graphs (DAGs)
- Used in runtime schedulers (StarPU, StarSs, XKaapi, ParSEC ...)

Online Scheduling
- Unknown graph
  - tasks not submitted yet
  - depends on results
- Advantages vs offline
  - quicker decisions
  - robust to inaccuracies
- **Semi-online:** partial information, e.g., bottom-levels (≈ critical path)

Main challenge
- Take binary decisions without knowing the future
Model and toy example

Model

- $m$ CPUs $\geq k$ GPUs
- Graph of tasks $T_i: \{\overline{p_i} = \text{CPU time}; \ p_i = \text{GPU time}\}$
- Online: only available tasks are known

Objective: minimize makespan

Example (2 CPUs, 1 GPU)
Model and toy example

**Model**

- $m$ CPUs $\geq k$ GPUs
- Graph of tasks $T_i : \{ \overline{p_i} = \text{CPU time} ; \underline{p_i} = \text{GPU time} \}$
- Online: only available tasks are known

**Objective: minimize makespan**

Example (2 CPUs, 1 GPU)
Model and toy example

Model

- $m$ CPUs $\geq k$ GPUs
- Graph of tasks $T_i : \{\bar{p}_i = \text{CPU time}; \underline{p}_i = \text{GPU time}\}$
- Online: only available tasks are known

Objective: minimize makespan

Example (2 CPUs, 1 GPU)
Model and toy example

Model

- $m$ CPUs $\geq k$ GPUs
- Graph of tasks $T_i: \{\overline{p_i} = \text{CPU time}; \underline{p_i} = \text{GPU time}\}$
- Online: only available tasks are known

Objective: minimize makespan

Example (2 CPUs, 1 GPU)
Model and toy example

Model

- $m$ CPUs $\geq k$ GPUs
- Graph of tasks $T_i: \{ \overline{p_i} = \text{CPU time}; \underline{p_i} = \text{GPU time} \}$
- Online: only available tasks are known

Objective: minimize makespan

Example (2 CPUs, 1 GPU)
Model and toy example

Model

- $m$ CPUs $\geq k$ GPUs
- Graph of tasks $T_i : \{ \overline{p_i} = \text{CPU time} ; p_i = \text{GPU time} \}$
- Online: only available tasks are known

Objective: minimize makespan

Example (2 CPUs, 1 GPU)
Model and toy example

Model

- \( m \) CPUs ≥ \( k \) GPUs
- Graph of tasks \( T_i : \{ \overline{p_i} = \text{CPU time} ; \underline{p_i} = \text{GPU time} \} \)
- Online: only available tasks are known

Objective: minimize makespan

Example (2 CPUs, 1 GPU)
Related work

**Existing offline algorithms (NP-Complete)**

- Independent tasks:
  - $\frac{4}{3} + \frac{1}{3k}$ - approx
    
    [Bleuse, Kedad-Sidhoum, Monna, Mounié, Trystram 2015]

    Expensive PTAS

  - Low-complexity: 2 - approx
    
    [Bonifaci, Wiese 2012]

    3.41 - approx

    [Canon, Marchal, Vivien 2017]

- DAG: 6 - approx (LP rounding)

  [Kedad-Sidhoum, Monna, Trystram 2015]

**Existing online algorithms**

- Independent tasks: 4 - competitive

  [Imreh 2003]

  3.85 - competitive

  [Chen, Ye, Zhang 2014]

- DAG: $4\sqrt{\frac{m}{k}}$ - compet. ER-LS

  [Amarís, Lucarelli, Mommessin, Trystram 2017]
Outline

1. PhD thesis overview

2. Online scheduling of DAGs on hybrid platforms
   - Lower bounds
   - Competitive algorithms
   - Simulations results

3. Parallel scheduling of DAGs under memory constraints
   - Model and maximum parallel memory
   - Efficient scheduling with bounded memory
   - Simulation results
Lower bound

**Theorem**

No online algorithm $\mathcal{A}$ is $< \sqrt{m/k}$-competitive for any $m$, $k$.

**Proof** (where $\tau = \sqrt{m/k} = 3$): graph built in $n\tau$ phases.

Phase 1 - $k\tau$ independent tasks $\{p_i = \tau ; \ p_i = 1\}$: $\mathcal{A}$ needs a time $\tau$
Lower bound

Theorem

No online algorithm $A$ is $< \sqrt{m/k}$-competitive for any $m, k$.

Proof (where $\tau = \sqrt{m/k} = 3$): graph built in $n\tau$ phases.

Phase 1 - $k\tau$ independent tasks $\{p_i = \tau ; p_i = 1\}$: $A$ needs a time $\tau$

Phase 2 - same as phase 1, but are successors of the last task

Graph with $k = 2, n = 3$
Lower bound

**Theorem**

*No online algorithm $\mathcal{A}$ is $< \sqrt{m/k}$-competitive for any $m, k$.*

**Proof (where $\tau = \sqrt{m/k} = 3$):** graph built in $n\tau$ phases.

- **Phase 1** - $k\tau$ independent tasks $\{\overline{p}_i = \tau; \; p_i = 1\}$: $\mathcal{A}$ needs a time $\tau$
- **Phase 2** - same as phase 1, but are successors of the last task
- **Phase 3** - same as phase 2, but are successors of the last task

![Graph with $k = 2, n = 3$](image)
Lower bound

**Theorem**

No online algorithm $\mathcal{A}$ is $< \sqrt{m/k}$ - competitive for any $m, k$.

**Proof (where $\tau = \sqrt{m/k} = 3$):** graph built in $n\tau$ phases.

- **Phase 1** - $k\tau$ independent tasks $\{ \bar{p}_j = \tau ; p_i = 1 \}$: $\mathcal{A}$ needs a time $\tau$
- **Phase 2** - same as phase 1, but are successors of the last task
- **Phase 3** - same as phase 2, but are successors of the last task
- **Phase $x$** - ...

$\Rightarrow$ Makespan obtained by $\mathcal{A}$: $n\tau^2$

Graph with $k = 2, n = 3$
Lower bound

Theorem

No online algorithm $\mathcal{A}$ is $< \sqrt{m/k}$-competitive for any $m$, $k$.

Proof (where $\tau = \sqrt{m/k} = 3$): graph built in $n\tau$ phases.

Graph with $k = 2$, $n = 3$
Lower bound

**Theorem**

*No online algorithm $\mathcal{A}$ is $< \sqrt{m/k}$-competitive for any $m, k$.*

**Proof (where $\tau = \sqrt{m/k} = 3$):** graph built in $n\tau$ phases.

Graph with $k = 2, \ n = 3$
Lower bound

**Theorem**

No online algorithm $\mathcal{A}$ is $< \sqrt{m/k}$-competitive for any $m, k$.

**Proof (where $\tau = \sqrt{m/k} = 3$):** graph built in $n\tau$ phases.

Graph with $k = 2$, $n = 3$
No online algorithm $A$ is $< \sqrt{m/k}$-competitive for any $m, k$.

Proof (where $\tau = \sqrt{m/k} = 3$): graph built in $n\tau$ phases.

Graph with $k = 2, n = 3$
Lower bound

Theorem

No online algorithm $\mathcal{A}$ is $< \sqrt{m/k}$-competitive for any $m, k$.

Proof (where $\tau = \sqrt{m/k} = 3$): graph built in $n\tau$ phases.

$n\tau$ phases

OPT $\Rightarrow$ $(n+1)\tau$

$\mathcal{A} \Rightarrow n\tau^2$

Lower bound: $\frac{n}{n+1}\tau$

Graph with $k = 2$, $n = 3$
Generalized lower bounds

Recall previous lower bound: $\sqrt{m/k}$, for $m$ CPUs, $k$ GPUs

**Precomputed information**

- Bottom-level ($\approx$ remaining critical path) does not help
- All descendants: non-constant LB $= \Omega \left( (m/k)^{1/4} \right)$

**Powerful scheduler**

- $Kill + migrate$ does not help
- $Preempt + migrate$ hardly helps

**Note: allocation is difficult**

- How to choose which tasks to speed-up?
- Fixed allocation: 3-competitiveness
Outline

1. PhD thesis overview

2. Online scheduling of DAGs on hybrid platforms
   - Lower bounds
   - Competitive algorithms
   - Simulations results

3. Parallel scheduling of DAGs under memory constraints
   - Model and maximum parallel memory
   - Efficient scheduling with bounded memory
   - Simulation results
ER-LS algorithm \((4\sqrt{m/k}\text{-competitive}, [\text{Amarís et al.}])\)

**Main concept**

- Pick any available task \(T_i\)
- Allocate \(T_i\) to CPUs or GPUs
- Schedule it as soon as possible

**Where to allocate an available task \(T_i\)**

**If** \(T_i\) can be completed on GPU before time \(\bar{p}_i\):
- put \(T_i\) on GPU

**Otherwise:**
- if \(\frac{\bar{p}_i}{p_i} \leq \sqrt{\frac{m}{k}}\) : put it on CPU
- else : put it on GPU
Our proposition: QA (Quick Allocation) algorithm

**Main concept**
- Pick any available task $T_i$
- Allocate $T_i$ to CPUs or GPUs
- Schedule it as soon as possible

<table>
<thead>
<tr>
<th>Where to allocate an available task $T_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>If $T_i$ can be completed on GPU before time $\bar{p}_i$:</td>
</tr>
<tr>
<td>- put $T_i$ on GPU</td>
</tr>
<tr>
<td>Otherwise:</td>
</tr>
<tr>
<td>- if $\frac{\bar{p}_i}{p_i} \leq \sqrt{\frac{m}{k}}$: put it on CPU</td>
</tr>
<tr>
<td>- else: put it on GPU</td>
</tr>
</tbody>
</table>

$m$ CPUs, $k$ GPUs
Our proposition: QA (Quick Allocation) algorithm

Main concept

- Pick any available task $T_i$
- Allocate $T_i$ to CPUs or GPUs
- Schedule it as soon as possible

Where to allocate an available task $T_i$

If $T_i$ can be completed on GPU before time $\bar{p}_i$:
- put $T_i$ on GPU

Otherwise:
- if $\frac{\bar{p}_i}{p_i} \leq \sqrt{\frac{m}{k}}$ : put it on CPU
- else : put it on GPU

Theorem

QA is $2\sqrt{m/k} + 1$ - competitive. This ratio is (almost) tight.
What about easy cases?

Problem with QA

- Expect the worse: aim at $\Theta(\sqrt{m/k})$-competitiveness
- 😞 Poor performance on easy graphs

Well-known EFT algorithm (Earliest Finish Time)

- Terminate each $T_i$ as soon as possible;
- ☑ Greedy version, works great on non-pathological cases
- 😞 Can be really bad: $\geq \left(\frac{m}{k} + 2\right) \text{OPT}$

Can we have both benefits? MixEFT

- Run EFT and simulate QA;
  When EFT is $\lambda$ times worse than QA: switch to QA;
- Tunable: $\lambda = 0 \rightarrow$ QA ; $\lambda = \infty \rightarrow$ EFT
- $(\lambda + 1)(2\sqrt{m/k} + 1)$-competitive — conjectured $\max(\lambda, 2\sqrt{m/k} + 1)$
- Same idea as ER-LS but pushed to the extreme
Outline

1 PhD thesis overview

2 Online scheduling of DAGs on hybrid platforms
   - Lower bounds
   - Competitive algorithms
   - Simulations results

3 Parallel scheduling of DAGs under memory constraints
   - Model and maximum parallel memory
   - Efficient scheduling with bounded memory
   - Simulation results
Simulations

Heuristics (makespan normalized by offline HEFT’s)

- **EFT** (= **MixEFT** as EFT better than QA here)
- **QA** (switch at $\sqrt{m/k}$)
- **ER-LS** (= QA + greedy rule: slightly more tasks on GPUs)
- **Quickest** (= QA with switch at 1: more tasks on GPUs)
- **Ratio** (= QA with switch at $m/k$: more tasks on CPUs)

Datasets for $m = 20$ CPUs and $k = 2$ GPUs

- **Cholesky** 4 types of tasks
- **Synthetic** STG set, 300 tasks, random GPU acceleration ($\mu = \sigma = 15$)
- **Ad-hoc** one chain & independent tasks
Results for Cholesky graphs (lower is better)

$m$ CPUs, $k$ GPUs

\[
\frac{m}{k} = 10 \quad \sqrt{\frac{m}{k}} \approx 3.3
\]

\[
\frac{\text{CPU time}}{\text{GPU time}} \in \{28, 26, 11, \frac{2}{\text{POTRF}}\}
\]

Algorithm
- EFT = MixEFT
- QA
- ER-LS
- RATIO
- QUICKEST
Results for synthetic graphs (lower is better)
Results for 300-tasks ad-hoc graphs (lower is better)

Algorithm
- EFT = MixEFT
- QA
- ER-LS
- Ratio
- Quickest

CPU

GPU

OPT (left, $\bar{p}_i \approx p_i$)

OPT (middle)

OPT (right, $\bar{p}_i \approx \frac{m}{k} p_i$)
Conclusion of this project

Summary

- No online algo. is $< \sqrt{m/k}$ - competitive
  Additional knowledge or power hardly helps

- QA: $(2\sqrt{m/k} + 1)$ - competitive
  MIXEFT: compromise effectiveness / guarantees

- Extended to multiple types of processors (not in this talk)

Perspectives

- Low-cost offline algorithm with constant ratio
- Communication times
- Parallel tasks
Outline

1. PhD thesis overview

2. Online scheduling of DAGs on hybrid platforms
   - Lower bounds
   - Competitive algorithms
   - Simulations results

3. Parallel scheduling of DAGs under memory constraints
   - Model and maximum parallel memory
   - Efficient scheduling with bounded memory
   - Simulation results
Parallel scheduling of DAGs under memory constraints

**DAGs of tasks**
- Describe many applications
- Used by increasingly popular runtime schedulers
  
  \( (XKAAPI, \text{StarPU, StarSs, ParSEC, } \ldots) \)

**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. PhD thesis overview
2. Online scheduling of DAGs on hybrid platforms
   - Lower bounds
   - Competitive algorithms
   - Simulations results
3. Parallel scheduling of DAGs under memory constraints
   - Model and maximum parallel memory
   - Efficient scheduling with bounded memory
   - Simulation results
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

![Task graph diagram]

$M_{used} = 0$
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

![Task graph with weights and memory usage](image)

$M_{used} = 3$
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

![Diagram of a task graph with edges and vertices labeled with numbers. The text below the diagram states $M_{used} = 3$.]
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

![Task graph diagram]

$\sum_{i=1}^{n} w_i = 9$
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

![Task Graph]

$M_{used} = 9$
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

\[
\begin{align*}
A &: w_A = 10 \\
A_1 &: w_{A_1} = 10 \\
A_2 &: w_{A_2} = 0
\end{align*}
\]
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 \(T\)

Topological cut \(\leftrightarrow\) execution state where \(T\) nodes are not started yet

\[ M_{used} = 12 \]
Computing the maximum topological cut

**Literature**

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

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

- Any graph: integer solution \Leftrightarrow \text{cut}
Maximum topological cut – using LP

A classical min-cut LP formulation

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

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

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

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

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

A classical min-cut LP formulation

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

- 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 \(G^{diff}\) 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 $\text{max}^{\text{diff}}$ in $G^{\text{diff}}$
4. $F - \text{max}^{\text{diff}}$ 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., $O(|V|^2|E|)$
Outline

1 PhD thesis overview

2 Online scheduling of DAGs on hybrid platforms
   - Lower bounds
   - Competitive algorithms
   - Simulations results

3 Parallel scheduling of DAGs under memory constraints
   - Model and maximum parallel memory
   - Efficient scheduling with bounded memory
   - Simulation results
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*

![Task Graph]

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

![Diagram of a directed acyclic graph (DAG) with labeled edges and a critical path highlighted in red. The available memory $M_{\text{available}} = 10$.](image-url)
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.*

**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 [Sbirlea 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 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 \(Inputs(u) + Outputs(v)\)
- **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 $\min\{\text{Inputs}(u), \text{Outputs}(v)\}$
Outline

1. PhD thesis overview

2. Online scheduling of DAGs on hybrid platforms
   - Lower bounds
   - Competitive algorithms
   - Simulations results

3. Parallel scheduling of DAGs under memory constraints
   - Model and maximum parallel memory
   - Efficient scheduling with bounded memory
   - Simulation results
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)

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

- **y**: \(\text{CP / original CP} \rightarrow \text{lower is better}\)

- **MinLevels** performs best, but might fail
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
Conclusion of this project

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