Algorithms for Co-scheduling of Kernels on GPUs

Cristiana Bentes, Lionel Eyraud-Dubois

Plenary meeting, Dec. 2020

Co-scheduling on a single GPU

Context

- Scientific computing on GPUs is getting more common
- Multi-stream on GPUs allow to run several kernels in parallel
- Some kernels slow each other (because of resource limitations)
- Modern GPUs start to allow preemption
- How to select which kernels to run together ?

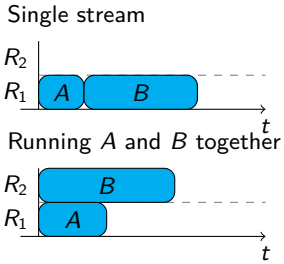
Contributions

- Algorithms to schedule the execution of independent tasks (one task = one kernel)
- Optimal total computation time (*makespan*), minimum number of preemptions
- Or minimum makespan without preemptions
- Result: preemptions reduce makespan by 5-12%, with 5-10% kernels preempted

Co-scheduling on a single GPU

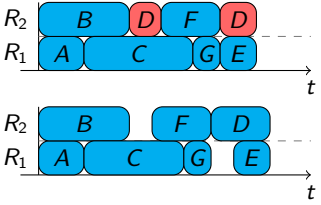
Model: MinMakespan

- We **measure** $S_{i,j}$: speed of task i if executed with j
  During time t, task i makes progress $t \cdot S_{i,j}$.
- Each task i has a processing time p_i



Valid schedules

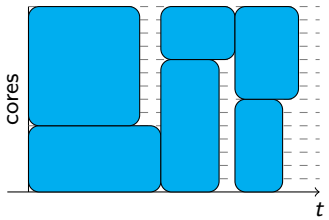
- At most two tasks at a time
- Each task i needs to make total progress p_i
- Minimize the completion time
- Two versions: preemptive or non preemptive



Previous works

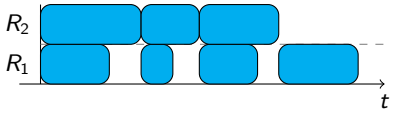
Multicore co-scheduling [Raghavan et al, 2013, 2016, 2018]

- Running two or more tasks together
- Possibility to choose how cores are shared
- Ensures that co-scheduled tasks have similar duration



MaxPair GPU co-scheduling [Wen et al, 2018]

- Group tasks by pairs, run one pair at a time
- Synchronous execution
- Maximum weight matching



Algorithmic analysis

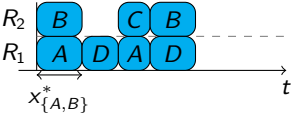
Optimal preemptive schedule

- Optimal makespan by Linear Programming
- $x_{\{i,j\}}$: execution time of i and j together
- $\frac{n(n+1)}{2}$ variables, $n + \frac{n(n+1)}{2}$ constraints
  $\Rightarrow$ at most n non-zero variables

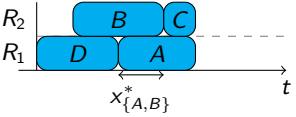
$$\begin{aligned} \text{min.} \quad & \sum x_{\{i,j\}} \\ \text{s.t.} \quad & \forall i, \quad \sum_j x_{\{i,j\}} S_{i,j} \geq p_i \\ & \forall i,j, \quad x_{\{i,j\}} \geq 0 \end{aligned}$$

Result: schedule with at most n preemptions

At most $2n$ pieces appear, the first piece of a task is not a preemption



Same solution, fewer preemptions



Further optimization:

- Minimize the number of preemptions
- Non preemptive schedule with minimum makespan

Introduce the co-execution graph

Graph $G(x^*)$ from an optimal solution x^*

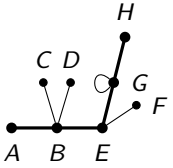
- one vertex per task, edge from i to j if and only if $x^*_{\{i,j\}} > 0$ (can have self-loops)
- $G(x^*)$ is a set of pseudo-trees: k vertices, at most k edges, thus at most one cycle

Lemma: connection between graphs and schedules (see next slide)

It is possible to provide a non-preemptive schedule for a solution x^*
if and only if $G(x^*)$ is a caterpillar graph

A caterpillar graph is

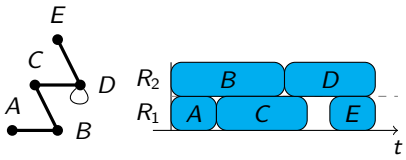
- an *internal* path P,
- with degree-one nodes connected to it.



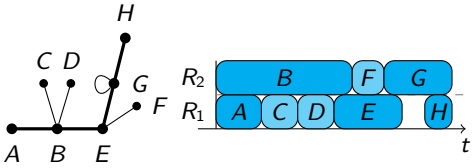
Which graphs can be turned into schedules

Graph $G(x^*)$: one vertex per task, edge $i \leftrightarrow j$ iff $x^*_{\{i,j\}} > 0$

- $G(x^*)$ is a pseudo-tree
- Non-preemptive schedule $\Leftrightarrow$ G is a caterpillar



(a) Simple path

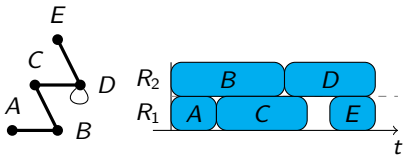


(b) Caterpillar graph

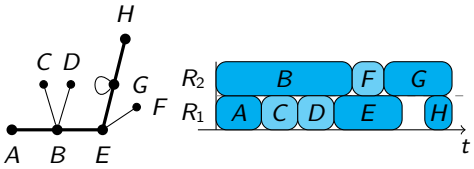
Which graphs can be turned into schedules

Graph $G(x^*)$: one vertex per task, edge $i \leftrightarrow j$ iff $x^*_{\{i,j\}} > 0$

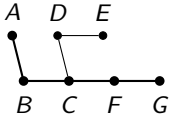
- $G(x^*)$ is a pseudo-tree
- Non-preemptive schedule $\Leftrightarrow$ G is a caterpillar



(a) Simple path



(b) Caterpillar graph

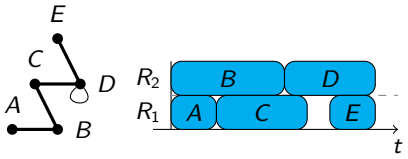


(c) More complex tree

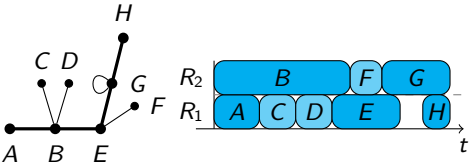
Which graphs can be turned into schedules

Graph $G(x^*)$: one vertex per task, edge $i \leftrightarrow j$ iff $x^*_{\{i,j\}} > 0$

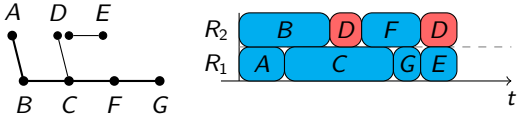
- $G(x^*)$ is a pseudo-tree
- Non-preemptive schedule $\Leftrightarrow$ G is a caterpillar



(a) Simple path



(b) Caterpillar graph



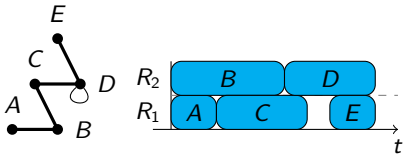
(c) More complex tree

preemptive

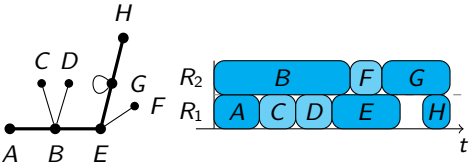
Which graphs can be turned into schedules

Graph $G(x^*)$: one vertex per task, edge $i \leftrightarrow j$ iff $x^*_{\{i,j\}} > 0$

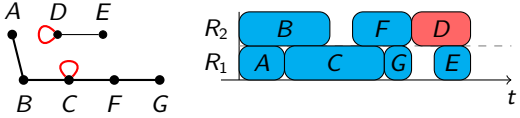
- $G(x^*)$ is a pseudo-tree
- Non-preemptive schedule $\Leftrightarrow$ G is a caterpillar



(a) Simple path



(b) Caterpillar graph

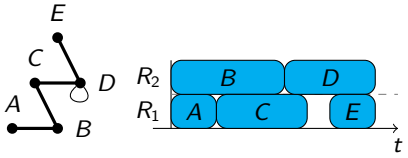


(c) More complex tree, non-preemptive

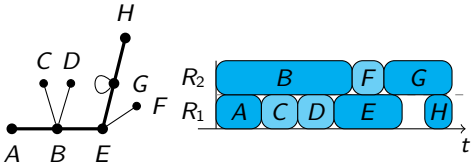
Which graphs can be turned into schedules

Graph $G(x^*)$: one vertex per task, edge $i \leftrightarrow j$ iff $x^*_{\{i,j\}} > 0$

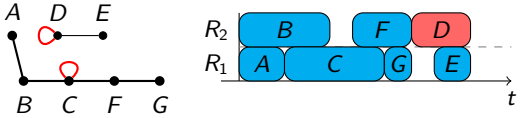
- $G(x^*)$ is a pseudo-tree
- Non-preemptive schedule $\Leftrightarrow$ G is a caterpillar



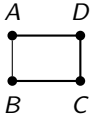
(a) Simple path



(b) Caterpillar graph



(c) More complex tree, non-preemptive

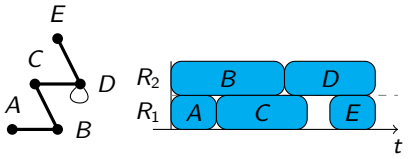


(d) Cycle

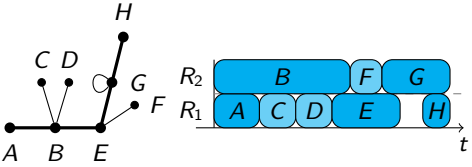
Which graphs can be turned into schedules

Graph $G(x^*)$: one vertex per task, edge $i \leftrightarrow j$ iff $x^*_{\{i,j\}} > 0$

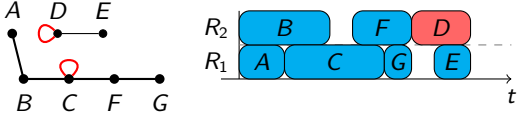
- $G(x^*)$ is a pseudo-tree
- Non-preemptive schedule $\Leftrightarrow$ G is a caterpillar



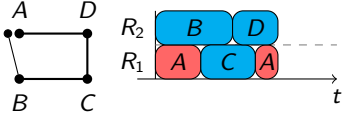
(a) Simple path



(b) Caterpillar graph



(c) More complex tree, non-preemptive

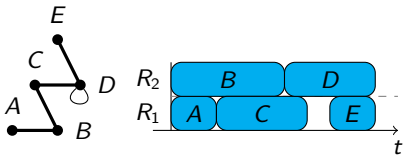


(d) Cycle preemptive

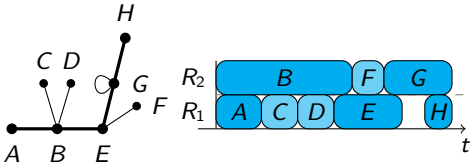
Which graphs can be turned into schedules

Graph $G(x^*)$: one vertex per task, edge $i \leftrightarrow j$ iff $x^*_{\{i,j\}} > 0$

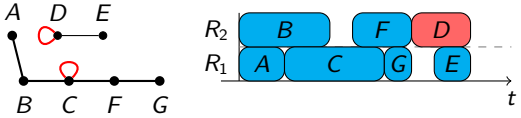
- $G(x^*)$ is a pseudo-tree
- Non-preemptive schedule $\Leftrightarrow$ G is a caterpillar



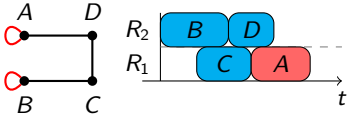
(a) Simple path



(b) Caterpillar graph



(c) More complex tree, non-preemptive



(d) Cycle, non-preemptive

We now have graph problems

Two possible operations

- Splitting a node: it appears at two places in the schedule ($\Rightarrow$ Preemption)
- Breaking an edge and replace it with loops at each endpoint

Minimizing preemption becomes CaterpillarSplit

Split a minimum number of nodes to obtain vertex-disjoint caterpillars.

Non-preemptive scheduling becomes EdgeBreaking

Break a minimum weight set of edges to obtain a set of caterpillar graphs.

Main contributions (details and proofs are in the paper)

- We reduce EdgeBreaking to PathCover and obtain an optimal algorithm
- We propose and prove a greedy algorithm to solve CaterpillarSplit (first on trees, then on pseudo-trees)

Solving MinMakespan problem

MinMakespan

Given processing times p_i and co-execution speeds $S_{i,j}$
compute a schedule to minimize makespan and number of preemptions

Our algorithm

- With Linear Programming, get a schedule of optimal makespan
- With CaterpillarSplit, minimize the number of preemptions for this schedule
- With EdgeBreaking, get a non-preemptive schedule with minimal overhead

- Not optimal: the LP has several optimal solutions
- Another solution of LP may allow even fewer preemptions or lower overhead

Second contribution

MinMakespan is a NP-hard problem

Experimental setting

Input data: measurements from 60 kernels from real applications [Carvalho et al, 2017]
$\Rightarrow$ 3540 pairwise measurements

Instance generation for n tasks, n from 10 to 500

- Pick a kernel type for each of the n tasks
- `uniform`: pick each kernel with uniform probability
- `weighted`: pick each kernel with probability inversely proportional to its duration
- `random duration`: pick kernels uniformly, pick duration at random in $[0, 10]$
- 15 instances for each case

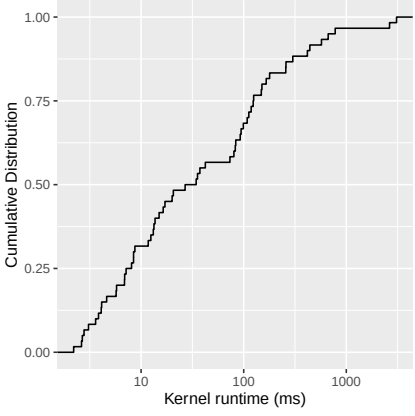
Algorithms considered

- sequential, MaxPair [Wen et al, 2018], CaterpillarSplit, EdgeBreaking
- Magnified preemption costs: 20% of average kernel execution time

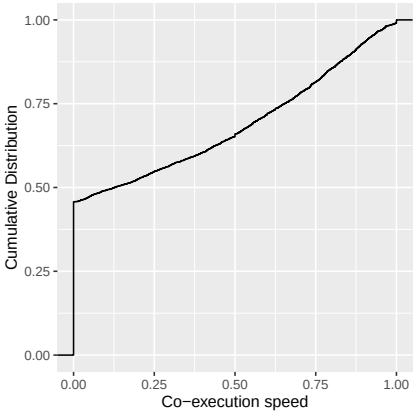
Measurements [Carvalho et al, 2017]

- 60 kernels from different applications, measured interference between each pair

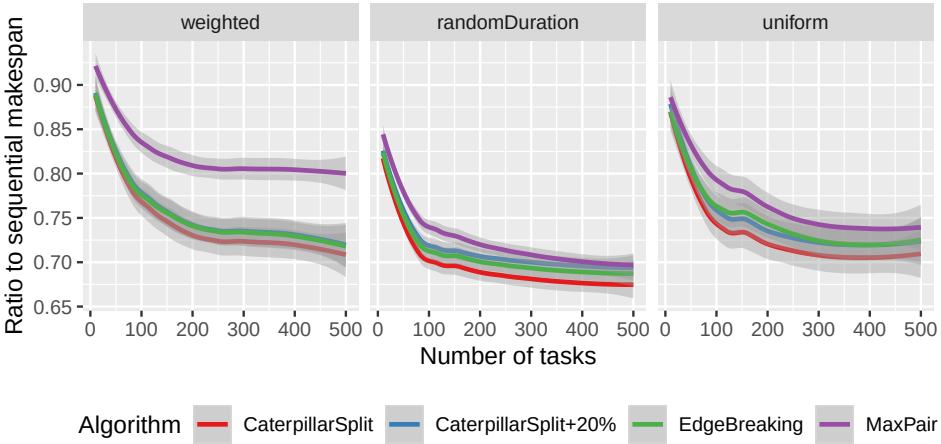
Distribution of kernel runtimes



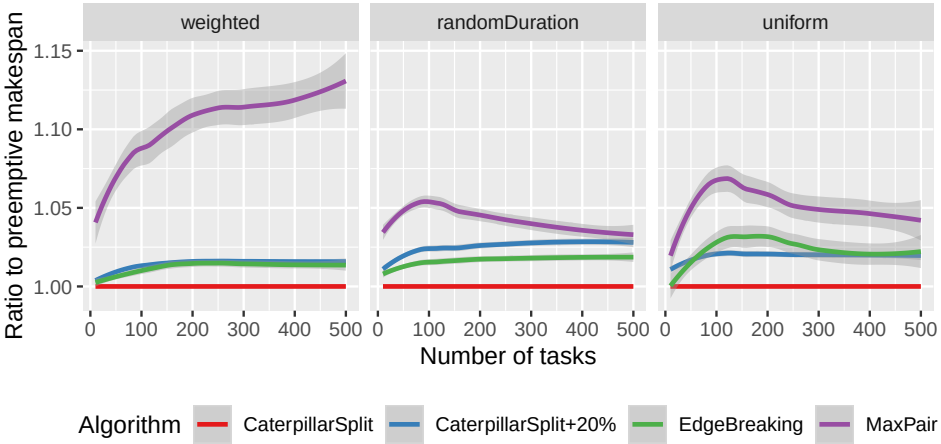
Distribution of co-execution speeds



Results: makespan



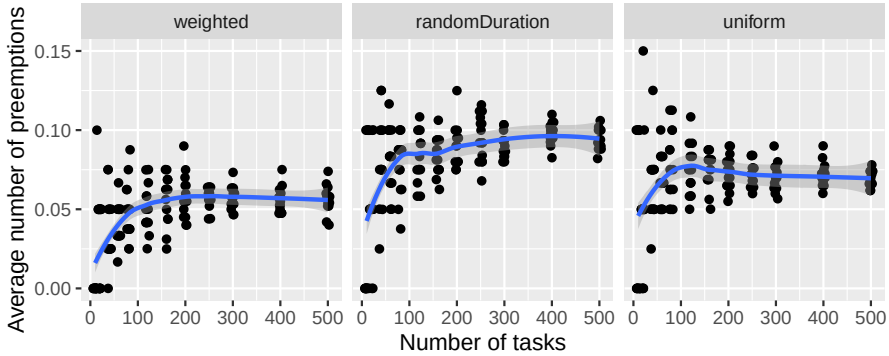
Results: makespan



Results: makespan

- Two-stream execution (MaxPair) improves over sequential by 20-30%
- Using preemptions (CaterpillarSplit) reduces further:
  - 10-12% for `weighted`
  - 5-7% for `uniform`
  - 3-5% for `random duration`
- CaterpillarSplit is still more efficient with 20% overhead
- About the same benefit as EdgeBreaking

Results: number of preemptions



CaterpillarSplit needs to preempt only 5-10% tasks on average

Conclusions

- Adressed a GPU co-scheduling problem, with and without preemptions
- Solved corresponding graph problems: CaterpillarSplit, EdgeBreaking
- Very efficient solutions in practice

Perspectives / Open questions

- Consider a preemption overhead, minimize makespan
- Schedule on multiple GPUs (still at most 2 tasks per GPU)
- Online variant, with kernels submitted over time

Thank you for your interest! Questions?