PERFORMANCE ANALYSIS OF PARALLEL CODES ON HETEROGENEOUS SYSTEMS

E. Agullo, B. Bramas, A. Buttari, A. Guermouche, F. Lopez, S. Nakov, S. Thibault

SOLHAR plenary meeting, Bordeaux 25-01-2026

A MOTIVATING EXAMPLE

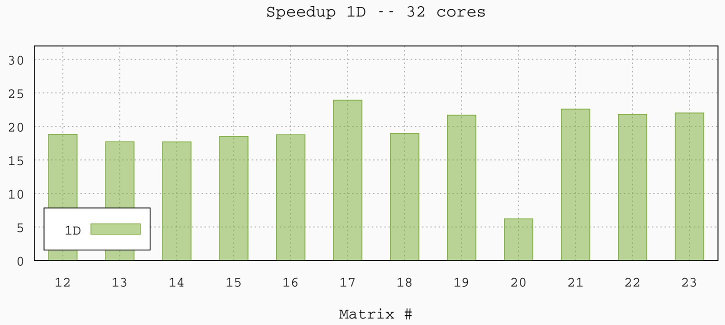
Plain speedup is not enough

- qr_mumps +StarPU with 1D, block-column partitioning
- Matrices from UF collection

#	Matrix	Mflops	#	Matrix	Mflops
12	hirlam	1384160	18	spal_004	30335566
13	flower_8_4	2851508	19	n4c6-b6	62245957
14	Rucci1	5671282	20	sls	65607341
15	ch8-8-b3	10709211	21	TF18	194472820
16	GL7d24	16467844	22	lp_nug30	221644546
17	neos2	20170318	23	mk13-b5	259751609

- One node of the ADA supercomputer (IBM x3750-M4, Intel Sandy Bridge E5-4650 @ 2.7 GHz, 4×8 cores)

Experimental results: speedups



Speedup says something, e.g., performance is poor on small matrices and good on bigger ones.

Speedup doesn't say anything on the reason.

Is there a problem in the implementation, in the algorithm, in the data? what's that crappy matrix?

PERFORMANCE ANALYSIS APPROACH, THE HOMOGENEOUS CASE

Area performance upper bound

Parallel efficiency

The parallel efficiency is defined as

$$e(p) = \frac{\tilde{t}(1)}{t(p) \cdot p}$$

- $\tilde{t}(1)$ is the execution time of the best sequential algorithm on one core;
- $t(p)$ is the execution time of the best parallel algorithm on p cores.

Note that, in general, $t(1) \geq \tilde{t}(1)$ because:

- parallelism requires partitioning of data and operations which reduces the efficiency of tasks;
- the parallel algorithm may trade some extra flops for concurrency.

A finer performance analysis

The execution time $t(p)$ can be decomposed in the following three terms:

- $t_t(p)$: the time spent executing tasks.
- $t_r(p)$: the overhead of the runtime system. $t_r(1) := 0$.
- $t_i(p)$: idle time. $t_i(1) := 0$.

The overall efficiency can thus be written as:

$$
\begin{aligned}
e(p) &= \frac{\tilde{t}_t(1)}{t_t(p) + t_r(p) + t_i(p)} \\
&= \overbrace{\frac{\tilde{t}_t(1)}{t_t(1)}}^{e_g} \cdot \overbrace{\frac{t_t(1)}{t_t(p)}}^{e_t} \cdot \overbrace{\frac{t_t(p)}{t_t(p) + t_r(p)}}^{e_r} \cdot \overbrace{\frac{t_t(p) + t_r(p) + t_c(p)}{t_t(p) + t_r(p) + t_c(p) + t_i(p)}}^{e_p}.
\end{aligned}
$$

with:

A finer performance analysis

The execution time $t(p)$ can be decomposed in the following three terms:

- $t_t(p)$: the time spent executing tasks.
- $t_r(p)$: the overhead of the runtime system. $t_r(1) := 0$.
- $t_i(p)$: idle time. $t_i(1) := 0$.

The overall efficiency can thus be written as:

$$
\begin{aligned}
e(p) &= \frac{\tilde{t}_t(1)}{t_t(p) + t_r(p) + t_i(p)} \\
&= \overbrace{\frac{\tilde{t}_t(1)}{t_t(1)}}^{e_g} \cdot \overbrace{\frac{t_t(1)}{t_t(p)}}^{e_t} \cdot \overbrace{\frac{t_t(p)}{t_t(p) + t_r(p)}}^{e_r} \cdot \overbrace{\frac{t_t(p) + t_r(p) + t_c(p)}{t_t(p) + t_r(p) + t_c(p) + t_i(p)}}^{e_p}.
\end{aligned}
$$

with:

e_g: the granularity efficiency. Measures the impact exploiting of parallel algorithms compared to sequential ones.

A finer performance analysis

The execution time $t(p)$ can be decomposed in the following three terms:

- $t_t(p)$: the time spent executing tasks.
- $t_r(p)$: the overhead of the runtime system. $t_r(1) := 0$.
- $t_i(p)$: idle time. $t_i(1) := 0$.

The overall efficiency can thus be written as:

$$
\begin{aligned}
e(p) &= \frac{\tilde{t}_t(1)}{t_t(p) + t_r(p) + t_i(p)} \\
&= \overbrace{\frac{\tilde{t}_t(1)}{t_t(1)}}^{e_g} \cdot \overbrace{\frac{t_t(1)}{t_t(p)}}^{e_t} \cdot \overbrace{\frac{t_t(p)}{t_t(p) + t_r(p)}}^{e_r} \cdot \overbrace{\frac{t_t(p) + t_r(p) + t_c(p)}{t_t(p) + t_r(p) + t_c(p) + t_i(p)}}^{e_p}.
\end{aligned}
$$

with:

e_t: the task efficiency. Measures the exploitation of data locality.

A finer performance analysis

The execution time $t(p)$ can be decomposed in the following three terms:

- $t_t(p)$: the time spent executing tasks.
- $t_r(p)$: the overhead of the runtime system. $t_r(1) := 0$.
- $t_i(p)$: idle time. $t_i(1) := 0$.

The overall efficiency can thus be written as:

$$
\begin{aligned}
e(p) &= \frac{\tilde{t}_t(1)}{t_t(p) + t_r(p) + t_i(p)} \\
&= \overbrace{\frac{\tilde{t}_t(1)}{t_t(1)}}^{e_g} \cdot \overbrace{\frac{t_t(1)}{t_t(p)}}^{e_t} \cdot \overbrace{\frac{t_t(p)}{t_t(p) + t_r(p)}}^{e_r} \cdot \overbrace{\frac{t_t(p) + t_r(p) + t_c(p)}{t_t(p) + t_r(p) + t_c(p) + t_i(p)}}^{e_p}.
\end{aligned}
$$

with:

e_r: the runtime efficiency. Measures how the runtime overhead affects performance.

A finer performance analysis

The execution time $t(p)$ can be decomposed in the following three terms:

- $t_t(p)$: the time spent executing tasks.
- $t_r(p)$: the overhead of the runtime system. $t_r(1) := 0$.
- $t_i(p)$: idle time. $t_i(1) := 0$.

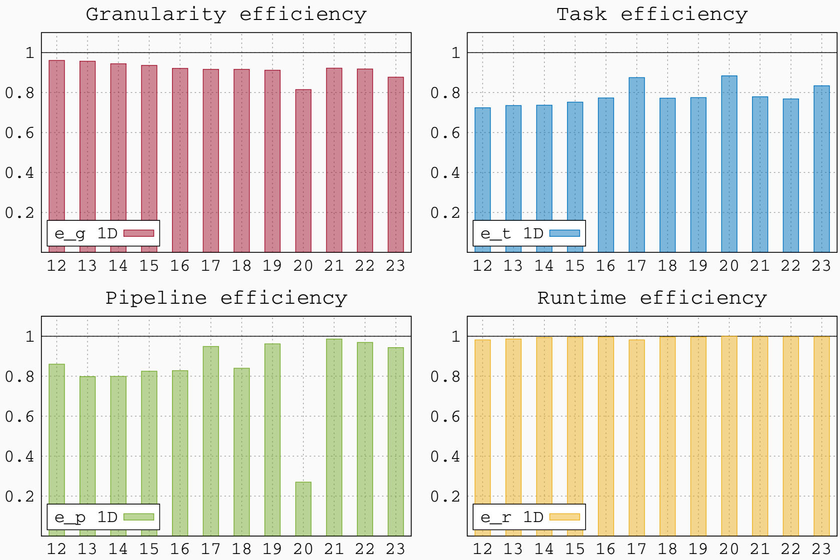
The overall efficiency can thus be written as:

$$e(p) = \frac{\tilde{t}_t(1)}{t_t(p) + t_r(p) + t_i(p)}$$
$$= \overbrace{\frac{\tilde{t}_t(1)}{t_t(1)}}^{e_g} \cdot \overbrace{\frac{t_t(1)}{t_t(p)}}^{e_t} \cdot \overbrace{\frac{t_t(p)}{t_t(p) + t_r(p)}}^{e_r} \cdot \overbrace{\frac{t_t(p) + t_r(p) + t_c(p)}{t_t(p) + t_r(p) + t_c(p) + t_i(p)}}^{e_p}.$$

with:

e_p: the pipeline efficiency. Measures how much concurrency is available and how well it is exploited.

Experimental results: efficiency breakdown



2D partitioning + CA front factorization

1D partitioning is not good for (strongly) overdetermined matrices:

▼ Most fronts are overdetermined

▲ The problem is mitigated by concurrent front factorizations

- 2D block partitioning (not necessarily square)
- Communication avoiding algorithms

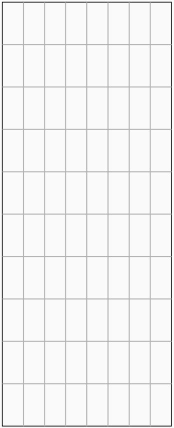
▲ More concurrency

▼ More complex dependencies

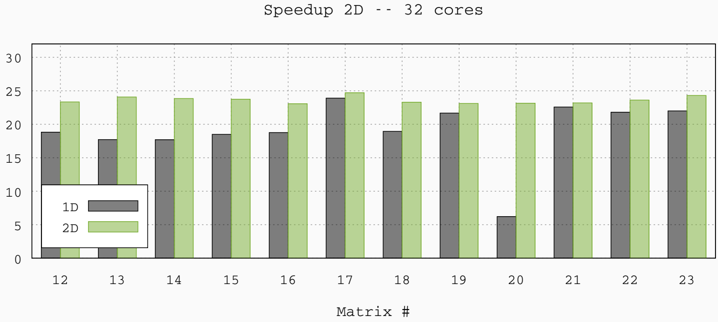
▼ Many more tasks (higher runtime overhead)

▼ Finer task granularity (less kernel efficiency)

Thanks to the simplicity of the STF programming model it is possible to plug in 2D methods for factorizing the frontal matrices with a relatively moderate effort



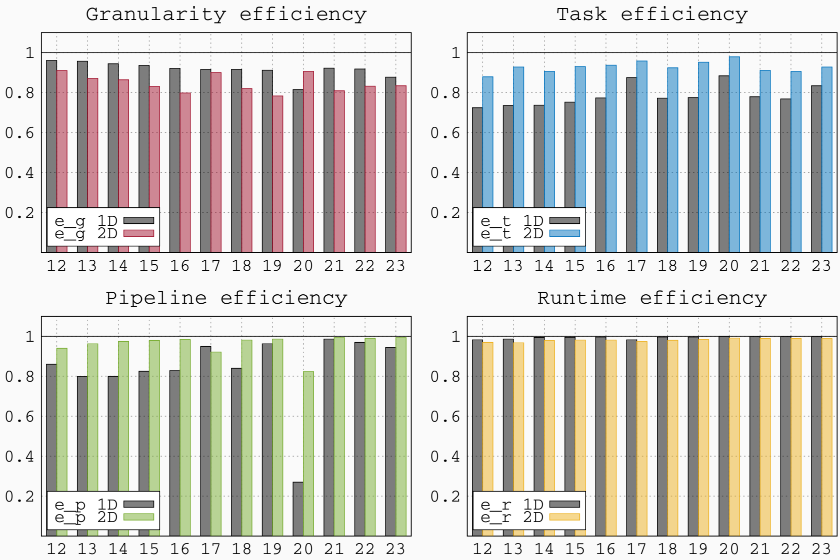
Experimental results: speedups



The scalability of the task-based multifrontal method is enhanced by the the introduction of 2D CA algorithms:

- Speedups are uniform for all tested matrices.
- We perform a comparative performance analysis wrt to the 1D case to show the benefits of the 2D scheme.

Experimental results: efficiency breakdown



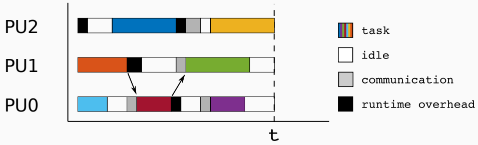
PERFORMANCE ANALYSIS APPROACH, THE HETEROGENEOUS CASE

Area performance upper bound

The parallel efficiency can be defined as

$$e(p) = \frac{t^{min}(p)}{t(p)}$$

where $t^{min}(p)$ is a lower bound on execution time on p resources corresponding to the best schedule under the following assumptions:



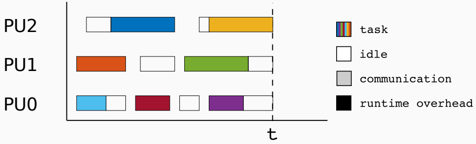
Area performance upper bound

The parallel efficiency can be defined as

$$e(p) = \frac{t^{min}(p)}{t(p)}$$

where $t^{min}(p)$ is a lower bound on execution time on p resources corresponding to the best schedule under the following assumptions:

1. No runtime overhead and no communications.



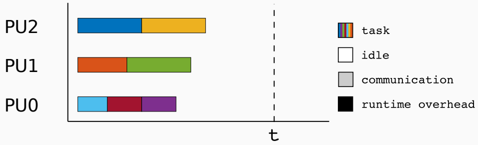
Area performance upper bound

The parallel efficiency can be defined as

$$e(p) = \frac{t^{min}(p)}{t(p)}$$

where $t^{min}(p)$ is a lower bound on execution time on p resources corresponding to the best schedule under the following assumptions:

1. No runtime overhead and no communications.
2. No tasks dependencies.



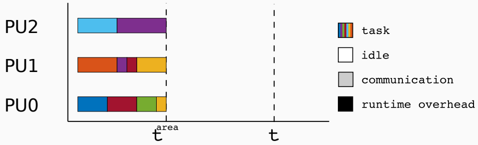
Area performance upper bound

The parallel efficiency can be defined as

$$e(p) = \frac{t^{min}(p)}{t(p)}$$

where $t^{min}(p)$ is a lower bound on execution time on p resources corresponding to the best schedule under the following assumptions:

1. No runtime overhead and no communications.
2. No tasks dependencies.
3. Tasks are moldable.



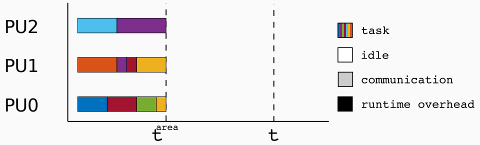
Area performance upper bound

The parallel efficiency can be defined as

$$e(p) = \frac{t^{min}(p)}{t(p)}$$

where $t^{min}(p)$ is a lower bound on execution time on p resources corresponding to the best schedule under the following assumptions:

1. No runtime overhead and no communications.
2. No tasks dependencies.
3. Tasks are moldable.



In the heterogeneous case we have $t^{area}(p)$ is the solution of a linear program.

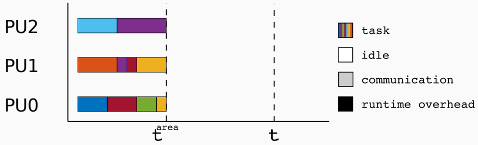
Area performance upper bound

The parallel efficiency can be defined as

$$e(p) = \frac{t^{min}(p)}{t(p)}$$

where $t^{min}(p)$ is a lower bound on execution time on p resources corresponding to the best schedule under the following assumptions:

1. No runtime overhead and no communications.
2. No tasks dependencies.
3. Tasks are moldable.



We consider $t^{area}(p)$ when there is no performance loss resulting from the parallelization: $\tilde{t}^{area}(p)$.

A finer performance analysis

The execution time $t(p)$ can be decomposed in the following four terms:

- $t_t(p)$: the time spent executing tasks.
- $t_r(p)$: the overhead of the runtime system.
- $t_i(p)$: idle time.

The overall efficiency can thus be written as:

$$e(p) = \frac{\tilde{t}^{area}(p)}{t(p)} = \frac{\tilde{t}^{area}(p) \times p}{t_t(p) + t_r(p) + t_c(p) + t_i(p)} = \frac{\tilde{t}_t^{area}(p)}{t_t(p) + t_r(p) + t_c(p) + t_i(p)}$$

$$= \overbrace{\frac{\tilde{t}_t^{area}(p)}{t_t^{area}(p)}}^{e_g} \cdot \overbrace{\frac{t_t^{area}(p)}{t_t(p)}}^{e_t} \cdot \overbrace{\frac{t_t(p)}{t_t(p) + t_r(p)}}^{e_r} \cdot \overbrace{\frac{t_t(p) + t_r(p) + t_c(p)}{t_t(p) + t_r(p) + t_c(p) + t_i(p)}}^{e_p}.$$

with:

A finer performance analysis

The execution time $t(p)$ can be decomposed in the following four terms:

- $t_t(p)$: the time spent executing tasks.
- $t_r(p)$: the overhead of the runtime system.
- $t_i(p)$: idle time.
- $t_c(p)$: the time spent performing communications.

The overall efficiency can thus be written as:

$$e(p) = \frac{\tilde{t}^{area}(p)}{t(p)} = \frac{\tilde{t}^{area}(p) \times p}{t_t(p) + t_r(p) + t_c(p) + t_i(p)} = \frac{\tilde{t}_t^{area}(p)}{t_t(p) + t_r(p) + t_c(p) + t_i(p)}$$

$$= \overbrace{\frac{\tilde{t}_t^{area}(p)}{t_t^{area}(p)}}^{e_g} \cdot \overbrace{\frac{t_t^{area}(p)}{t_t(p)}}^{e_t} \cdot \overbrace{\frac{t_t(p)}{t_t(p) + t_r(p)}}^{e_r} \cdot \overbrace{\frac{t_t(p) + t_r(p)}{t_t(p) + t_r(p) + t_c(p)}}^{e_c} \cdot \overbrace{\frac{t_t(p) + t_r(p) + t_c(p)}{t_t(p) + t_r(p) + t_c(p) + t_i(p)}}^{e_p}.$$

with:

e_c: the communication efficiency. measures the cost of communications with respect to the actual work done due to data transfers between workers.

A finer performance analysis

The execution time $t(p)$ can be decomposed in the following four terms:

- $t_t(p)$: the time spent executing tasks.
- $t_r(p)$: the overhead of the runtime system.
- $t_i(p)$: idle time.
- $t_c(p)$: the time spent performing communications.

The overall efficiency can thus be written as:

$$e(p) = \frac{\tilde{t}^{area}(p)}{t(p)} = \frac{\tilde{t}^{area}(p) \times p}{t_t(p) + t_r(p) + t_c(p) + t_i(p)} = \frac{\tilde{t}_t^{area}(p)}{t_t(p) + t_r(p) + t_c(p) + t_i(p)}$$

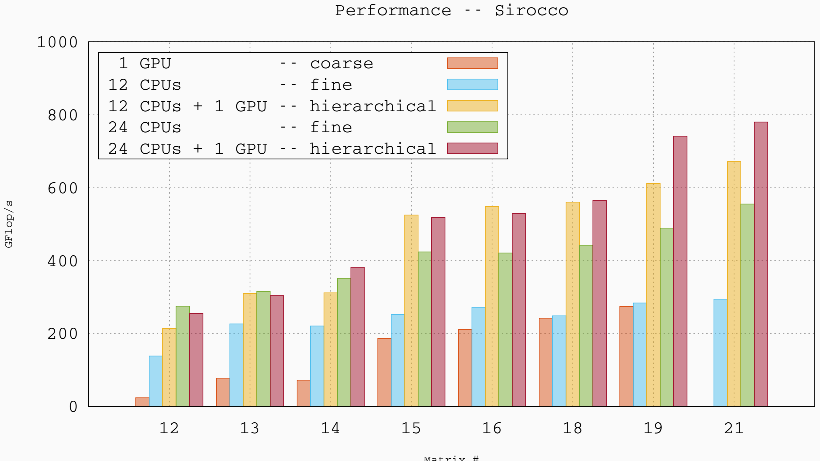
$$= \overbrace{\frac{\tilde{t}_t^{area}(p)}{t_t^{area}(p)}}^{e_g} \cdot \overbrace{\frac{t_t^{area}(p)}{t_t(p)}}^{e_t} \cdot \overbrace{\frac{t_t(p)}{t_t(p) + t_r(p)}}^{e_r} \cdot \overbrace{\frac{t_t(p) + t_r(p)}{t_t(p) + t_r(p) + t_c(p)}}^{e_c} \cdot \overbrace{\frac{t_t(p) + t_r(p) + t_c(p)}{t_t(p) + t_r(p) + t_c(p) + t_i(p)}}^{e_p}.$$

with:

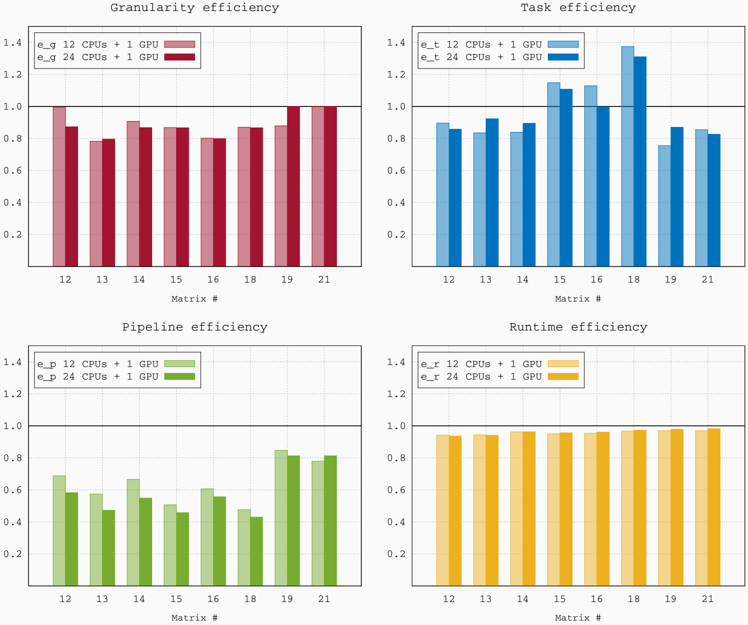
e_t: the task efficiency. Measures how well the assignment of tasks to processing units matches the tasks properties to the units capabilities.

Experimental results: absolute performance

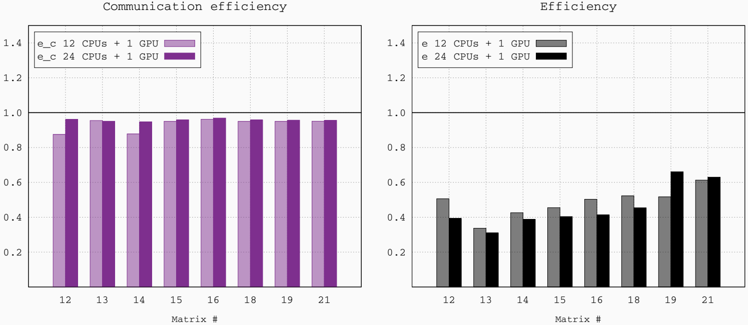
- qr_mumps +StarPU with hierarchical partitioning and HeteroPrio++ scheduler
- One node of the Sirocco computer (Haswell Intel Xeon E5-2680 @ 2.5 GHz, 2×12 cores + Nvidia K40)



Experimental results: efficiency breakdown

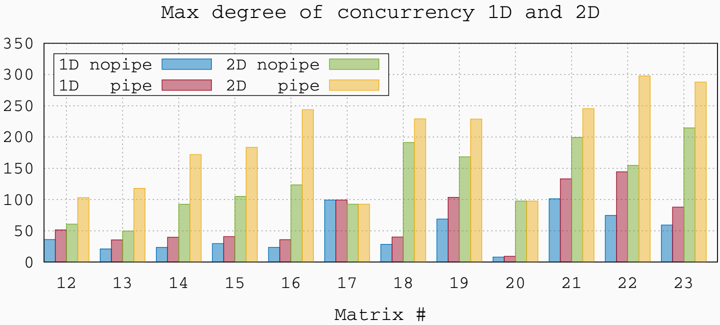


Experimental results: efficiency breakdown



CRITICAL PATH ANALYSIS

Critical path analysis



$$\text{max\_speedup} = \text{avg\_concurrency} = \frac{\sum_{i \in DAG} w_i}{\sum_{i \in CP} w_i}$$

- The DAG used to conduct this analysis is the one related to the case where 32 working threads are used.
- The weight of tasks is chosen to be equal to the execution time measured in an execution with only one working thread.

Thanks!

Questions?