Scheduling on Multi-Cores with GPU

Safia Kedad-Sidhoum[1], Florence Monna[1], Grégory Mounié[2], Denis Trystram[2,3]

[1]Laboratoire d'Informatique de Paris 6, 4 Place Jussieu,75005 Paris
[2]Grenoble Institute of Technology, 51 avenue Kuntzmann,38330 Montbonnot Saint Martin, France
[3]Institut Universitaire de France

August 26, 2013

Outline

Scheduling with GPU

Most computers today include

- a Multi-core CPU
- a high performance parallel computing accelerators : the GPGPU (General Purpose Graphical Processing Unit).

Examples:

- Laptop/Tablet/Smartphone (Intel Core i7, Nvidia Tegra 4)
- Game console (PS4, Xbox One)
- Titan (in the top of the Top500 of the supercomputers)

In each machine, there are vectorial coprocessors with very high computing throughput, an interesting asset for High Performance Computing (HPC).

GPU programming example

Vector addition element by element

Compute $Y = alpha + X$, Y and X being two vectors of 1024 float.

```
1  prog = create_program([<<EOF
2  __kernel void addition( float alpha,
3                          __global const float *x,
4                          __global float *y) {
5    size_t ig = get_global_id(0);
6    y[ig] = alpha + x[ig];
7  }
8  EOF
9  ])
10 create_kernel("addition",prog)
```

```
1 input = OpenCL::VArray::new(FLOAT, 1024)
2 output = OpenCL::VArray::new(FLOAT, 1024)
3 input_gpu = create_buffer(1024*4)
4 output_gpu = create_buffer(1024*4)
```

GPU programming example

Vector addition element by element

Compute $Y = alpha + X$, Y and X being two vectors of 1024 float.

```
1  prog = create_program([<<EOF
2  __kernel void addition( float alpha,
3                          __global const float *x,
4                          __global float *y) {
5    size_t ig = get_global_id(0);
6    y[ig] = alpha + x[ig];
7  }
8  EOF
9  ])
10 create_kernel("addition",prog)
```

```
1 input = OpenCL::VArray::new(FLOAT, 1024)
2 output = OpenCL::VArray::new(FLOAT, 1024)
3 input_gpu = create_buffer(1024*4)
4 output_gpu = create_buffer(1024*4)
```

GPU programming example

Sequence of commands

line 1 copy input buffer from CPU memory to GPU memory

line 2-4 compute the kernel with the arguments and vector of size 1024 float split in 64 bloc size

line 5 copy output buffer from GPU memory to CPU memory

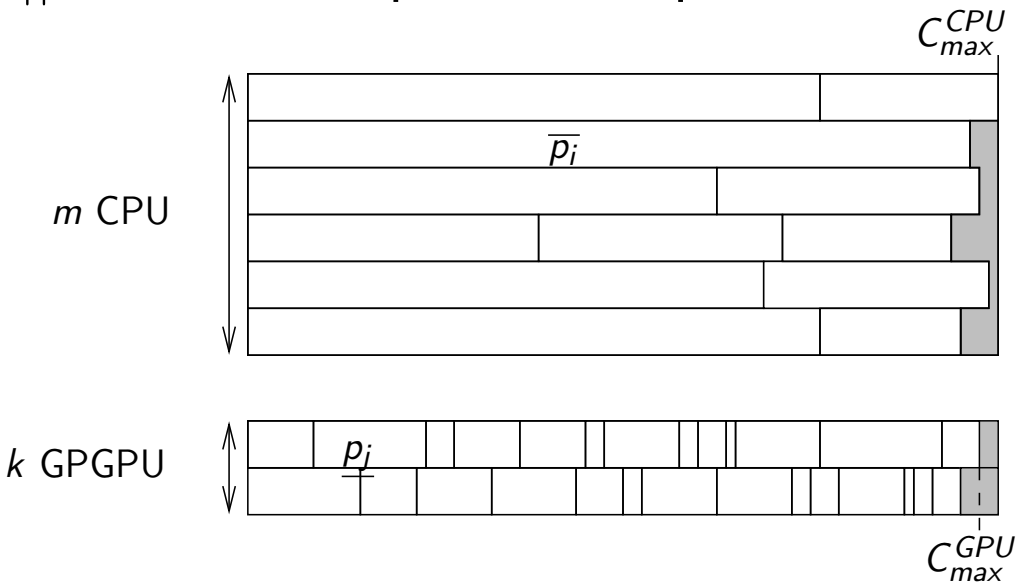
```
1  enqueue_write_buffer(1024*4, input, input_gpu)
2  args= set_args([OpenCL::Float::new(5.0),
3                  input_gpu, output_gpu])
4  enqueue_NDrange_kernel(prog, args, [1024], [64])
5  enqueue_read_buffer(1024*4, output_gpu, output)
```

Contribution

- The tasks assigned to the GPUs must be carefully chosen.
- Generic method to do the assignment for High Performance Computing Systems.
- No previous model:
  start with a simplified problem, without communication issues, precedence relations...

Description of the Problem - Complexity

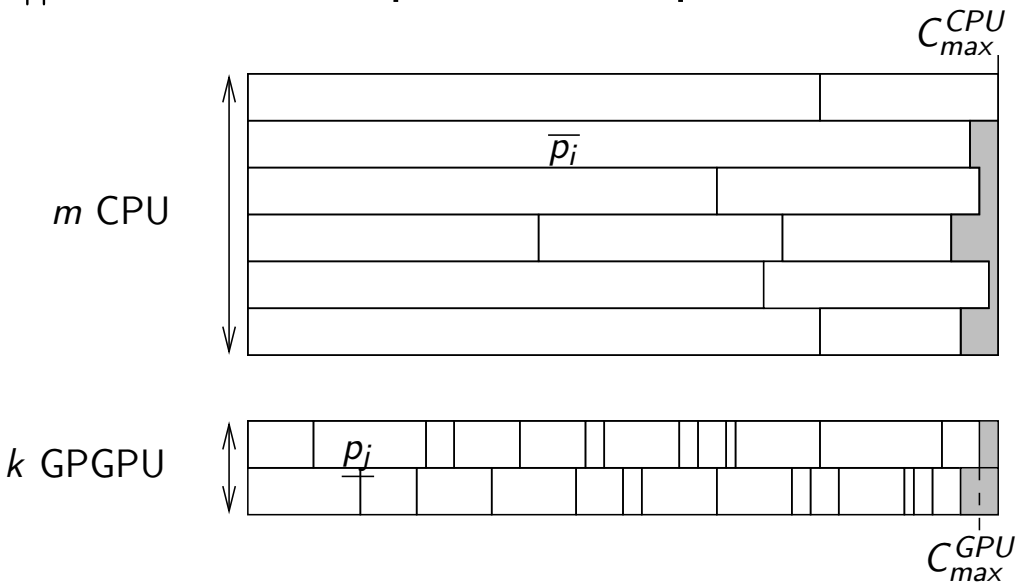
- $(Pm, Pk) \mid\mid C_{max}$: n independent sequential tasks $T_1, \ldots, T_n$.



- Objective: minimize the makespan of the schedule.
- If $p_j = \overline{p_j}$ for all tasks $(Pm, P1) \mid\mid C_{max} \Leftrightarrow P \mid\mid C_{max}$, NP-hard
  $\Longrightarrow$ Problem of scheduling with GPUs also NP-hard

Description of the Problem - Complexity

- $(Pm, Pk) \mid\mid C_{max}$: n independent sequential tasks $T_1, \ldots, T_n$.

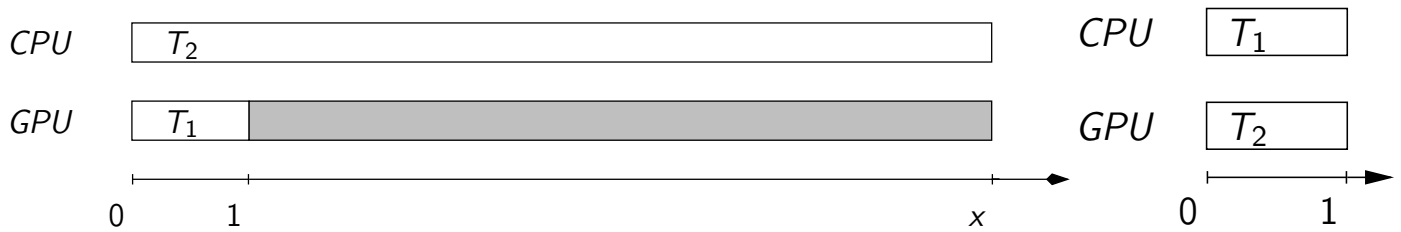


- Objective: minimize the makespan of the schedule.
- If $p_j = \overline{p_j}$ for all tasks $(Pm, P1) \mid\mid C_{max} \Leftrightarrow P \mid\mid C_{max}$, NP-hard $\Longrightarrow$ Problem of scheduling with GPUs also NP-hard

List based sheduling

Lemma

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



Dual approximation technique

- Use of the dual approximation technique [Hochbaum & Shmoys, 1988]: for a ratio g, take a guess λ and either delivers a schedule of makespan at most $g\lambda$, or answers that there exists no schedule of length at most λ.
- At each step of the dual approximation, dynamic programming algorithm.
- Case $k = 1$: performance ratio of $g = \frac{4}{3}$ in time $O\left(n^2 m^2\right)$.
- Case $k \geq 2$ ratio $g = \frac{4}{3} + \frac{1}{3k}$ in time $O\left(n^2 m^2 k^3\right)$.

Dual approximation technique

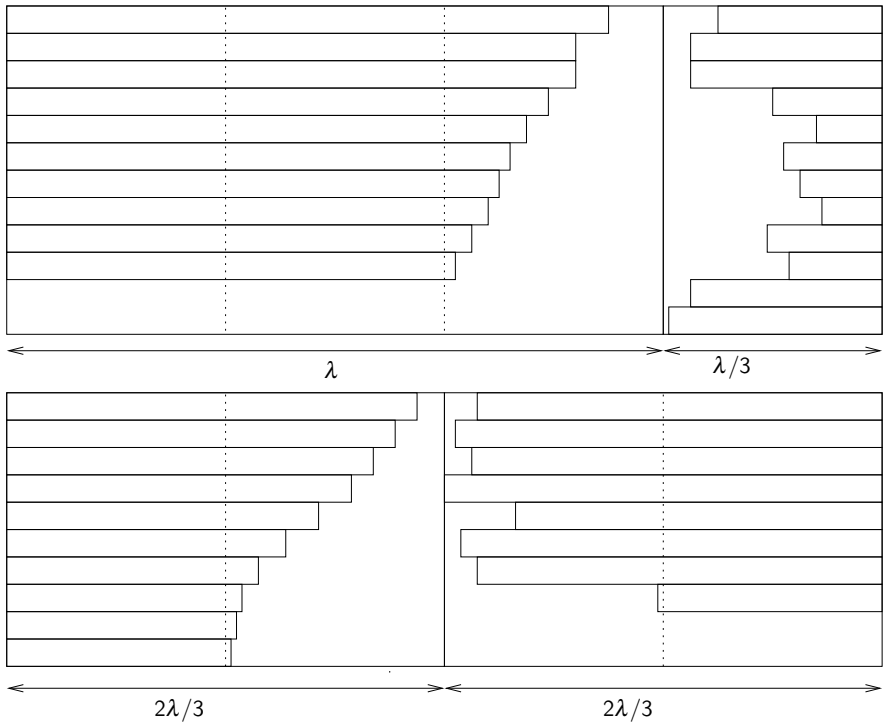
- Use of the dual approximation technique [Hochbaum & Shmoys, 1988]: for a ratio g, take a guess λ and either delivers a schedule of makespan at most $g\lambda$, or answers that there exists no schedule of length at most λ.
- At each step of the dual approximation, dynamic programming algorithm.
- Case $k = 1$: performance ratio of $g = \frac{4}{3}$ in time $O\left(n^2 m^2\right)$.
- Case $k \geq 2$ ratio $g = \frac{4}{3} + \frac{1}{3k}$ in time $O\left(n^2 m^2 k^3\right)$.

Dual approximation technique

- Use of the dual approximation technique [Hochbaum & Shmoys, 1988]: for a ratio g, take a guess λ and either delivers a schedule of makespan at most $g\lambda$, or answers that there exists no schedule of length at most λ.
- At each step of the dual approximation, dynamic programming algorithm.
- Case $k = 1$: performance ratio of $g = \frac{4}{3}$ in time $O\left(n^2 m^2\right)$.
- Case $k \geq 2$ ratio $g = \frac{4}{3} + \frac{1}{3k}$ in time $O\left(n^2 m^2 k^3\right)$.

The Shelves' Idea

For $k = 1$, assuming a schedule of length lower than λ exists.
The idea: to partition the set of tasks on the CPUs into two sets, each consisting in two shelves.



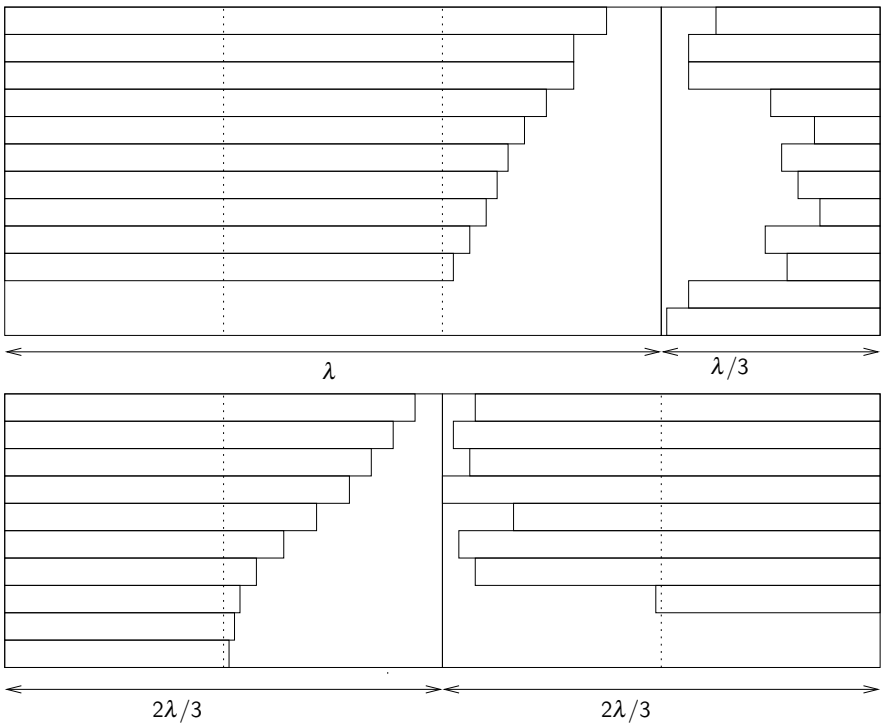
- a first set with a shelf of length λ the other of length $\frac{\lambda}{3}$,
- a second set with two shelves of length $\frac{2\lambda}{3}$.

The Shelves' Idea

- Partition ensures that the makespan on the GPUs is lower than $\frac{4\lambda}{3}$.
- The tasks are independent: the scheduling is straightforward when the assignment of the tasks has been determined.
- The main problem is to assign the tasks in each shelf on the CPUs or on the GPUs in order to obtain a feasible solution.

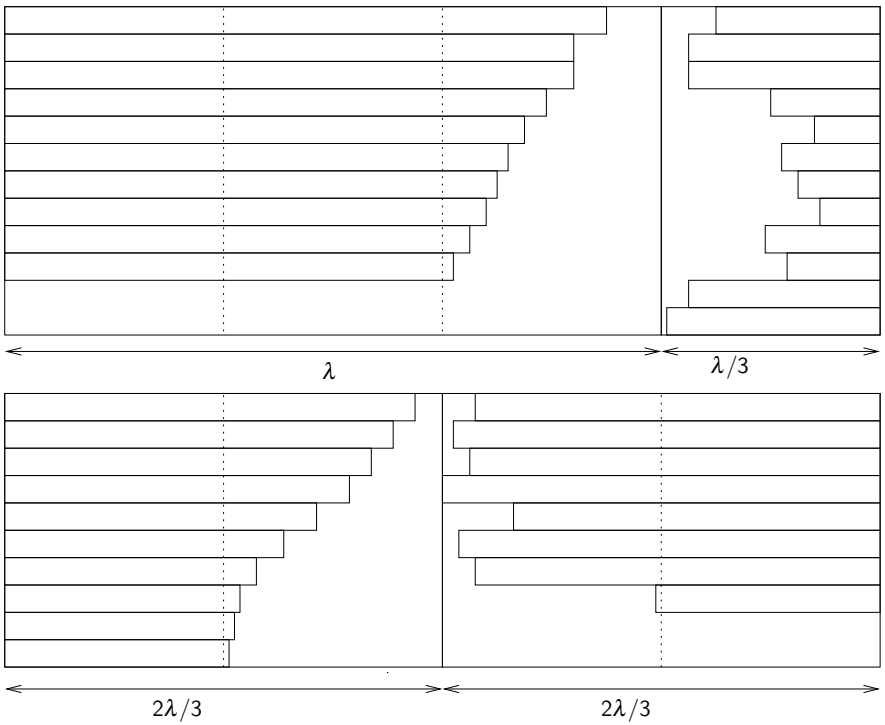
Structure of an Optimal Schedule for $k = 1$

If there exists a schedule of length at most λ



Structure of an Optimal Schedule for $k = 1$

If there exists a schedule of length at most λ

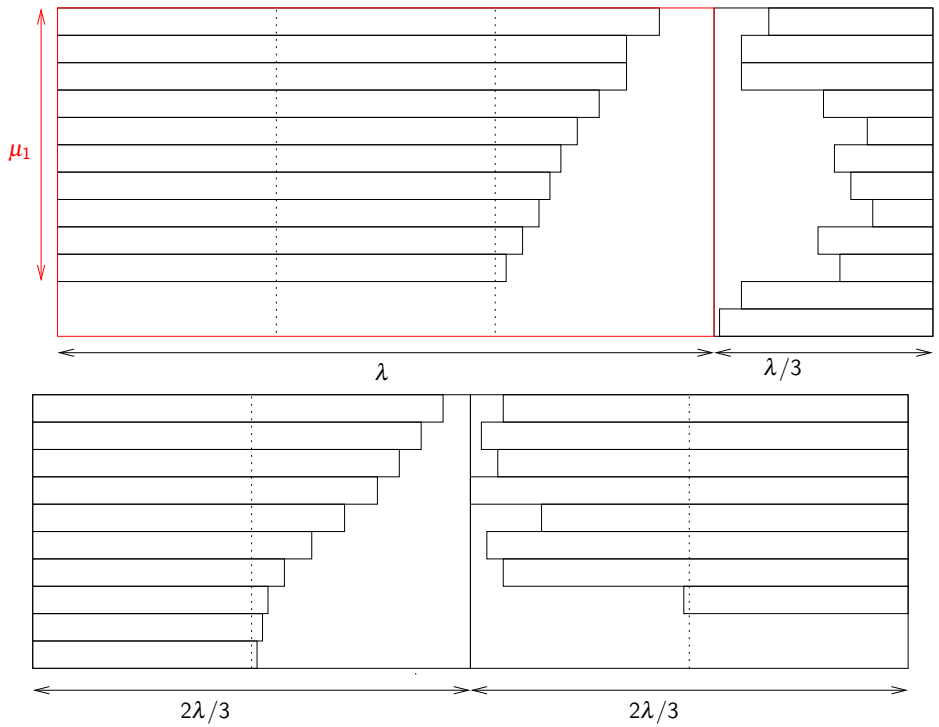


Property (1)

For each task T_j, $\overline{p_j} \leqslant \lambda$, *and* $\sum_{\pi(j) \in \mathscr{C}} \overline{p_j} \leqslant m\lambda$.

Structure of an Optimal Schedule for $k = 1$

If there exists a schedule of length at most λ

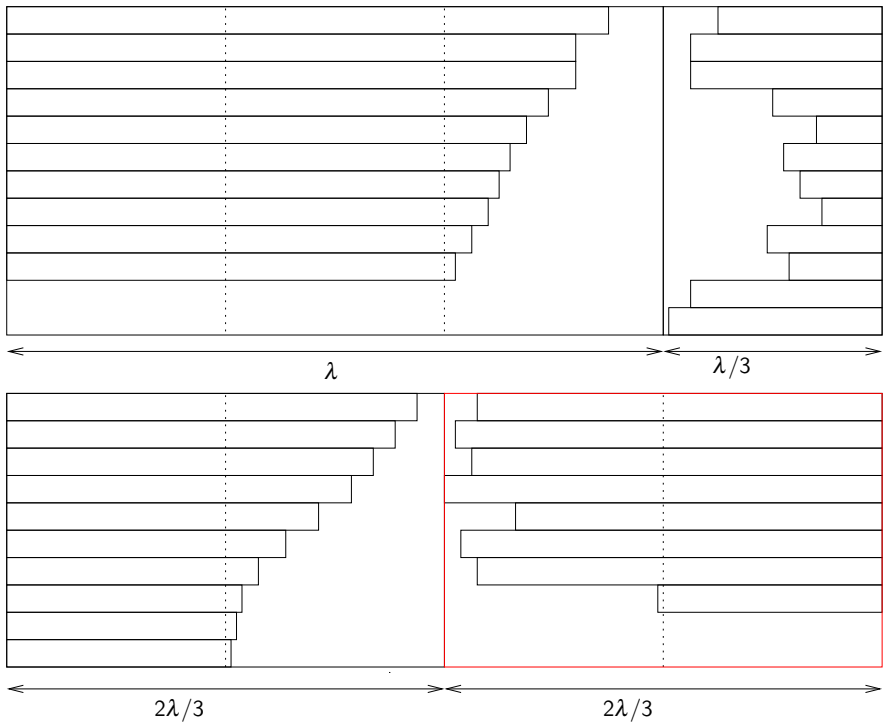


Property (2)

T_i, T_j two successive tasks on a CPU. If $\overline{p_i} > \frac{2\lambda}{3}$, then $\overline{p_j} \leqslant \frac{\lambda}{3}$.

Structure of an Optimal Schedule for $k = 1$

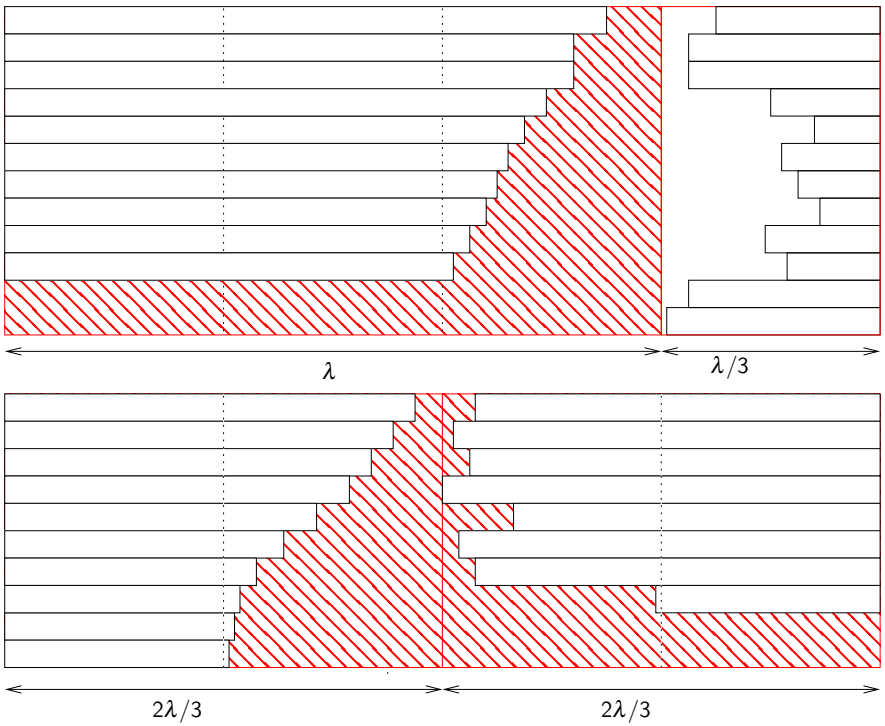
If there exists a schedule of length at most λ



Property (3)

Two tasks T_i, T_j with $\frac{\lambda}{3} < \overline{p_l} \leqslant \frac{2\lambda}{3}$ $(l = i, j)$ can be executed successively on the same CPU within a time $\frac{4\lambda}{3}$.

Structure of an Optimal Schedule for $k = 1$



The remaining tasks (with a processing time lower than $\frac{\lambda}{2q}$ (resp. $\frac{\lambda}{2q+1}$)) fit in the remaining space in front of S_1 and between all the others shelves, otherwise the schedule would not satisfy Property 1.

Partitioning the Tasks into Shelves

We solve the assignment problem with a dynamic programming summing up the previous constraints. Here, we take $g = \frac{4}{3}$.

For task T_j, binary variable : $x_j = \begin{cases} 1 & \text{if assigned to a CPU} \\ 0 & \text{if assigned to the GPU} \end{cases}$

$$W_C^* = \min \sum_{j=1}^{n} \overline{p_j} x_j \tag{1}$$

$$\text{s.t. } \frac{1}{2} \sum_{2\lambda/3 \geqslant \overline{p_j} > \lambda/3} x_j + \sum_{\overline{p_j} > 2\lambda/3} x_j \leqslant m \tag{2}$$

$$\sum_{j=1}^{n} \underline{p_j} (1 - x_j) \leqslant \frac{4\lambda}{3} \tag{3}$$

$$x_j \in \{0, 1\} \tag{4}$$

Partitioning the Tasks into Shelves

- Dynamic programming algorithm: solves the previous problem in $\mathcal{O}\left(n^2 m^2\right)$.
- Reduction of the states on the GPU to a smaller number.
  Number of time intervals of length $\frac{\lambda}{3n}$ for a task T_j executed on the GPU, $\nu_j = \left\lfloor \frac{p_j}{\lambda/(3n)} \right\rfloor$.
  $N = \sum_{\pi(j) \in \mathcal{G}} \nu_j$ total number of these intervals on the GPU.
- Error on processing time of each task $\varepsilon_j = \underline{p_j} - \nu_j \leqslant \frac{\lambda}{3n}$
  If all the tasks are assigned to the GPU, error at most $n\frac{\lambda}{3n} = \frac{\lambda}{3}$.
- Constraint (3) becomes $N = \sum_{\pi(j) \in \mathcal{G}} \nu_j \leqslant 3n$

Binary Search - Cost Analysis

- If optimum $W_C^* = \min_{0 \leqslant \mu \leqslant m, 0 \leqslant \mu' \leqslant 2(m-\mu), 0 \leqslant N \leqslant 2n} W_c(n, \mu, \mu', N) > m\lambda$, no solution with makespan $\leqslant \lambda$ exists, algorithm answers "NO"
- Otherwise, construct feasible solution with makespan $\leqslant \frac{4\lambda}{3}$, with shelves on the CPUs and μ^*, μ'^* and N^* values.
- One step of the dual-approximation algorithm, with a fixed guess. Binary search in $\log(B_{max} - B_{min})$.
- At each step, we have $1 \leqslant j \leqslant n$, $1 \leqslant \mu \leqslant m$, $1 \leqslant \mu' \leqslant 2(m-\mu)$, and $0 \leqslant N \leqslant 3n$ so the time complexity is in $O\left(n^2 m^2\right)$.

Extension

Algorithm can be extended to $(Pm, Pk) \,||\, C_{max}$ with $k \geq 2$.

$$W_C^* = \min \sum_{j=1}^{n} \overline{p_j} x_j \tag{5}$$

$$\text{s.t.} \quad \frac{1}{2} \sum_{2\lambda/3 \geqslant \overline{p_j} > \lambda/3} x_j + \sum_{\overline{p_j} > 2\lambda/3} x_j \leqslant m \tag{6}$$

$$\frac{1}{2} \sum_{2\lambda/3 \geqslant \underline{p_j} > \lambda/3} x_j + \sum_{\underline{p_j} > 2\lambda/3} x_j \leqslant k \tag{7}$$

$$N = \sum_{\pi(j) \in \mathcal{G}} v_j \leqslant 3kn \tag{8}$$

$$x_j \in \{0, 1\} \tag{9}$$

Extension

- The approximation algorithm can be extended to the problem with $k \geq 2$ GPUs with a performance guarantee of $\frac{4}{3} + \frac{1}{3k}$.
- To solve each step of the binary search, $O\left(n^2 k^3 m^2\right)$ states are considered, since $1 \leqslant j \leqslant n$, $1 \leqslant \mu \leqslant m$, $1 \leqslant \mu' \leqslant 2(m - \mu)$, $1 \leqslant \kappa \leqslant k$, $1 \leqslant \kappa' \leqslant 2(k - \kappa)$, and $0 \leqslant N \leqslant 3kn$.
  $\Longrightarrow$ Time complexity in $O\left(n^2 k^3 m^2\right)$ for each step of the binary search.

Experimental Analysis

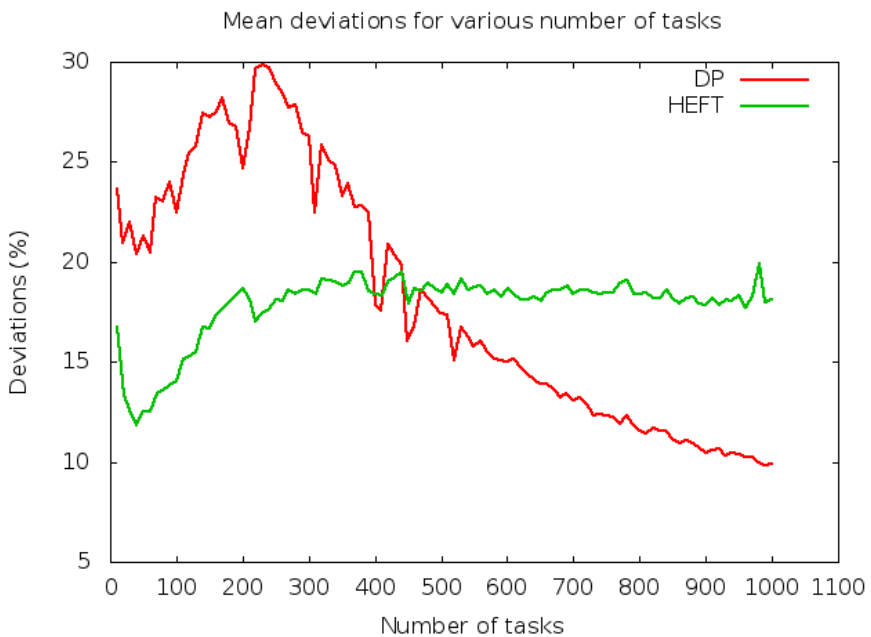
Comparison of a relaxed version of the dynamic programming (DP) algorithm with a ratio of 2 to the HEFT algorithm (Heterogeneous-Earliest-Finish-Time) [Topcuoglu et al. 2002]: prioritizing with decreasing average execution time, heterogeneous earliest finish time rule.

Lemma

For the $(Pm, P1)$-problem, the worst case performance ratio of HEFT is larger than $m/2$.

Series of experiments

- Random instances: 10, 20, 40, 80 tasks, 1, 2, 4, 8, 16, 32, 64 CPUs, 1,2, 4, 8 GPUs.
- A task is assigned an acceleration factor of 1/15 or 1/35 with a probability of 1/2.



Conclusion and Perpectives

- Contribution: fast algorithms with constant garantee for independent tasks on CPUs and GPUs. In the case of a single (resp. multiple) GPU(s) a ratio of $\frac{4}{3} + \varepsilon$ (resp. $\frac{4}{3} + \frac{1}{3k} + \varepsilon$) is achieved, and can be degradated to a ratio of 2 for efficiency.
- Finer ratios can be obtained with a sacrifice in time complexity.
- Extensions to partial preemption and malleable tasks can be done.
- On-going research on the problem with precedence relations, and data communication.
- Protocols in writing for an integration into parallel programming environment like StarPU and xKaapi.