Task-based FMM on multi-cores architectures

Berenger Bramas
HiePACS Team

MCIA Journée Scientifique
February 1st 2013

Outline

- Overview of the FMM
- Parallelization strategies on multi-cores architectures
- Results and performance analysis
- Conclusion and Perspectives

Chebyshev Fast Multipole Method Overview

N-body problems

Example, interactions between planets in a galaxy:

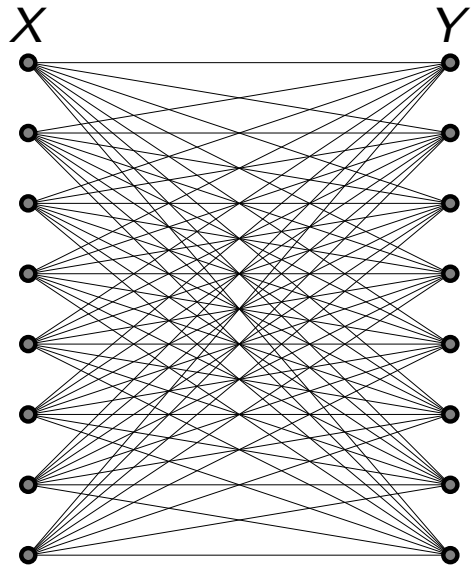
$N(N-1)$ interactions $\rightarrow$ Complexity is $0(n^2)$.

Motivations

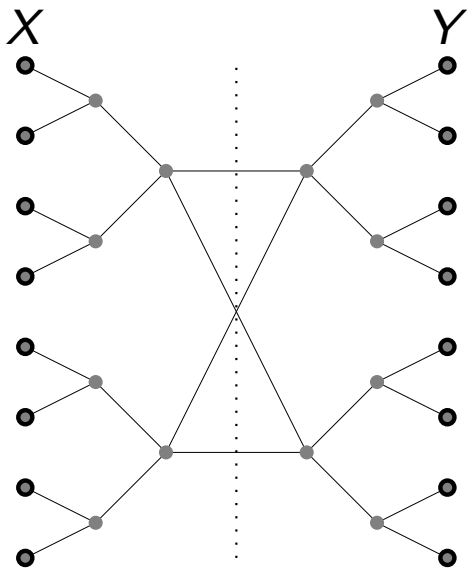
- Pairwise interactions among N bodies.

$$f_i = \sum_{j=1}^{N} K(x_i, y_j)\, w_j \quad \text{for } i = 1, \ldots, M.$$

- Direct computation $0(n^2) \rightarrow$ FMM $O(n)$.



$\mathcal{O}(N^2)$



$\mathcal{O}(N)$

Motivations

- Pairwise interactions among N bodies.

$$f_i = \sum_{j=1}^{N} K(x_i, y_j)\, w_j \quad \text{for } i = 1, \ldots, M.$$

- Direct computation $0(n^2) \rightarrow$ FMM $O(n)$.
- Potentials (Kernel $\approx 1/r$)
  - Electrostatic $\longrightarrow$ Molecular dynamics
  - Gravitational $\longrightarrow$ Astrophysics
  - but also electromagnetism, fluid dynamics, VLSI capacitance, ...

Overview of the FMM

- Potential decomposition

$$f_i = f_i^{near} + f_i^{far}$$

  - Near field is computed by direct interactions.
  - Different approaches to approximate the far field
    - Expansions (Cartesian or Spherical)
    - Interpolation (Chebyshev FMM)

Chebyshev FMM

Idea: Interpolate the kernel on a Chebyshev grid

$$\frac{1}{|x-y|} \sim \sum_{m=1}^{\ell} S_\ell(x,\bar{x}_m) \sum_{n=1}^{\ell} \frac{1}{|\bar{x}_m - \bar{y}_m|} S_\ell(y,\bar{y}_n)$$

where S_ℓ is the interpolation polynomial

$$S_\ell(x,x_m) = \frac{1}{\ell} + \frac{2}{\ell} \sum_{n=1}^{\ell-1} T_n(\bar{x}_m)\, T_n(x)$$

The higher is ℓ the more accurate is the far field

Chebyshev FMM

Idea: Interpolate the kernel on a Chebyshev grid

$$
\begin{aligned}
f_i &= \sum_{j=1}^{N} \frac{1}{|x-y|} w_j \\
&\sim \underbrace{\sum_{m=1}^{\ell} S_\ell(x, \bar{x}_m)}_{\text{L2L}} \underbrace{\sum_{n=1}^{\ell} \frac{1}{|\bar{x}_m - \bar{y}_m|}}_{\text{M2L}} \underbrace{\sum_{j=1}^{N} S_\ell(y, \bar{y}_n)}_{\text{M2M}} w_j
\end{aligned}
$$

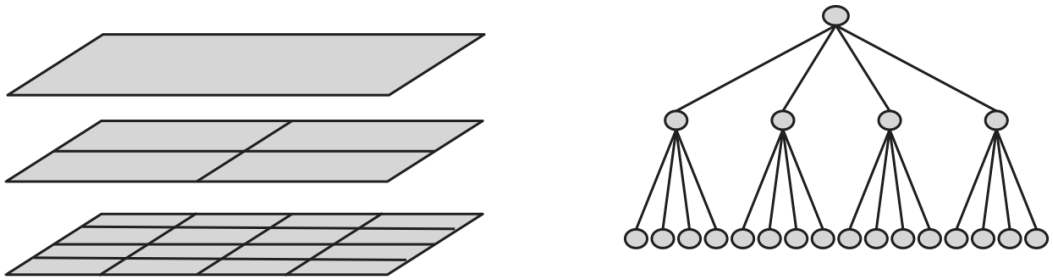
for all $i = 1, \ldots, N$.

Space decomposition

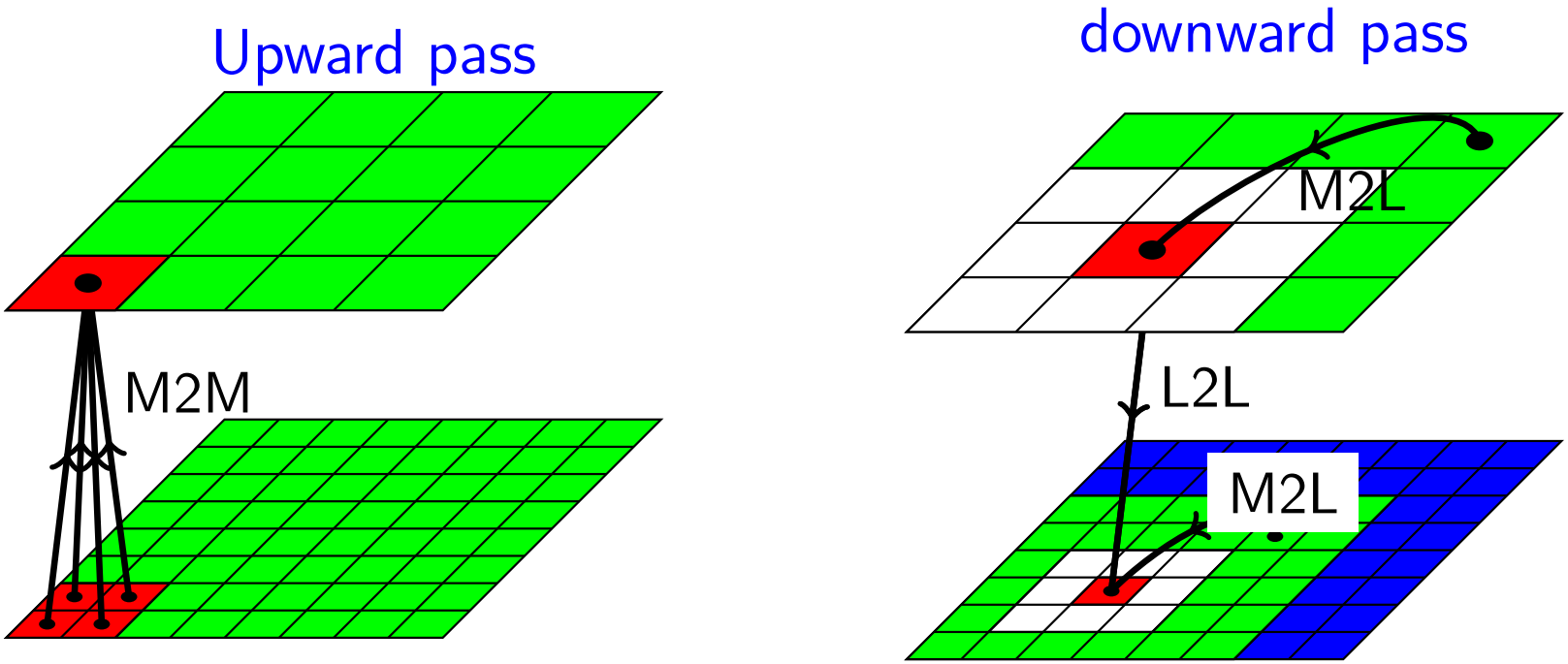
- Potential decomposition

$$f_i = f_i^{near} + f_i^{far}$$

- Hierarchical space decomposition with an octree



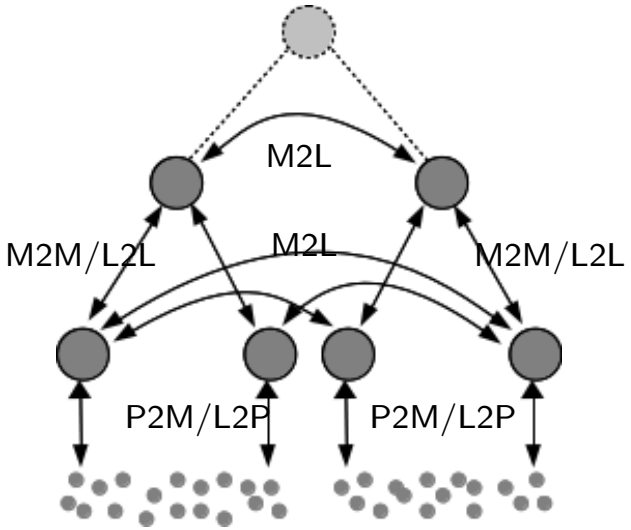
O(N) algorithm



- Upward pass/Downard pass
- Direct interactions (P2P) for the nearest-neighbor leaf cells.

Tree view

- The bodies (particles) are contained in cells
- Cells are subdivided by 2 in each dimension
- h the octree height is choosen to balance near field and far field
- There are $(2^{DIM})^{H-1}$ leaves (8^{H-1} in 3D)



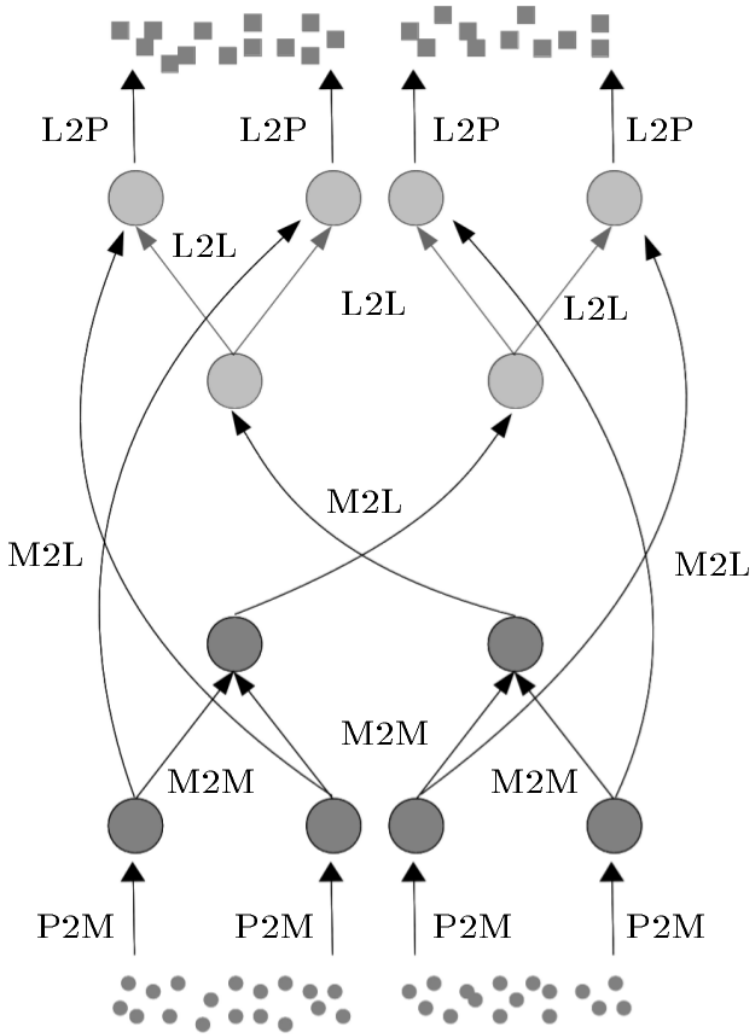
Parallelization strategies of the FMM

From Fork-join to runtime system

The parallelism in the FMM

Dependencies are between:

- Cells
  - parents
  - children
  - neighbors
- Operators



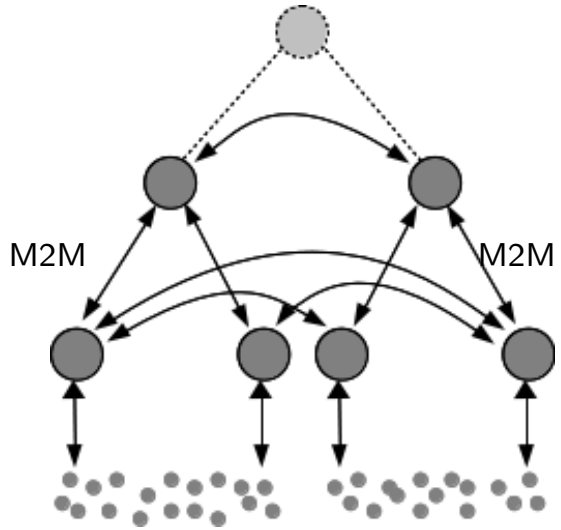
OpenMP-For : Global view of the algorithm

```
function FMM(tree, kernel)
    // Near-field
    P2P(kernel, tree.levels[tree.height-1]);
    // Far-field
    P2M(kernel, tree.levels[tree.height-1]);
    forall the levels l from tree.height-2 to 2 do
        M2M(kernel, tree.levels[l]);
    forall the levels l from 2 to tree.height-2 do
        M2L(kernel, tree.levels[l]);
        L2L(kernel, tree.levels[l]);
    M2L(kernel, tree.levels[tree.height-1]);
    L2P(kernel, tree.levels[tree.height-1]);
```

OpenMP-For : The parallelization of the M2M operator

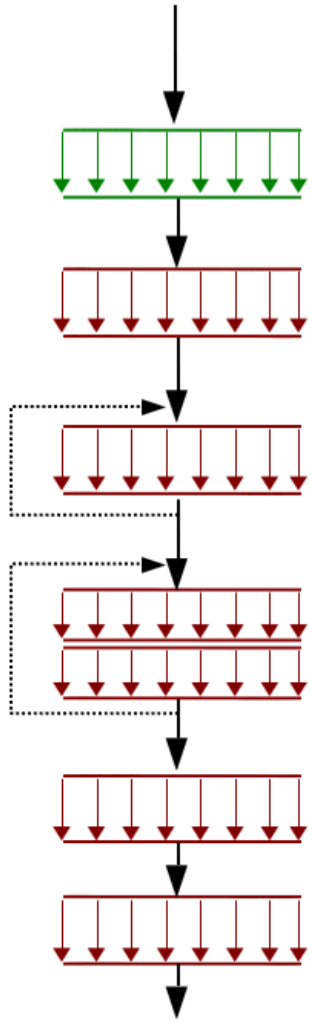
```
function M2M(kernel, level)
    #pragma omp parallel for
        foreach cell cl in level.cells
        do
            kernel.m2m(cl,
            cl.children);
```

Performance can change depending of the particles distribution



OpenMP-For : Fork-Join model

```
function FMM(tree, kernel)
    // Near-field
    P2P(...);
    // Far-field
    P2M(...);
    forall the levels l ... do
        M2M(...);
    forall the levels l ... do
        M2L(...);
        L2L(...);
    M2L(...);
    L2P(...);
```



General introduction to task

A task is composed by:

- A block of instructions
- Some parameters
- An environment

Specificity:

- Can be computed by any thread
- Such thread is not known a priori
- Tasks is native in many "libraries" (OpenMP 3.1, StarPU)

OpenMP-Task : Task based FMM

OpenMP task syntax:

```
#pragma omp task
    some code;
#pragma omp taskwait;
```

Using OpenMP tasks:

- For statement are replaced by a task insertion loop
- **Cells are blocked to increase the granularity**

OpenMP-Task : Global FMM algorithm

```
function FMMtask(kernel, tree)
    #pragma omp parallel
        #pragma omp single
            P2Ptask(kernel, tree.levels[tree.height-1]);
            P2Mtask(kernel, tree.levels[tree.height-1]);
            forall the levels l from tree.height-2 to 2 do
                M2Mtask(kernel, tree.levels[l]);
            forall the levels l from 2 to tree.height-2 do
                M2Ltask(kernel, tree.levels[l]);
                L2Ltask(kernel, tree.levels[l]);
            M2Ltask(kernel, tree.levels[tree.height-1]);
            L2Ptask(kernel, tree.levels[tree.height-1]);
```

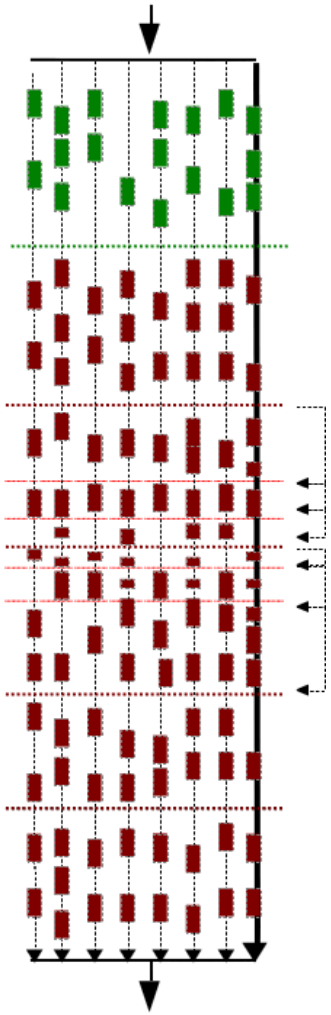
OpenMP-Task : Task based M2M operator

```
function M2M(kernel, level)
    foreach Cell cl in cells[level] do
        #pragma omp task
            kernel.m2m(cl, cl.children);
    #pragma omp taskwait;
```

Several cells are computed together and called a block.

OpenMP-For : Fork-Join model

```
function FMMtask(kernel, tree)
    #pragma omp parallel
        #pragma omp single
            P2Ptask(...);
            P2Mtask(...);
            forall the levels l ... do
                M2Mtask(...);
            forall the levels l ... do
                M2Ltask(...);
                L2Ltask(...);
            M2Ltask(...);
            L2Ptask(...);
```

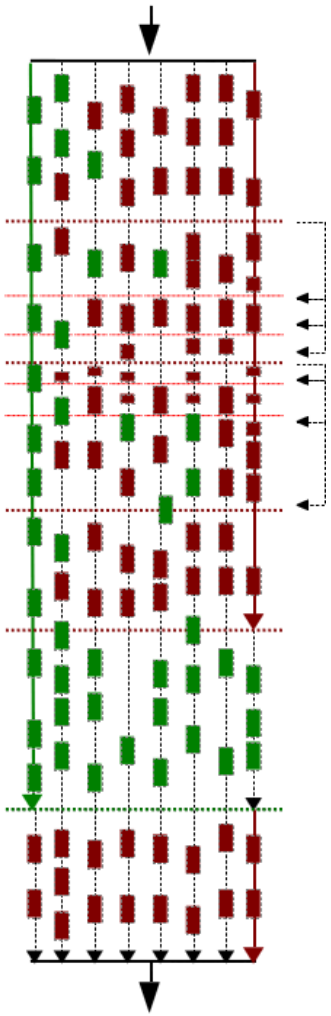


OpenMP-Task : Interleaving FF and NF

```
function FMMtask(kernel, tree)
    #pragma omp parallel
        #pragma omp sections
            #pragma omp section
                P2Ptask(kernel, tree.levels[tree.height-1]);
            #pragma omp section
                P2Mtask(kernel, tree.levels[tree.height-1]);
                forall the levels l from tree.height-2 to 2 do
                    M2Mtask(kernel, tree.levels[l]);
                forall the levels l from 2 to tree.height-2 do
                    M2Ltask(kernel, tree.levels[l]);
                    L2Ltask(kernel, tree.levels[l]);
                M2Ltask(kernel, tree.levels[tree.height-1]);
        #pragma omp single
            L2Ptask(kernel, tree.levels[tree.height-1]);
```

OpenMP-For : Fork-Join model

```
function FMMtask(kernel, tree)
    #pragma omp parallel
        #pragma omp sections
            #pragma omp section
                P2Ptask(...);
            #pragma omp section
                P2Mtask(...);
                forall the levels l ... do
                    M2Mtask(...);
                forall the levels l from ... do
                    M2Ltask(...);
                    L2Ltask(...);
                M2Ltask(...);
        #pragma omp single
            L2Ptask(...);
```



Motivation to use StarPU

Limits of the OpenMP tasks

- No priorities between tasks
- No fine grain (task-grain) dependencies
- Lots of synchronizations

StarPU

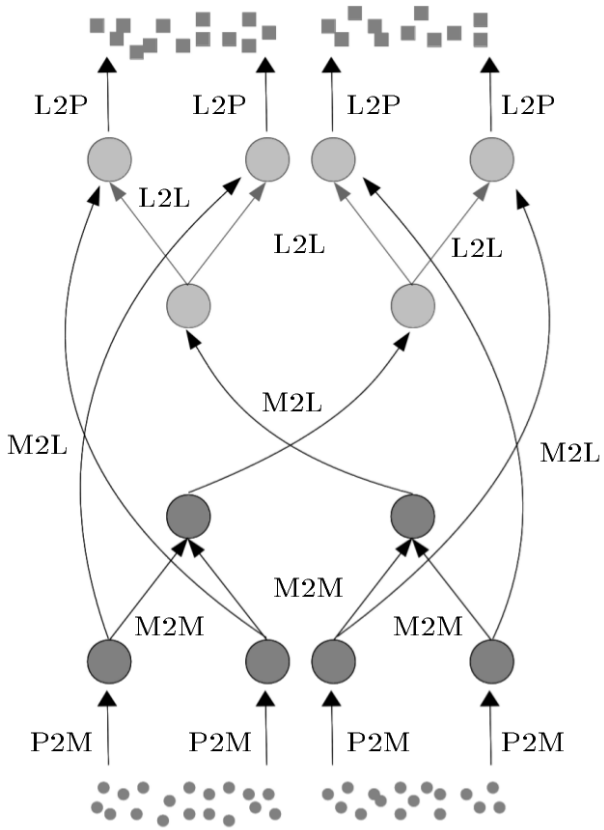
- Is a runtime
- Advanced tasks scheduling
- Tasks dependencies expression
- Gives an abstraction of the architecture (useful for GPU)

StarPU : The FMM task flow (Dag)

Insert_task function:

1: insert_task (operator, ACCESS_MODE, handle, ...)

- callback ← function
- ACCESS_MODE ← READ;WRITE
- handles are StarPU data interface



StarPU : The M2M operator

```
function M2M(kernel, level)
    foreach Block bl in blocks[level] do
        insert_task(kernel.m2m, READ, bl.sub_block,
        WRITE, bl);
```

Results & Performances

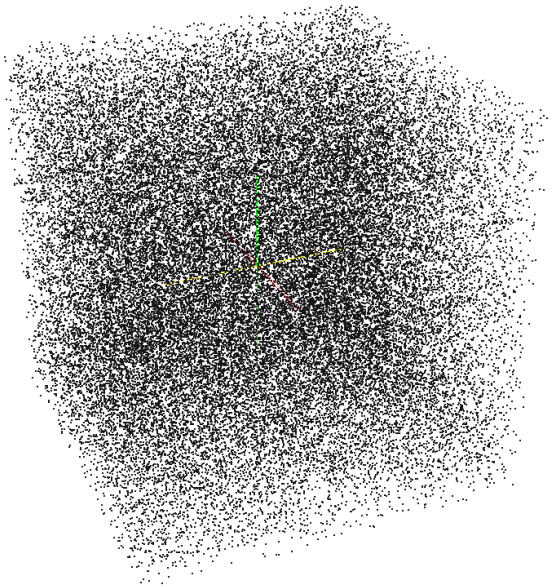
on multi-cores architectures

Test cases

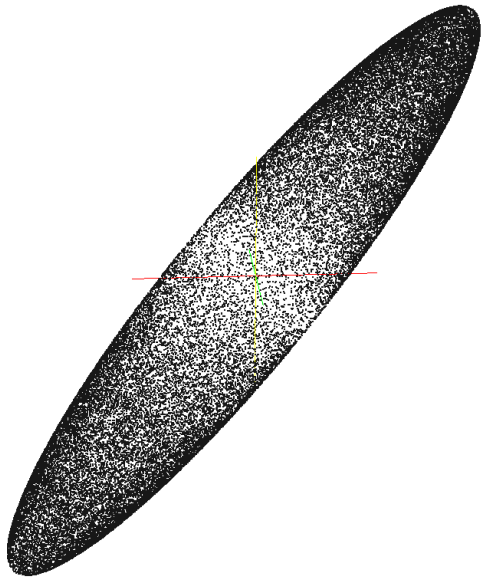
Two test cases

- *Volume* : uniform distribution of particles in 3D inside a cube
- *Surface* : distribution of particles on the surface of an ellipsoid

Not the same difficulty



Box (Volume)



Ellipsoid (Surface)

Experimental Setup

ScalFMM:

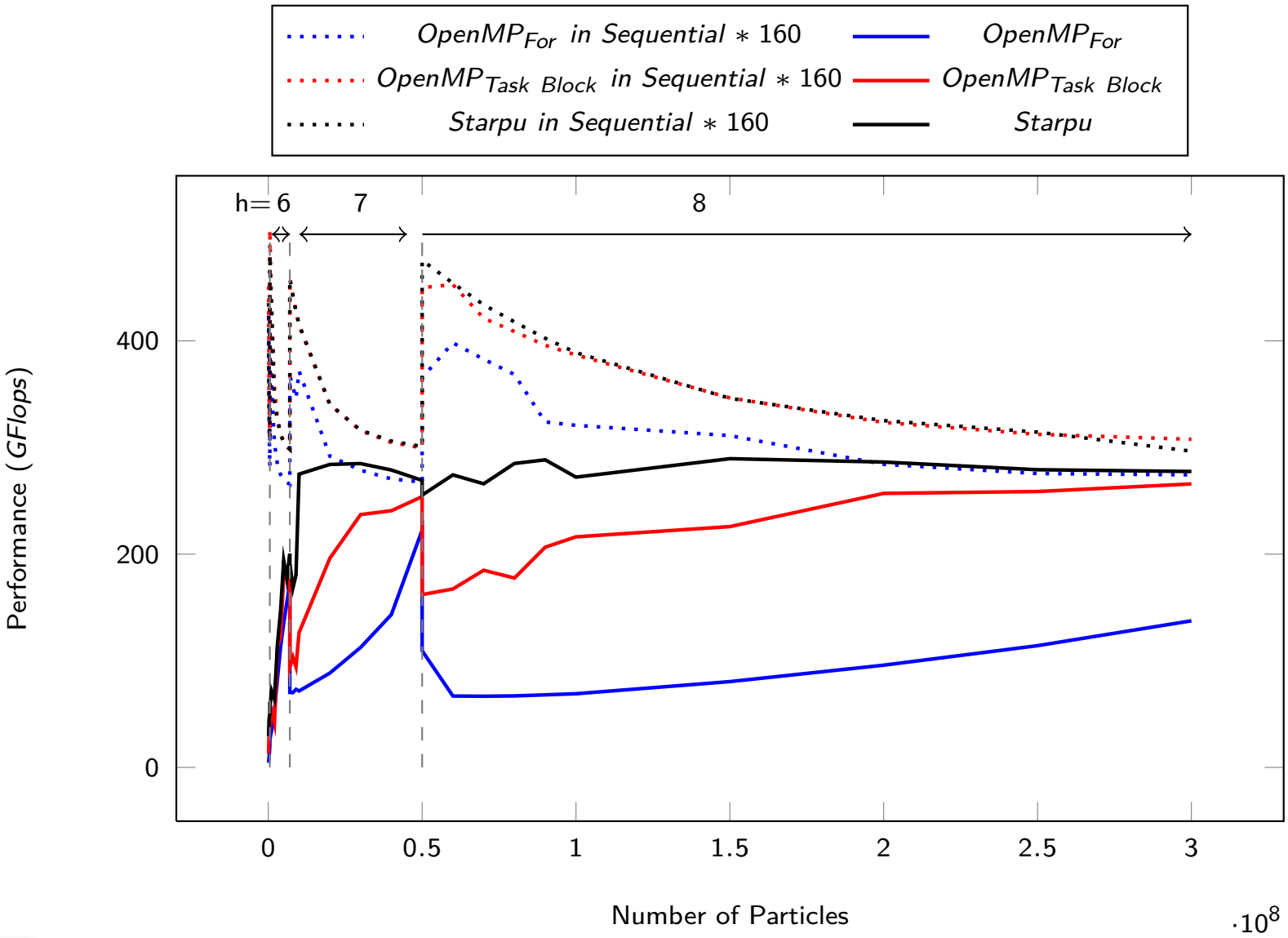
- Developed in C++ (OpenMP, MPI, and now StarPU)
- Available at http://scalfmm-public.gforge.inria.fr/

SGI Altix UV 100 (ccNUMA machine) of 160 cores (20 sockets)

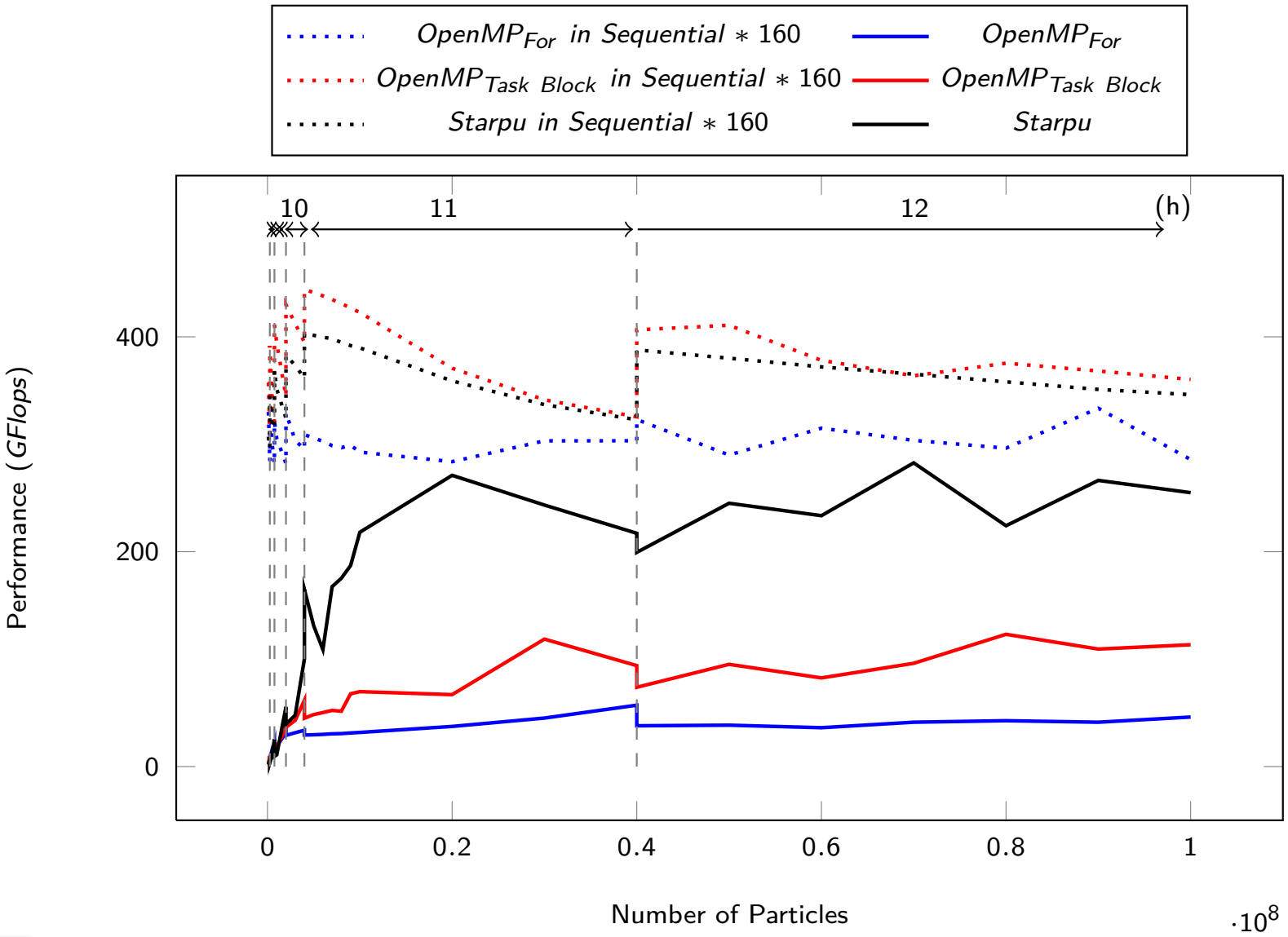
- Twenty octa-core Intel Xeon E7-8837 processors running at $2.67 GHz$
- Each socket has 32 GB of RAM and 24 MB of L3 cache.
- Each CPU core has its own L1 and L2 caches of size 32 KB and 256 KB, respectively.

In the following results we use Intel Compiler (12.0.5).

Performances against particles number (*Volume*)



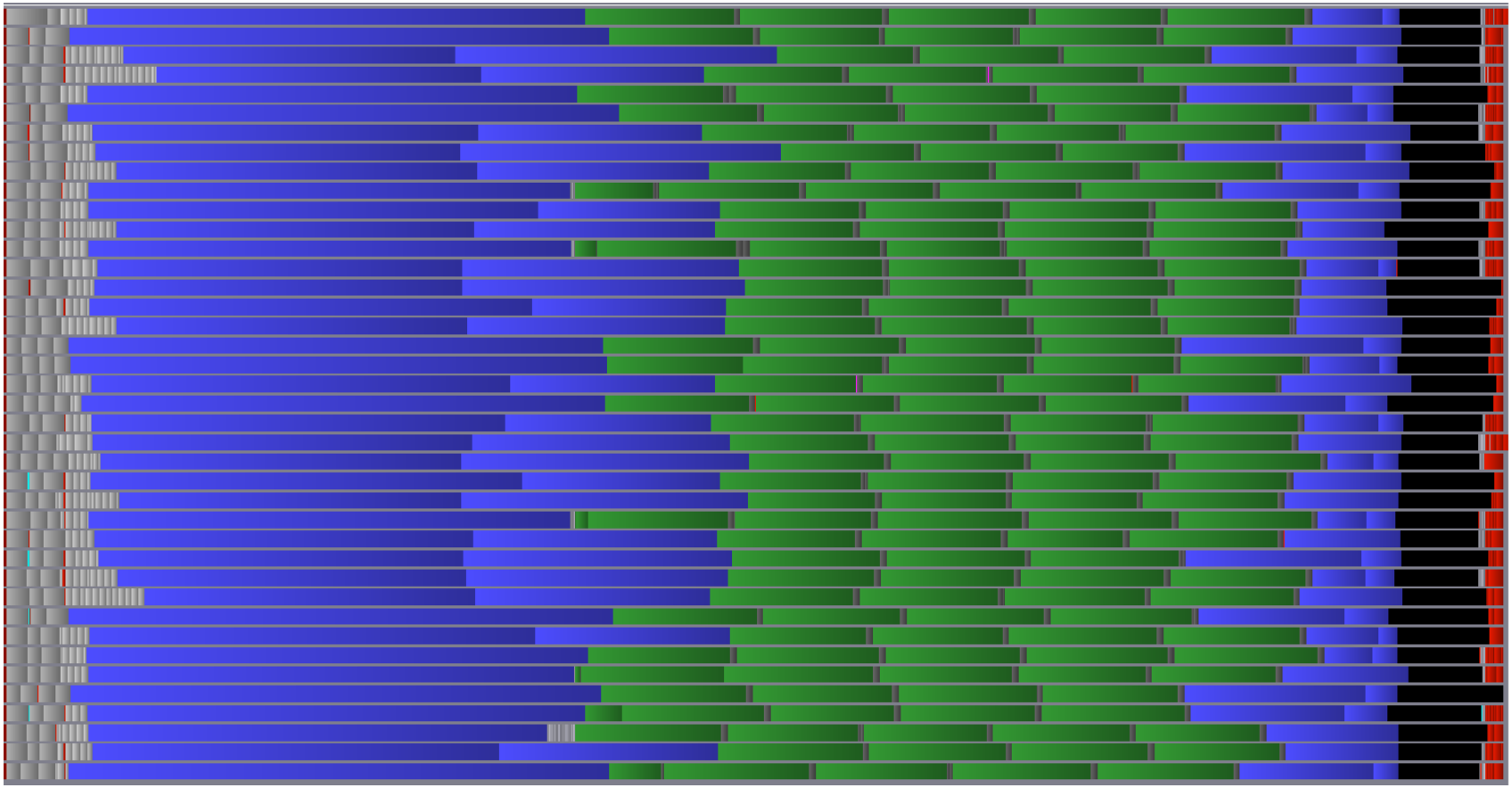
Performances against particles number (*Surface*)



StarPU Execution traces (*Surface*)

For 20.10^6 particles, octree height $h = 11$, granularity $n_g = 6000$, computation time $6.9s$, $Acc = 5$

Legend : P2P, P2M, M2M, M2L, L2L, L2P.



Parallel Efficiency

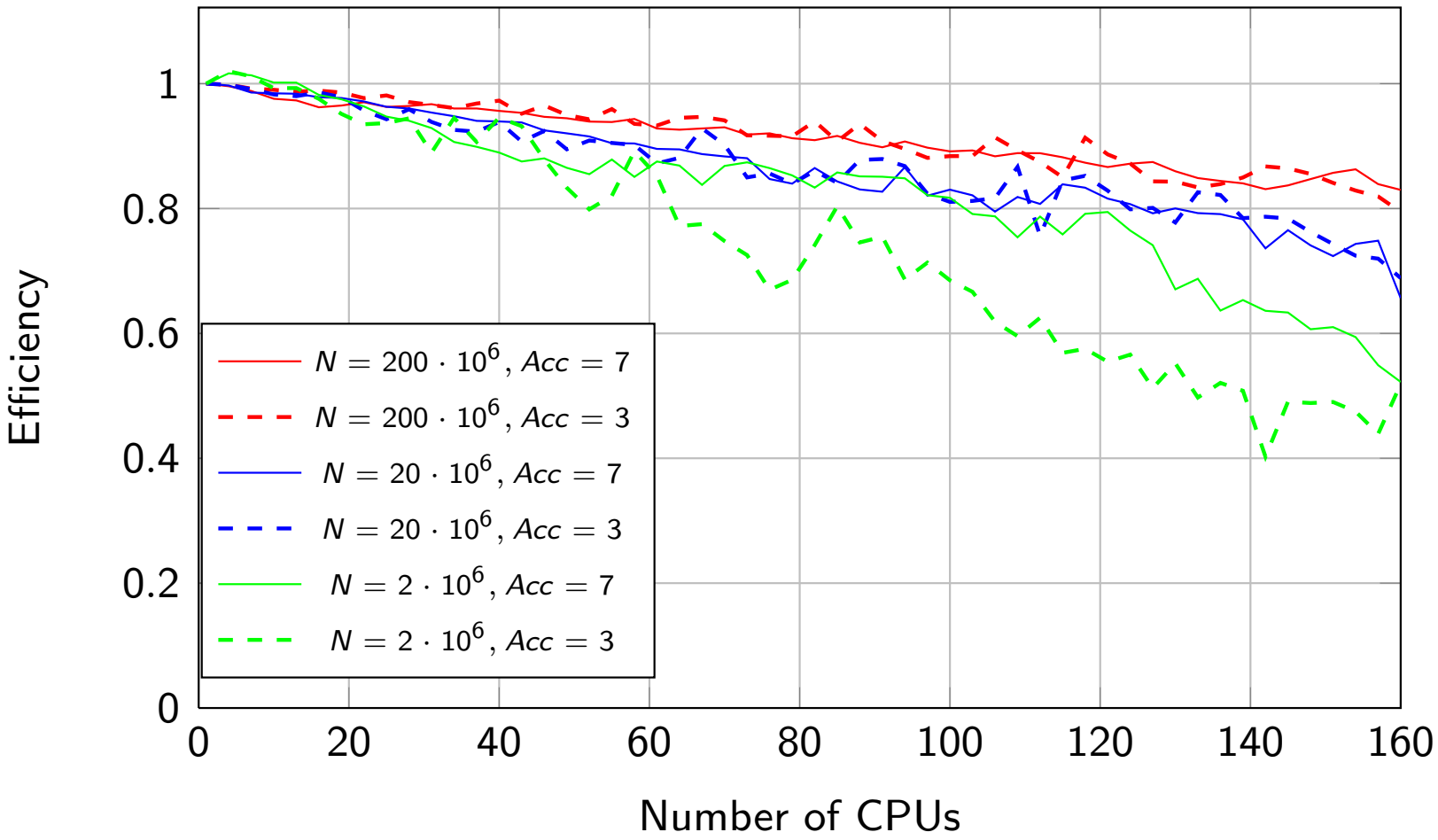
The efficiency is obtained per :

$$e_n = \frac{t_1}{nt_n}$$

Where t_n is the time taken per n threads/processes.
A perfect efficiency is : $e_n = 1$.

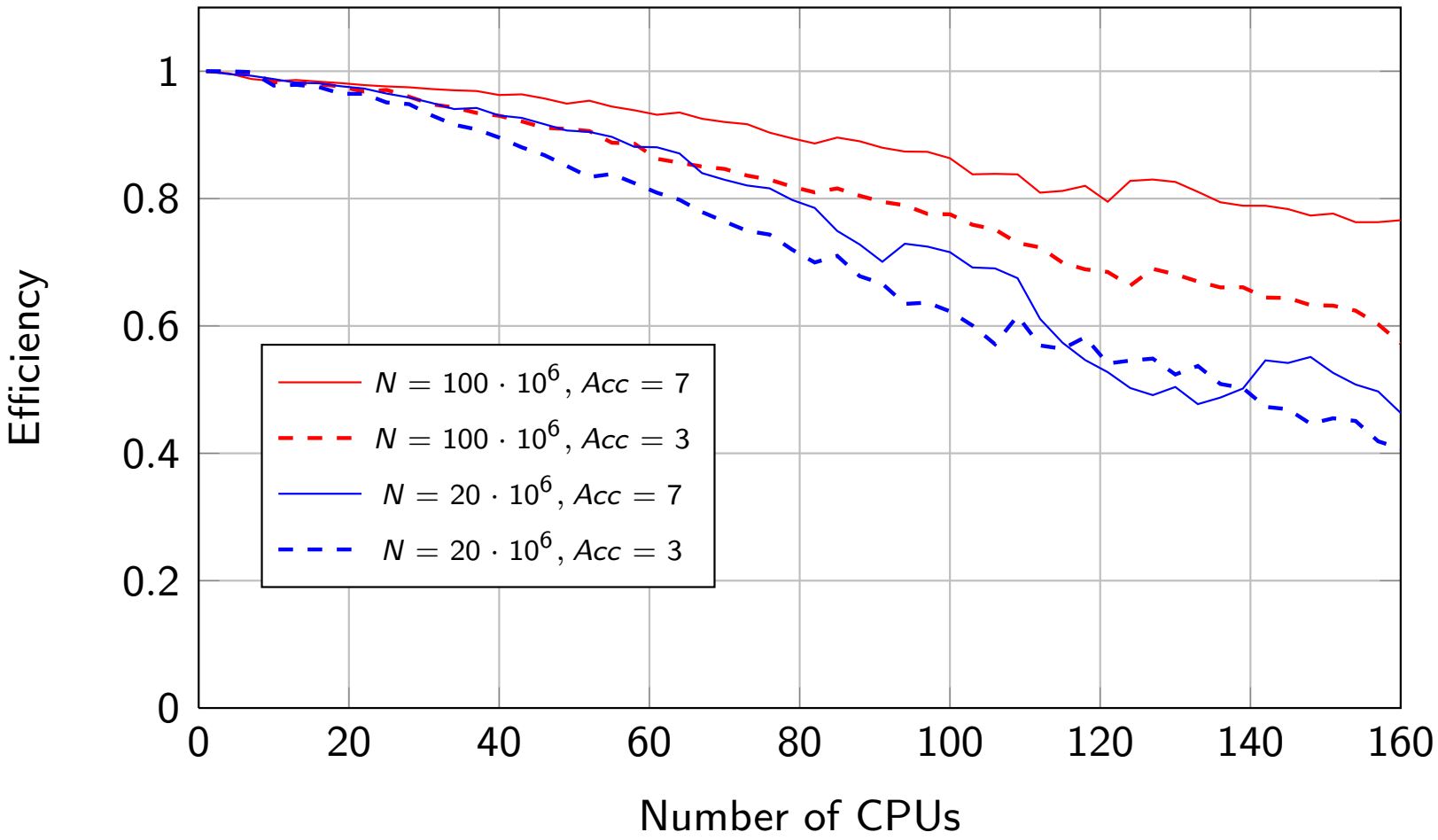
StarPU Efficiency (*Volume*)

For $N = 2 \cdot 10^6$, $n_g = 200$ and $h = 6$, for $N = 20 \cdot 10^6$, $n_g = 1000$ and $h = 7$ and for $N = 200 \cdot 10^6$, $n_g = 3000$ and $h = 8$



StarPU Efficiency (*Surface*)

For $N = 20 \cdot 10^6$, $n_g = 3550$ and $h = 11$ and for $N = 100 \cdot 10^6$, $n_g = 8000$ and $h = 12$



Conclusion

- A new FMM approach
- Good results
- Better pipelining in difficult cases
- Multi GPU

Future work:

- Distributed memory/Multi GPU
- Intel MIC

Acknowledgement

- Inria & FastLA associated team.
- StarPU community.

Thanks - Questions?