

Are we expecting too much from GPUs?

Denis Trystram

Grenoble Institute of Technology
and Institut Universitaire de France

Warszawa – sept. 11, 2013

- 1 Introduction
- 2 Some basics on scheduling algorithms
- 3 Scheduling on hybrid systems
- 4 Concluding remarks

The GPU wave...

Tianhe-2 and Titan (top of the Top500 ranking) – 4 hybrid machines in the top10.

HPC platforms include a set of:

- Multi-core CPU
- Computing accelerators: GPGPU (General Purpose GPU)

Rough analysis

- Clear interest for improving the performances of many algorithms (mostly regular ones).
- Take some time to tune the existing algorithms for the successive generations of platforms. Some implementations need a complete re-organization of the code.
- Hard to extend efficiently to highly irregular problems.
- May lead to unexpected bad performances.

The GPU wave...

Tianhe-2 and Titan (top of the Top500 ranking) – 4 hybrid machines in the top10.

HPC platforms include a set of:

- Multi-core CPU
- Computing accelerators: GPGPU (General Purpose GPU)

Rough analysis

- Clear interest for improving the performances of many algorithms (mostly regular ones).
- Take some time to tune the existing algorithms for the successive generations of platforms. Some implementations need a complete re-organization of the code.
- Hard to extend efficiently to highly irregular problems.
- May lead to unexpected bad performances.

The GPU wave...

Tianhe-2 and Titan (top of the Top500 ranking) – 4 hybrid machines in the top10.

HPC platforms include a set of:

- Multi-core CPU
- Computing accelerators: GPGPU (General Purpose GPU)

Rough analysis

- Clear interest for improving the performances of many algorithms (mostly regular ones).
- Take some time to tune the existing algorithms for the successive generations of platforms. Some implementations need a complete re-organization of the code.
- Hard to extend efficiently to highly irregular problems.
- May lead to unexpected bad performances.

For the old fellows...

In 1986-87.

I worked on a *T40 hypercube* with 32 nodes consisting in Transputers with efficient FPS vector boards. Hard to program. Thus, for me, programming multi-cores with GPGPU has some taste of "madeleine cake" (Proust).

Objective of the talk

To share some thoughts on how to deal with accelerators.¹

¹Warning: reflecting a personal view...

For the old fellows...

In 1986-87.

I worked on a *T40 hypercube* with 32 nodes consisting in Transputers with efficient FPS vector boards. Hard to program. Thus, for me, programming multi-cores with GPGPU has some taste of "madeleine cake" (Proust).

Objective of the talk

To share some thoughts on how to deal with accelerators.¹

¹Warning: reflecting a personal view...

Informal definition of Scheduling

The **scheduling problem** is to answer the two following questions for each of the n tasks of the DAG which represents an application.

- **Where?** – the computing resources executing it
- **When?** – its date of execution

How to design efficient schedulers for hybrid machines (m CPU cores and k accelerators)?

- The tasks assigned to the GPGPU must be carefully chosen.
- Need generic methods to do the assignment.
- No consensus model:
start with a simplified problem under the classical model (without communications, no precedence relations, all is known, ...)

Informal definition of Scheduling

The **scheduling problem** is to answer the two following questions for each of the n tasks of the DAG which represents an application.

- **Where?** – the computing resources executing it
- **When?** – its date of execution

How to design efficient schedulers for hybrid machines (m CPU cores and k accelerators)?

- The tasks assigned to the GPGPU must be carefully chosen.
- Need generic methods to do the assignment.
- No consensus model:
start with a simplified problem under the classical model (without communications, no precedence relations, all is known, ...)

Content of the talk

Describe and discuss two useful advanced algorithmic techniques:
work stealing and **dual approximation**.

Positioning

We are looking for low cost methods with performance guaranties in standard situations.

Then, analyze the limits coming from the management of heterogeneity.

Performance guarantee (1/2)

Solution value

The scheduling algorithm \mathcal{A} applied on instance I produces a solution σ_I with a value $v(\sigma_I)$ for the considered objective.

Solution quality

For an instance I , the solution with the best value is denoted by σ_I^* . The quality of the solution is measured by the ratio

$$\frac{v(\sigma_I)}{v(\sigma_I^*)}$$

Performance guarantee (2/2)

The performance ratio of algorithm \mathcal{A} is defined by

$$\rho = \max_{\forall I} \frac{v(\sigma_I)}{v(\sigma_I^*)}$$

The performances are usually assessed by a worst case analysis

Worst case versus average case

Advantages of worst case analysis

- no hypothesis on the instances
- no frequency analysis of the instances (required for any average case analysis)
- relevant if ρ is small
- the worst case structures highlights some deficiencies of the algorithm

Notations and background

- n tasks T_i : processing times p_i on CPU and g_i on GPGPU
- m CPU and k GPGPU
- The acceleration factor of a task T_i is the ratio $\alpha_i = \frac{p_i}{g_i}$
- The completion time of T_i is denoted by C_i

makespan

The optimized objective is the completion time of the last finishing task (*makespan*):

$$C_{max} = \max_{1 \leq i \leq n} C_i$$

The optimal makespan is denoted by C_{max}^* .

Seminal Graham's results

Recall basics on scheduling on identical machines

Principle:

- 1 Keep a **list of the tasks** to be done (sorted by priority)
- 2 Apply the following rule: **start as soon as possible** one of the task of the list on **one of the available resources**. The task is one of the **ready task** starting the earliest (first) with the highest priority (second).

Execution order of the tasks

The order of the list is **not** the order of execution. e.g. precedence constraints or resources constraints change the order.

Seminal Graham's results

Recall basics on scheduling on identical machines

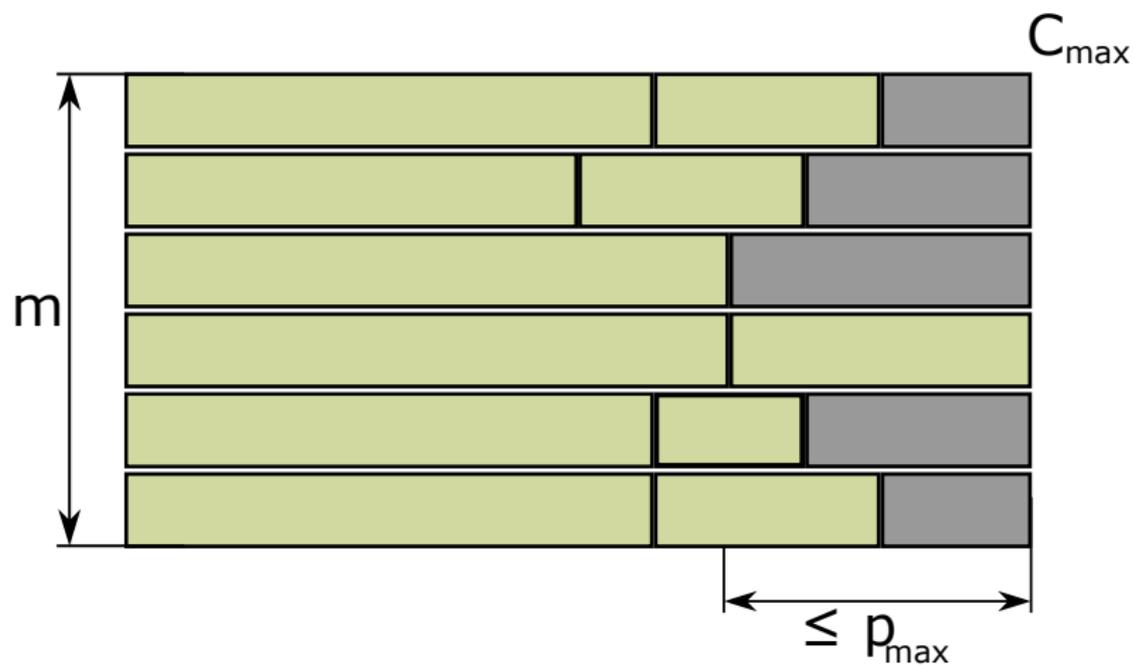
Principle:

- 1 Keep a **list of the tasks** to be done (sorted by priority)
- 2 Apply the following rule: **start as soon as possible** one of the task of the list on **one of the available resources**. The task is one of the **ready task** starting the earliest (first) with the highest priority (second).

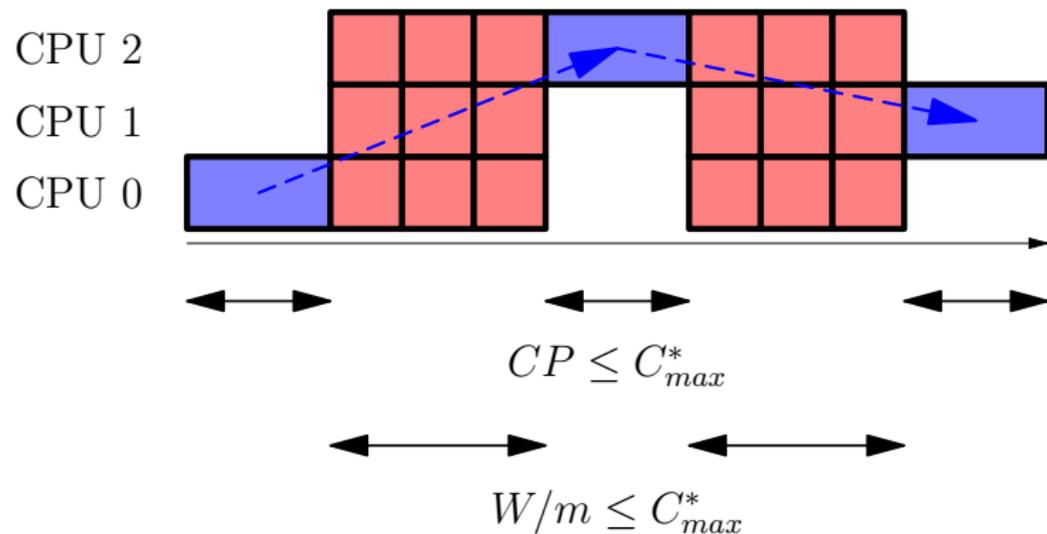
Execution order of the tasks

The order of the list is **not** the order of execution. e.g. precedence constraints or resources constraints change the order.

recall of basic Graham



Graham precedence constraints



Graham: the universal recipe...

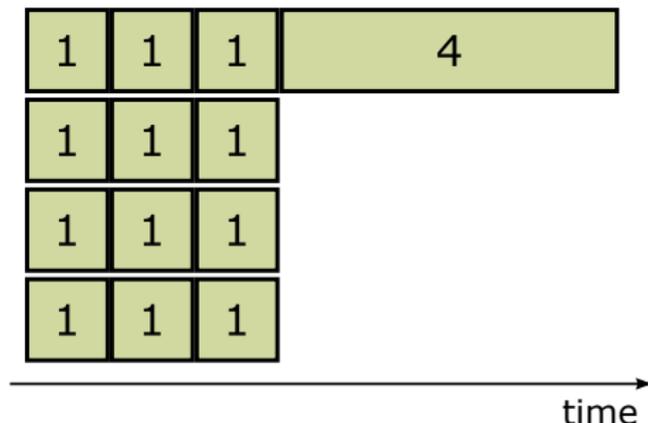
- Easy to extend with communication delays².
- The Graham's rules are dynamic and do not need any information on the ready tasks (not clairvoyant).
- Similarly, this principle holds also for rigid parallel tasks (Graham extension in 1975 with resource constraints).

²well, with a worse ratio for large delays $O(\log n)$...

$$1 + 1 \leq 2$$

As all the previous proofs are based on the same scheme.

1 + 1 The makespan is lower than $\frac{W}{m} + CP$



- 2 $\frac{W}{m}$ and CP are two lower bounds of C_{max}^* . Thus, **keeping busy the processors as much as possible is sufficient to be close to the optimal makespan (at a factor of 2).**

Warning !!!

In the problem of scheduling in hybrid platforms, the execution time of a task depends on its mapping (CPU or GPU)...

Total work and critical path

For any scheduling σ , two values summarize it well:

- the work $W_\sigma = \sum_{\sigma(T_i) \in CPU} p_i + \sum_{\sigma(T_i) \in GPU} g_i$
- the critical path CP_σ .

Coming back on Graham for independent tasks

Partially sorting the tasks in the priority list may improve the results:

m largest first, in decreasing order $\rho = 3/2$

$2m$ largest first, in decreasing order $\rho = 4/3$ ³

Looking more carefully...

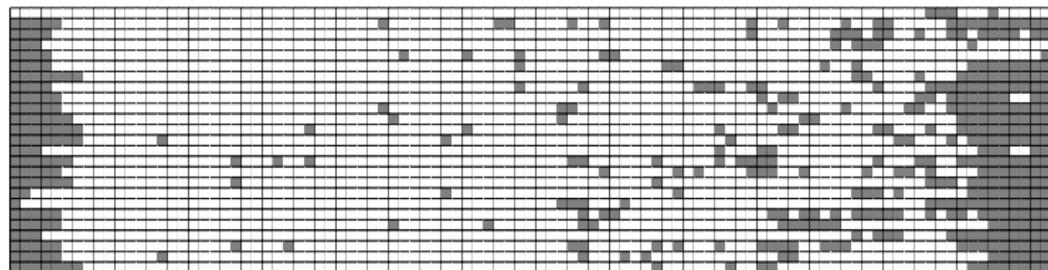
Graham's algorithm is optimal for LPT with few tasks ($n \leq 2m - 1$) and asymptotically optimal for large number of tasks.

³this leads to the well-known LPT approximation ratio 

Work stealing (dynamic settings)

It is basically the same idea as Graham's (keeping busy the processors) but with a distributed implementation⁴.

Thus, we lose some performance, but the solution is still bounded...



simulation of 2000 unit tasks on 25 processors. More precisely, the ratio is additive $O(\log n)$

⁴see recent analysis of Tchiboukdjian-Gast-Trystram in ANOR 

Obviously, WS holds also with precedence relations.

Three main ideas will help to obtain better performances:

- 1 One local list per computing resource (low contention, lock-free dequeue)
- 2 Most of the overhead of the distributed list management is done by the idle processors
- 3 Lazy task creation at steal time (task creation is not free) – on-line

Non-uniform work stealing

Work-stealing was used efficiently in distributed parallel computing and some ideas could also be used for (large) NUMA:

- **adaptive probability of victim selection.** The distance, the hierarchy of the computer, the number of failed steals
- **multiple simultaneous steals.** Simultaneously doing a long distance and a close steal.

Handling dynamic task graphs with precedence constraints.
Thus, any practical scheduling algorithm should manage on-line DAG. How?

Dynamic task graphs

The future tasks are not known:

- all the tasks must be done as efficiently as possible (Work minimization)
- it would be better to execute tasks on the critical path first, but we do not know the critical path...

Handling dynamic task graphs with precedence constraints.
Thus, any practical scheduling algorithm should manage on-line DAG. How?

Dynamic task graphs

The future tasks are not known:

- all the tasks must be done as efficiently as possible (Work minimization)
- it would be better to execute tasks on the critical path first, but we do not know the critical path...

Handling dynamic task graphs with precedence constraints.
Thus, any practical scheduling algorithm should manage on-line DAG. How?

Dynamic task graphs

The future tasks are not known:

- all the tasks must be done as efficiently as possible (Work minimization)
- it would be better to execute tasks on the critical path first, but we do not know the critical path...

Summary

Most scheduling algorithms on identical machines achieve good performance guarantees for almost free.
Most of the scheduling decisions are taken on independent tasks.

What about hybrid machines?

Let us now study how to achieve similar performances on heterogeneous systems?

Summary

Most scheduling algorithms on identical machines achieve good performance guarantees for almost free.
Most of the scheduling decisions are taken on independent tasks.

What about hybrid machines?

Let us now study how to achieve similar performances on heterogeneous systems?

Scheduling on hybrid systems

Description of the Problem.

- $(Pm, Pk) \parallel C_{max}$: m identical CPU and k identical GPGPU. n independent sequential tasks T_1, \dots, T_n .
 T_j has two processing times: p_j on CPU, g_j on GPU.
All the processing times are known.
- Objective: minimize the **makespan**
- Complexity: If $p_j = g_j$ for all tasks
 $(Pm, P1) \parallel C_{max} \Leftrightarrow P \parallel C_{max}$
 \Rightarrow Problem of scheduling with GPU is also NP-hard

Scheduling on hybrid systems

Description of the Problem.

- $(Pm, Pk) \parallel C_{max}$: m identical CPU and k identical GPGPU. n independent sequential tasks T_1, \dots, T_n .
 T_j has two processing times: p_j on CPU, g_j on GPU.
All the processing times are known.
- Objective: minimize the **makespan**
- Complexity: If $p_j = g_j$ for all tasks
 $(Pm, P1) \parallel C_{max} \Leftrightarrow P \parallel C_{max}$
 \implies Problem of scheduling with GPU is also NP-hard

Heterogeneity and list 1

Proposition

$(P1, P1) \parallel C_{max}$: list scheduling algorithm has a ratio larger than the maximum speedup ratio of a task⁵.

Instance:

	T1	T2
p_i (CPU)	α	1
g_i (GPU)	1	1

⁵which may be arbitrarily large

Heterogeneity and list 2

This result still holds for priority like "Largest Acceleration First".

Counter-example:

	T1	T2
p_i (CPU)	100	100
g_i (GPU)	1	1

Any list algorithm schedules this instance with $C_{max} = 100$.

$C_{max}^* = 2$



Heterogeneity and list 3

Contrary to the case of identical machines, **keeping the computing resources busy is not a good idea.**

Some computing resources should sometimes stay idle.

Adaptation for WS

A steal may fail even if some tasks are ready.

As list algorithm is unbounded, let us consider more sophisticated methods like HEFT.

Definition (Heterogeneous Earliest Finish Time First)

- Compute for all tasks i the following priority :
 $Rank_i = \tau_i + \max_{j \in succ(i)} (Comm_{ij} + Rank_j)$
with:
 - $\tau_i = (mp_i + kg_i)/(m + k)$ average execution time
 - $Comm_{ij}$ average communication cost
- A mapping rule: Put the first task on the resource where the it completes the earliest.

Properties

- Low complexity
- May include an accurate communication and execution (prediction) model
- Take into account precedence constraints (and the critical path)
- Need the full graph to compute the correct rank

HEFT (3/5)

HEFT priority rule selects the ready tasks according to the "average" critical path.

HEFT Counter example with independent tasks

- HEFT sorts the tasks by average decreasing execution times.
- let consider m CPU, $k = 1$ GPU

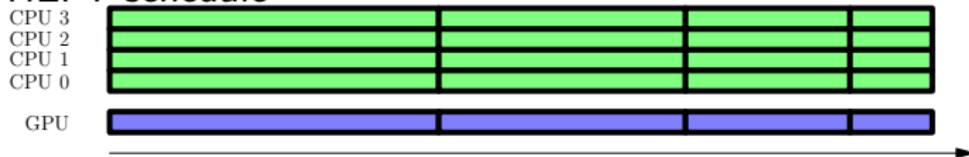
	T1	...	T _m
CPU	ε	...	ε
GPU	$m+2$...	$m+2$

for $i = 0..m-1$

	A_{i1}	B_{i1}	...	B_{im}
CPU	$1 - i/m$	$1 - i/m$...	$1 - i/m$
GPU	$1 - i/m$	$1/m^2$...	$1/m^2$

HEFT (4/5)

■ HEFT schedule



■ Optimal schedule



We obtain an approximation ratio in $\frac{m}{2}$.
How HEFT could be so wrong?

HEFT executes the top priority tasks with respect to the **average critical path**. Some resources will be used less efficiently and thus, the total work will increase.

To keep the total work low, sorting by the acceleration factor would be a better choice (not for the critical path)...

Open question

Does it lead to a constant approximation ratio?

HEFT (5/5)

We obtain a approximation ratio in $\frac{m}{2}$.
How HEFT could be so wrong?

HEFT executes the top priority tasks with respect to the **average critical path**. Some resources will be used less efficiently and thus, the total work will increase.

To keep the total work low, sorting by the acceleration factor would be better choice (not for the critical path)...

Open question

Does it lead to a constant approximation ratio?

HEFT does not take into account one major point of the problem

There are only two types of computing resources

CPU and GPU.

If a task is not executed by a GPU, it has to be executed by a CPU...

This is true also for the optimal solution!

Knapsack

Minimizing the total work is similar to a **knapsack problem**: select the best set of tasks executed on the GPU (and thus the complementary set of tasks executed on the CPU).

But how to choose the knapsack size?

Definition (Dual approximation)

Choose an arbitrary value λ and use it as guess of the optimal solution.

The algorithm has the following binary output:

- ACCEPT:: it provides a schedule with $C_{max} \leq \rho\lambda$
- REJECT :: $C_{max}^* \geq \lambda$

λ value will guide the structure of the solution

How to choose λ ?

Using binary search (with ε accuracy) on the value of λ , we get:

- a solution with $C_{max} = \rho\lambda + \varepsilon$
- and $C_{max}^* \geq \lambda$

Thus the performance guarantee of the solution is $\rho + \varepsilon$.

Structure of the optimal solution

If λ is the optimal value:

- 1 All execution times are lower than λ (Critical path)
- 2 Every computing resource computes at most λ , thus the total work is lower than $(m + k)\lambda$ (Work)

Application with $\rho = 2$

sort sort the tasks by largest acceleration factors

preloading schedule the tasks constrained to a single type of resources ($p_i > \lambda$ or $g_i > \lambda$)

knapsack schedule the GPU with the tasks by decreasing acceleration factor until the sum of work on GPU is larger than $k\lambda$.

The GPU are filled less than 2λ .

CPU filling schedule all remaining tasks using a list algorithm.
The CPU are filled less than 2λ

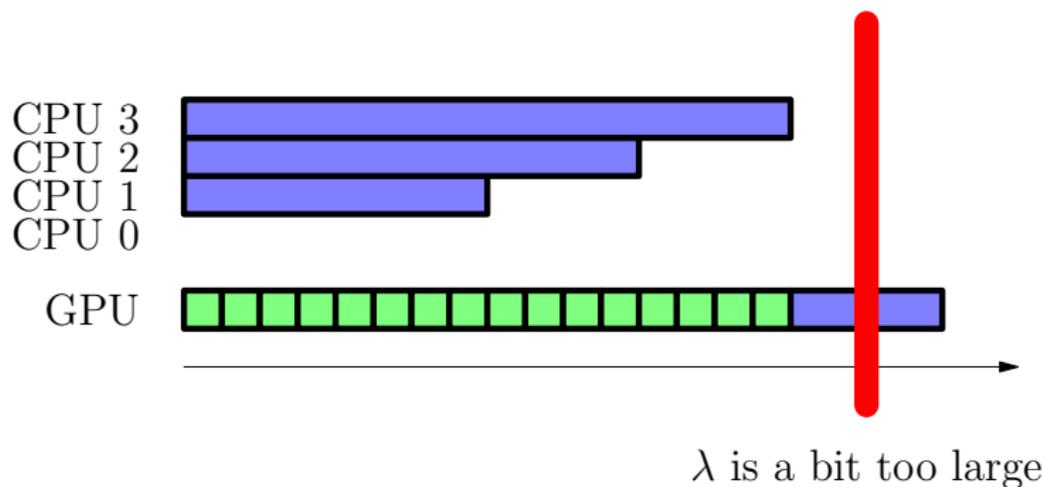
- REJECT
 - $\exists i, p_i > \lambda$ and $g_i > \lambda$
 - Total work larger than $(m+k)\lambda$
 - One CPU is filled more than 2λ
- Knapsack work is lower than the work of the optimal solution
If $\lambda \geq C_{max}^$, $W_\sigma \leq W_{\sigma^*}$*

coming back to the HEFT counter-example

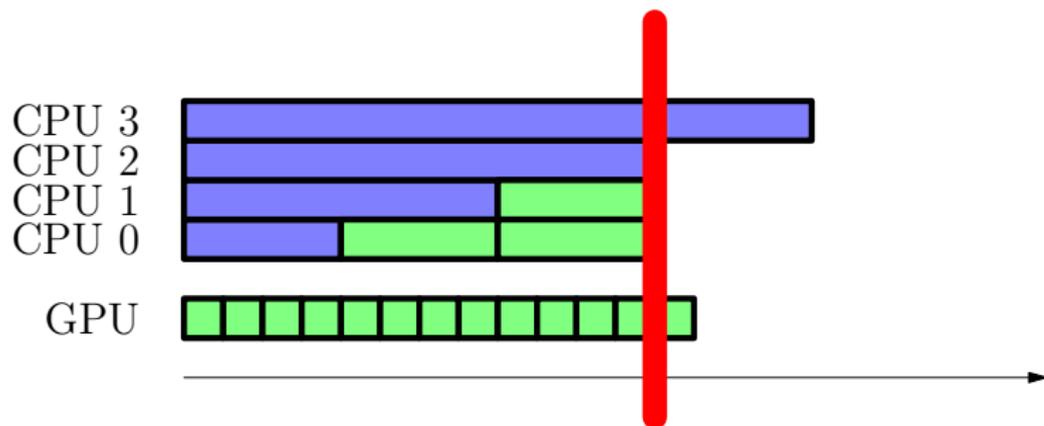
for $i = 0..m - 1$

	\mathcal{A}_{i1}	\mathcal{B}_{i1}	...	\mathcal{B}_{im}
CPU	$1 - i/m$	$1 - i/m$...	$1 - i/m$
GPU	$1 - i/m$	$1/m^2$...	$1/m^2$

If λ larger than C_{max}^*



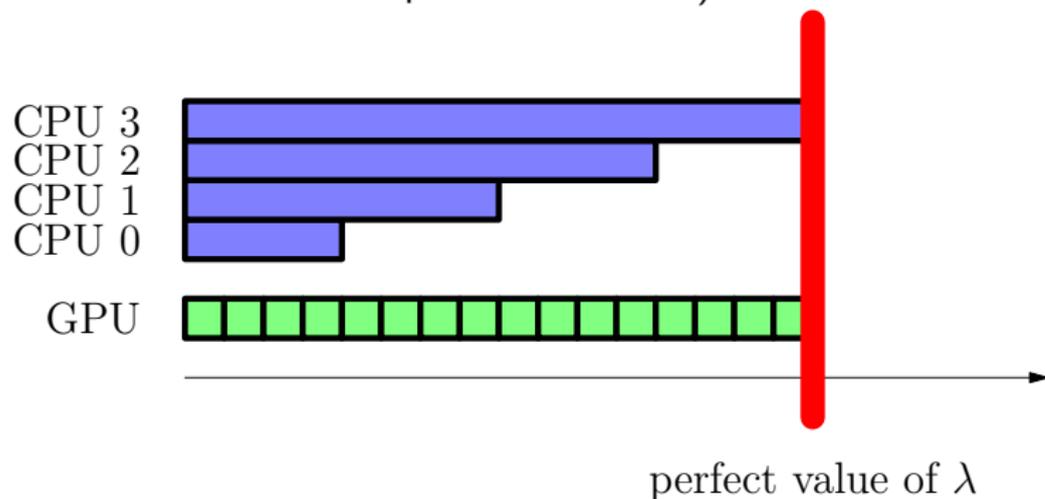
If λ smaller than C_{max}^*



λ a bit too small (too much work)

λ is at the right value

Gdual builds here the optimal solution :-)



Going further...

It is possible to improve the ratio 2.

- Refine the dual approximation technique.
- At each step of the dual approximation, solve a dynamic programming algorithm.
- Case $k = 1$: performance ratio of $g = \frac{4}{3}$ in time $O(n^2 m^2)$.
- Case $k \geq 2$ ratio $g = \frac{4}{3} + \frac{1}{3k}$ in time $O(n^2 m^2 k^3)$.

See [Monna, Kedad, Mounie and Trystram'2013] for more details.

Going further...

It is possible to improve the ratio 2.

- Refine the dual approximation technique.
- At each step of the dual approximation, solve a dynamic programming algorithm.
- Case $k = 1$: performance ratio of $g = \frac{4}{3}$ in time $O(n^2 m^2)$.
- Case $k \geq 2$ ratio $g = \frac{4}{3} + \frac{1}{3k}$ in time $O(n^2 m^2 k^3)$.

See [Monna, Kedad, Mounie and Trystram'2013] for more details.

Going further...

It is possible to improve the ratio 2.

- Refine the dual approximation technique.
- At each step of the dual approximation, solve a dynamic programming algorithm.
- Case $k = 1$: performance ratio of $g = \frac{4}{3}$ in time $O(n^2 m^2)$.
- Case $k \geq 2$ ratio $g = \frac{4}{3} + \frac{1}{3k}$ in time $O(n^2 m^2 k^3)$.

See [Monna, Kedad, Mounie and Trystram'2013] for more details.

Actual systems are much harder...

The previous dual approximation algorithm was designed and analyzed for the simplest possible problem.
It has been implemented using WS as a framework (Kaapi runtime).

It has a low complexity and a small performance guarantee.
However, it is almost impossible to extend to any more realistic features.

- precedence relations
- communication
- memory constraints
- ...

Moreover, important practical aspects of hybrid scheduling are missing and they should be taken into account.

Actual systems are much harder...

The previous dual approximation algorithm was designed and analyzed for the simplest possible problem.
It has been implemented using WS as a framework (Kaapi runtime).

It has a low complexity and a small performance guarantee.
However, it is almost impossible to extend to any more realistic features.

- precedence relations
- communication
- memory constraints
- ...

Moreover, important practical aspects of hybrid scheduling are missing and they should be taken into account.

Take home message

- Do not consider too sophisticated scheduling policies.
- WS allows to face automatically many of the on-line problems (uncertainties, congestion, ...).
- Dual approximation is very powerful, but it requires some a priori knowledge.
- It will be interesting to reconsider alternative models (like malleable tasks).

Thanks to my many close collaborators, especially Daniel Cordeiro, Gregory Mounie and Krzysztof Rządca.

Take home message

- Do not consider too sophisticated scheduling policies.
- WS allows to face automatically many of the on-line problems (uncertainties, congestion, ...).
- Dual approximation is very powerful, but it requires some a priori knowledge.
- It will be interesting to reconsider alternative models (like malleable tasks).

Thanks to my many close collaborators, especially Daniel Cordeiro, Gregory Mounie and Krzysztof Rządca.