Task Graph Scheduling on Modern Computing Platforms

Bertrand SIMON

ENS de Lyon, France

Bremen — May 2018

Presentation

3rd-year PhD student in ENS Lyon, ROMA team

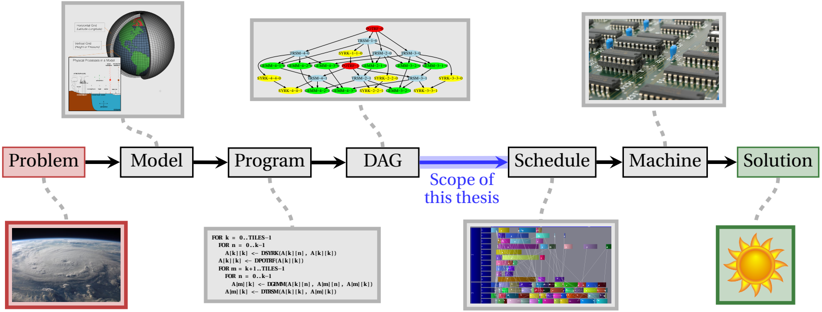
- Advisors: Loris Marchal & Frédéric Vivien
- Defending on July 4th

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

Task Graph Scheduling on Modern Computing Platforms



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 → complex algorithms
- Linear algebra: similar tasks so similar speedup
  design of low-complexity algorithms

Guermouche, Marchal, Simon, Vivien — EuroPar 2016

- Existing speedup function [Prasanna & Musicus 1996]

Marchal, Simon, Sinnen, Vivien — TPDS 2018

- Design a tunable two-threshold roofline speedup function
- High accuracy on extensive benchmarks for linear algebra kernels

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 — **IPDPS 2018**

- Prevent dynamic schedulers from exceeding memory

— Second focus of this talk

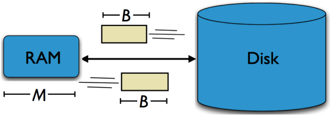
Second setting: insufficient memory, I/Os necessary

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

Marchal, McCauley, Simon, Vivien — **IPDPS Workshops 2017**

- Minimize I/Os in task trees with splittable files; complexity open

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

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

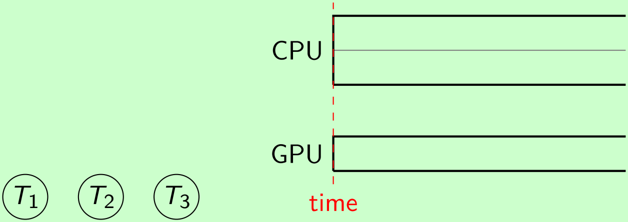
Model and toy example

Model

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

Objective: minimize makespan

Example (2 CPUs, 1 GPU)



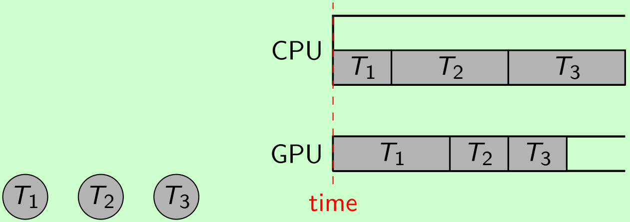
Model and toy example

Model

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

Objective: minimize makespan

Example (2 CPUs, 1 GPU)



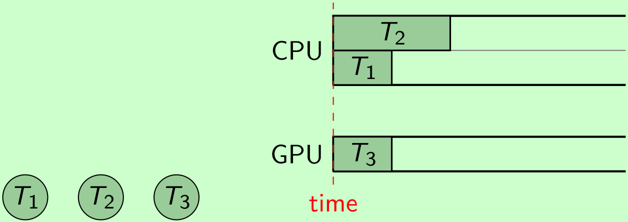
Model and toy example

Model

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

Objective: minimize makespan

Example (2 CPUs, 1 GPU)



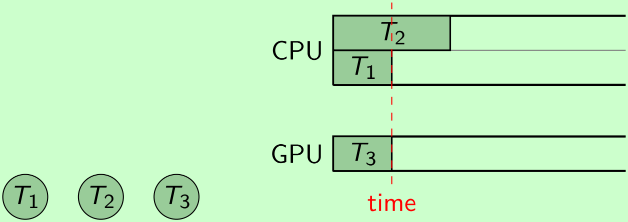
Model and toy example

Model

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

Objective: minimize makespan

Example (2 CPUs, 1 GPU)



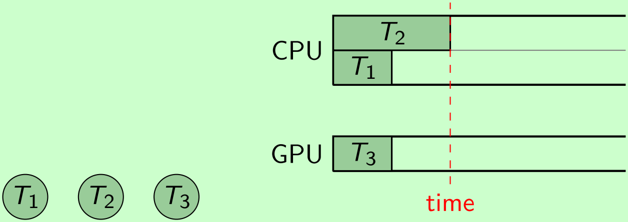
Model and toy example

Model

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

Objective: minimize makespan

Example (2 CPUs, 1 GPU)



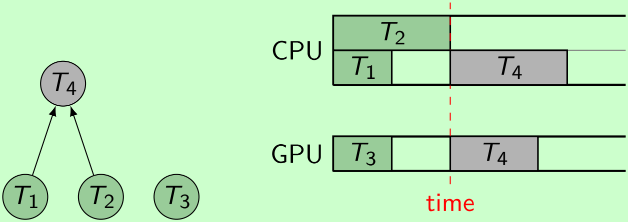
Model and toy example

Model

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

Objective: minimize makespan

Example (2 CPUs, 1 GPU)



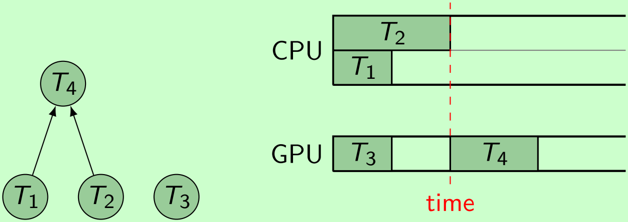
Model and toy example

Model

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

Objective: minimize makespan

Example (2 CPUs, 1 GPU)



Related work

m CPUs, *k* GPUs

Existing offline algorithms (NP-Complete)

- Independent tasks:
  - $\frac{4}{3} + \frac{1}{3k}$ - approx [Bleuse, Kedad-Sidhoum, Monna, Mounié, Trystram 2015]
    Expensive PTAS [Bonifaci, Wiese 2012]
  - Low-complexity: 2 - approx [Canon, Marchal, Vivien 2017]
    3.41 - approx [Beaumont, Eyraud-Dubois, Kumar 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

m CPUs, *k* GPUs

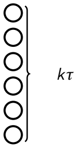
Theorem

No online algorithm $\mathscr{A}$ 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 $\left\{\overline{p_i} = \tau \; ; \; \underline{p_i} = 1\right\}$: $\mathscr{A}$ needs a time τ

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



$\square$

Lower bound

m CPUs, k GPUs

Theorem

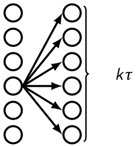
No online algorithm $\mathscr{A}$ is $< \sqrt{m/k}$-competitive for any m, k.

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

Phase 1 - $k\tau$ independent tasks $\left\{\overline{p_i} = \tau \ ; \ \underline{p_i} = 1\right\}$: $\mathscr{A}$ needs a time τ

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

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



Lower bound

m CPUs, *k* GPUs

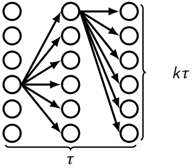
Theorem

No online algorithm $\mathscr{A}$ 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 $\left\{\overline{p_i} = \tau \; ; \; \underline{p_i} = 1\right\}$: $\mathscr{A}$ needs a time τ
- 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
$k = 2$, $n = 3$



□

Lower bound

m CPUs, k GPUs

Theorem

No online algorithm $\mathscr{A}$ 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 $\left\{\overline{p_i} = \tau \; ; \; \underline{p_i} = 1\right\}$: $\mathscr{A}$ needs a time τ

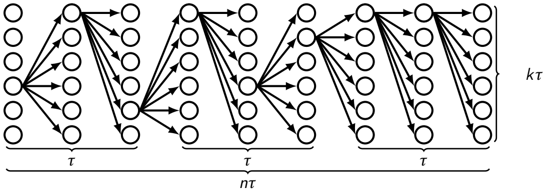
Phase 2 - same as phase 1, but are successors of the last task

Phase 3 - same as phase 2, but are successors of the last task

Phase x - ...

$\Longrightarrow$ Makespan obtained by $\mathscr{A}$: $n\tau^2$

Graph with
$k = 2, \; n = 3$



$\square$

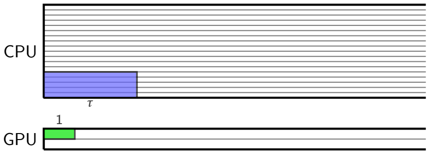
Lower bound

m CPUs, k GPUs

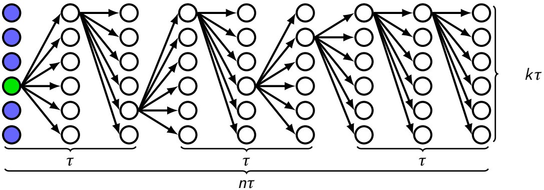
Theorem

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

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



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



□

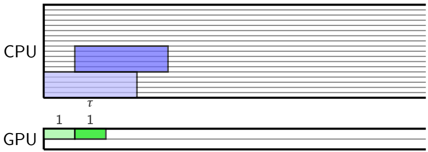
Lower bound

m CPUs, *k* GPUs

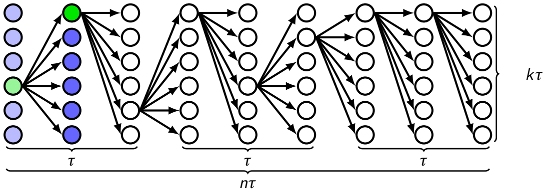
Theorem

No online algorithm $\mathscr{A}$ *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

m CPUs, k GPUs

Theorem

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

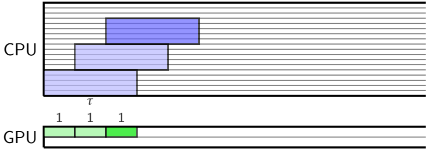
Proof (where $\tau = \sqrt{m/k} = 3$): graph built in $n\tau$ phases.

τ phases

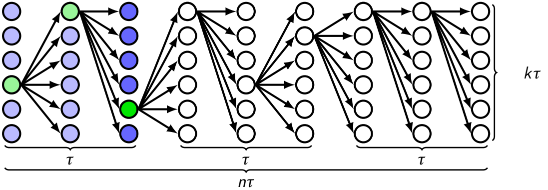
OPT $\Rightarrow 2\tau$

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

Lower bound: $\frac{1}{2}\tau$



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



$\square$

Lower bound

m CPUs, k GPUs

Theorem

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

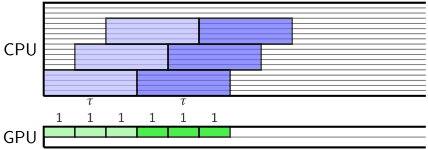
Proof (where $\tau = \sqrt{m/k} = 3$): graph built in $n\tau$ phases.

2τ phases

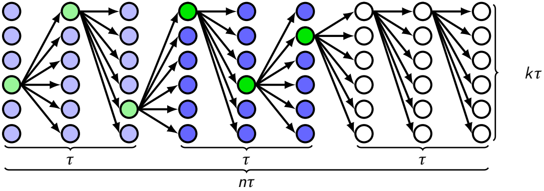
OPT $\Rightarrow 3\tau$

$\mathscr{A} \Rightarrow 2\tau^2$

Lower bound: $\frac{2}{3}\tau$



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



□

Lower bound

m CPUs, k GPUs

Theorem

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

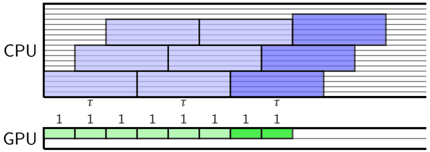
Proof (where $\tau = \sqrt{m/k} = 3$): graph built in $n\tau$ phases.

$n\tau$ phases

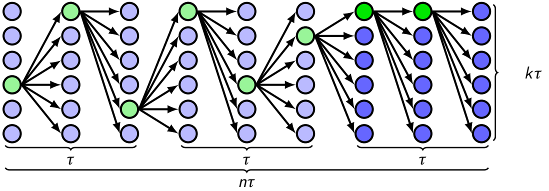
OPT $\Rightarrow (n+1)\tau$

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

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



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



$\square$

Generalized lower bounds

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

Precomputed information

- Bottom-level (≈ 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 ($4\sqrt{m/k}$-competitive, [Amarís et al.])

m CPUs, k GPUs

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 $\overline{p_i}$:

- put T_i on GPU

Otherwise:

- if $\frac{\overline{p_i}}{\underline{p_i}} \leq \sqrt{\frac{m}{k}}$: put it on CPU
- else : put it on GPU

Our proposition: QA (Quick Allocation) algorithm

m CPUs, k GPUs

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 $\overline{p_i}$:~~

- ~~put T_i on GPU~~

~~Otherwise:~~

- if $\frac{\overline{p_i}}{\underline{p_i}} \leq \sqrt{\frac{m}{k}}$: put it on CPU
- else : put it on GPU

Our proposition: QA (Quick Allocation) algorithm

m CPUs, k GPUs

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 $\overline{p_i}$:~~

- ~~put T_i on GPU~~

~~Otherwise:~~

- if $\dfrac{\overline{p_i}}{\underline{p_i}} \leq \sqrt{\dfrac{m}{k}}$: 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?

m CPUs, k GPUs

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: $\geq (\frac{m}{k}+2)$ OPT

Can we have both benefits? MIXEFT

- `Run EFT and simulate QA;`
  `When EFT is` λ `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

m CPUs, *k* GPUs

Heuristics (makespan normalized by offline HEFT's)

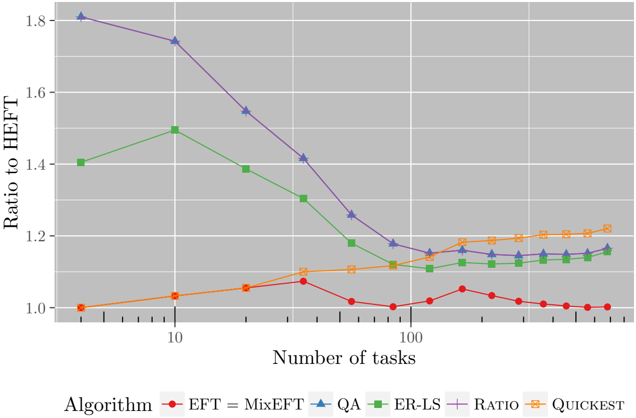
- EFT (= MIXEFT as EFT better than QA here)
- QA (switch at $\sqrt{m/k}$)
- ER-LS (= QA + greedy rule: slightly more tasks on GPUs)
- QUICKEST (= QA with switch at 1: more tasks on GPUs)
- RATIO (= QA with switch at m/k: more tasks on CPUs)

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

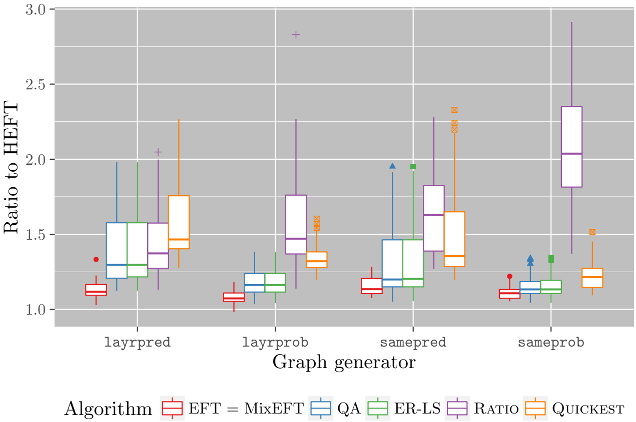
Results for Cholesky graphs (lower is better)

m CPUs, k GPUs

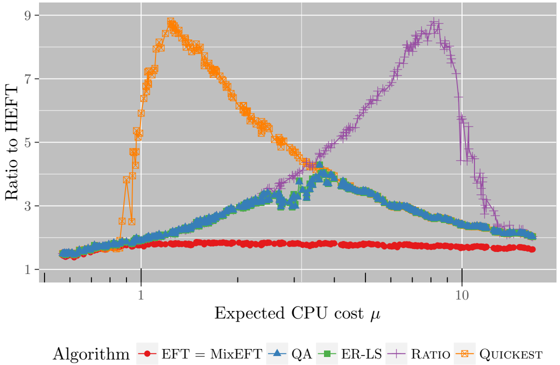


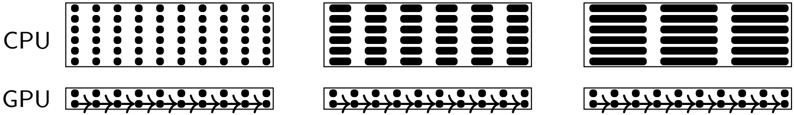
$$\frac{m}{k} = 10 \qquad \sqrt{\frac{m}{k}} \approx 3.3 \qquad \frac{\text{CPU time}}{\text{GPU time}} \in \{28, 26, 11, \underbrace{2}_{\text{POTRF}}\}$$

Results for synthetic graphs (lower is better)



Results for 300-tasks ad-hoc graphs (lower is better)





OPT (left, $\overline{p_i} \approx \underline{p_i}$) OPT (middle) OPT (right, $\overline{p_i} \approx \frac{m}{k}\underline{p_i}$)

Conclusion of this project

m CPUs, k GPUs

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)

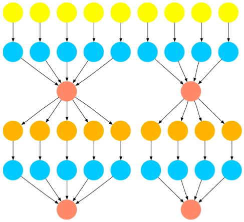
Perspectives

- 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, 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. 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

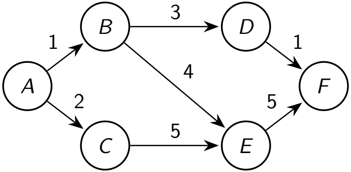
Task graph weights

- 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

$M_{used} = 0$



Memory model

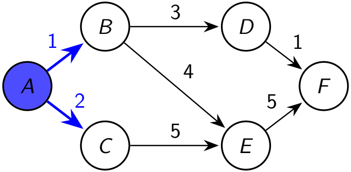
Task graph weights

- 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

$M_{used} = 3$



Memory model

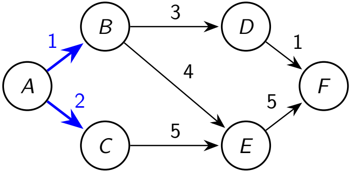
Task graph weights

- 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

$M_{used} = 3$



Memory model

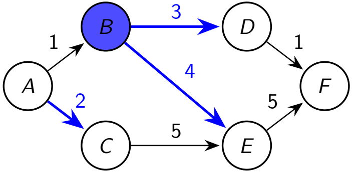
Task graph weights

- 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

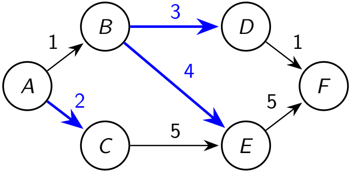
Task graph weights

- 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

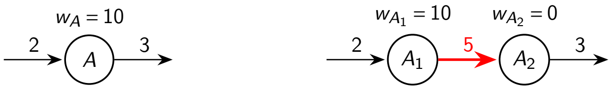
- Vertex w_i : estimated task duration
- Edge $m_{i,j}$: data size

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



Memory model

Task graph weights

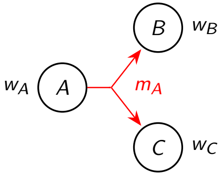
- Vertex w_i : estimated task duration
- Edge $m_{i,j}$: data size

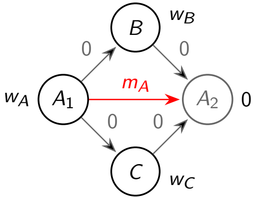
Simple memory model

- 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





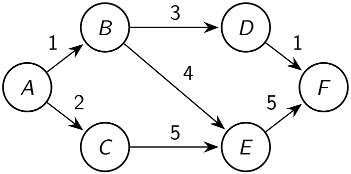
Computing the maximum memory peak

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

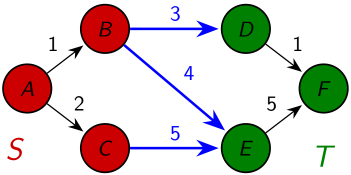
Two equivalent quantities (in our model)

- 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

$M_{used} = 12$



Topological cut $\longleftrightarrow$ 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

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

- Any graph: integer solution $\iff$ 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

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

- Any graph: integer solution $\Longleftrightarrow$ cut
- Modify LP: 'min' $\to$ 'max' ; '$\geq$' $\to$ '$=$'

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

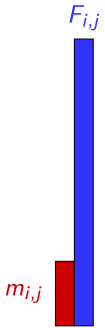
- Any rounding of the p_i's works (large $\in S$, small $\in T$)

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 → maximum topological cut



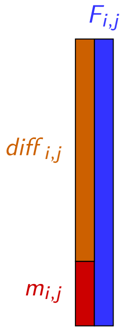
Complexity: same as maximum flow, e.g., $O(|V|^2|E|)$

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



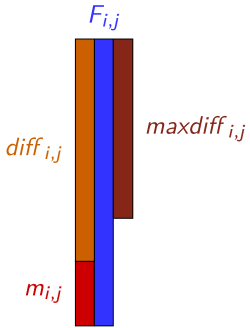
Complexity: same as maximum flow, e.g., $O(|V|^2|E|)$

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



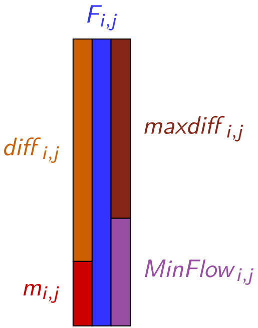
Complexity: same as maximum flow, e.g., $O(|V|^2|E|)$

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., $O(|V|^2|E|)$

Outline

Coping with limited memory

Problem

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

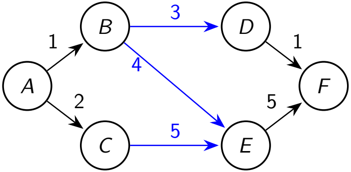
Coping with limited memory

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*



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

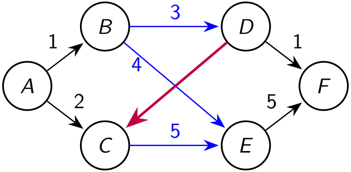
Coping with limited memory

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*



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

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
- $MaxTopologicalCut(G') \leq M$
- $CritPath(G')$ is minimized

Theorem (Sethi 1975)

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

Theorem

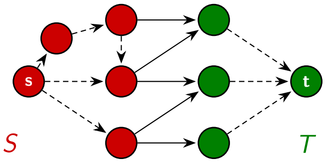
PartialSerialization *is NP-hard given a memory-efficient schedule.*

Optimal solution computable by an ILP (builds transitive closure)

Heuristic solutions for PartialSerialization

Framework – ***inspired by [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 σ
- Step 3: select first vertex $u \in T$, last vertex $v \in S$ in σ
  Always succeeds if $memory(\sigma) \leq M$

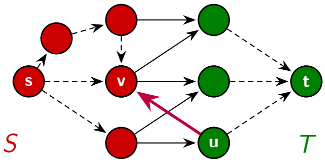
MaxSize Maximize $Inputs(u) + Outputs(v)$

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 σ
- Step 3: select first vertex $u \in T$, last vertex $v \in S$ in σ
  Always succeeds if $memory(\sigma) \leq M$

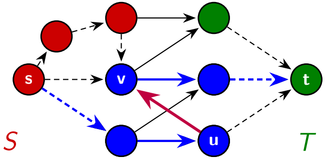
MaxSize Maximize $Inputs(u) + Outputs(v)$

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 σ
- Step 3: select first vertex $u \in T$, last vertex $v \in S$ in σ
  Always succeeds if $memory(\sigma) \leq M$

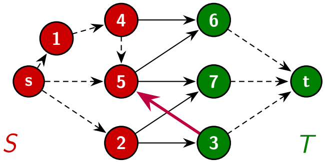
MaxSize Maximize $Inputs(u) + Outputs(v)$

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 $\mathit{TopLevel}(u) + \mathit{BottomLevel}(v)$

RespectOrder
- Pre-compute a *good* sequential schedule σ
- Step 3: select first vertex $u \in T$, last vertex $v \in S$ in σ
  Always succeeds if $\mathit{memory}(\sigma) \leq M$

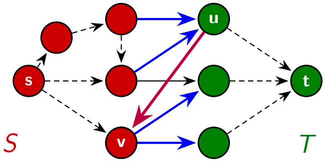
MaxSize Maximize $\mathit{Inputs}(u) + \mathit{Outputs}(v)$

MaxMinSize Maximize $\min\{\mathit{Inputs}(u)\ ,\ \mathit{Outputs}(v)\}$

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 σ
- Step 3: select first vertex $u \in T$, last vertex $v \in S$ in σ
  Always succeeds if memory$(\sigma) \leq M$

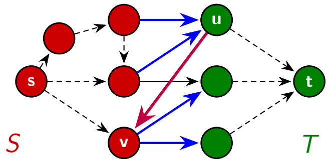
MaxSize Maximize $Inputs(u) + Outputs(v)$

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 σ
- Step 3: select first vertex $u \in T$, last vertex $v \in S$ in σ

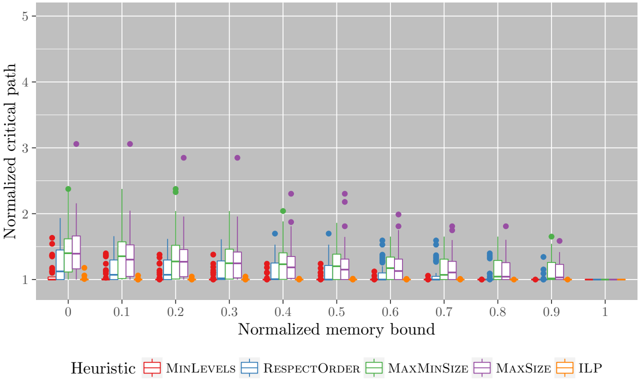
  Always succeeds if $memory(\sigma) \leq M$

MaxSize Maximize $Inputs(u) + Outputs(v)$

MaxMinSize Maximize $\min\{Inputs(u) , Outputs(v)\}$

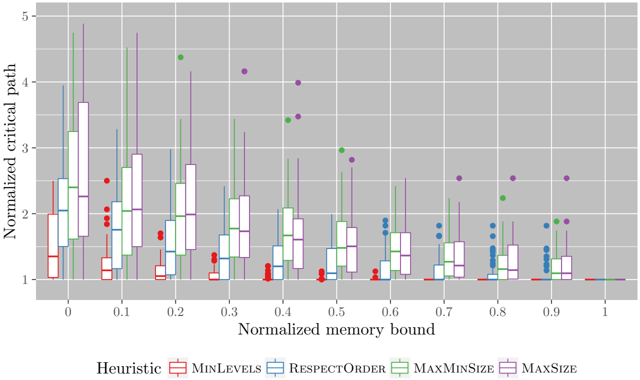
Outline

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



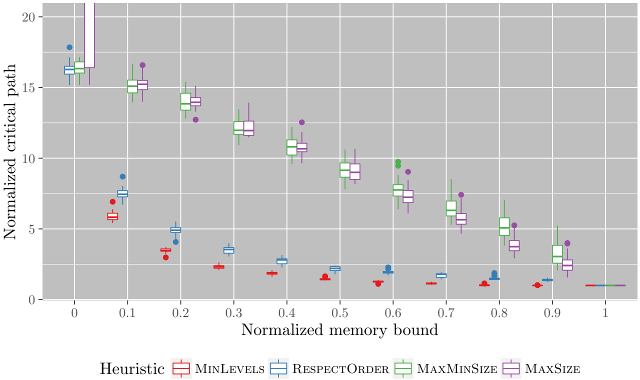
- *x: memory (0 = DFS, 1 = MaxTopCut)*
  median ratio MaxTopCut / DFS ≈ 1.3
- *y: CP / original CP → lower is better*
- MinLevels performs best

Sparse DAGGEN random graphs (25, 50, and 100 nodes)



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

Simulations – Pegasus workflows (LIGO 100 nodes)



- *Median ratio MaxTopCut / DFS ≈ 20*
- MinLevels performs best, 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