Parallelization on Heterogeneous Multicore and Multi-GPU Systems of the Fast Multipole Method for the Helmholtz Equation using a Runtime System

Cyril Bordage
CEA/CESTA
Le Barp, France
INRIA
Bordeaux, France
cyril.bordage@inria.fr

***Abstract*—The Fast Multipole Method (FMM) is considered as one of the top ten algorithms of the 20th century. The FMM can speed up solving of electromagnetic scattering problems. With N being the number of unknowns, the complexity usually $O(N^2)$ becomes $O(N \log N)$ allowing a problem with hundreds of millions of complex unknowns to be solved. The FMM applied in our context has a serious drawback: the parallel version is not very scalable. In this paper, we present a new approach in order to overcome this limit. We use StarPU, a runtime system for heterogeneous multicore architectures. Thus, our aim is to have good efficiency on a cluster with hundreds of CPUs, and GPUs. Much work have been done on parallelization with advanced distribution techniques but never with such a runtime system. StarPU is very useful, especially for the multi-level algorithm on a hybrid machine. At present, we have developed a multi-core and a GPU version. The techniques for distributing and grouping the data are detailed in this paper. The first results of the strategy used are promising.**

Keywords-Fast multipole method (FMM); Helmholtz equation; heterogeneous architecture; parallel algorithm.

I. INTRODUCTION

The main aim is the simulation of the electromagnetic behavior of 3D complex objects in the frequency domain. For that, we use standard numerical methods such as Boundary Integral Equations [1], [2] based on a classical Finite Element approximation of surface Integral Equations such as EFIE and CFIE formulations [3].

These formulations lead to a linear system with a full matrix, which is complex non Hermitian but symmetric. It is solved by an iterative method, which has a complexity of $O(N^2)$, with N being the number of unknowns The complexity comes from the matrix-vector products computed at each iteration. The Fast Multipole Method (FMM) [4] is able to reduce the complexity of these matrix-vector products, and so of the global problem, to $O(N \log(N))$ [5]. In this paper, we will study the FMM only in the context of electromagnetic scattering problems, with the kernel from the Helmholtz equation.

With modern parallel architectures, the parallelization of the FMM is essential, if we want to solve very large problems, which need a lot of memory. The different parallelizations are not efficient on distributed memory architectures and unfortunately, architectures nowadays are becoming more complex, by integrating accelerators like GPUs. With these new architectures, load balancing is more complicated and calculations have to be fitted.

This paper is organized as follows. In Section II, we outline the FMM. In Section III, we briefly describe the different strategies in the parallelization for distributing the computations. Our approach and its justification are explained in Section IV. Finally, in Section V, we present some results.

II. THE FMM

The FMM was introduced by Greengard and Rokhlin in 1987 [4]. In the 90s, the method was applied to electromagnetism by Rokhlin [6] and Chew [5] in its diagonal version. We will present briefly the FMM algebraically from [7],[8]. A good analytic presentation can be found in [9] or [10].

A. Principle

The FMM computes the matrix-vector product:

$$\vec{v} = \mathbb{G}.\vec{u} \tag{1}$$

with: $\mathbb{G}_{i,j} = G(|x_i - x_j|)$.

In our context, the Helmholtz equation, the Green function G, is defined by:

$$G(|x_i - x_j|) = \frac{\mathrm{e}^{ik|x_i - x_j|}}{4\pi|x_i - x_j|}$$

The FMM is based on a space partitioning. First, a partitioning $\mathcal{P}$ of the points is created, based on a geometrical criterion. The partitioning is made up of boxes. If $\mathcal{B}$ is a box of the partition, we define:

$$\begin{cases} \vec{u}^{\mathcal{B}} = (u_i)_{x_i \in \mathcal{B}} \\ \mathbb{G}^{\mathcal{B}_t, \mathcal{B}_s} = (G_{x_i,x_j})_{(x_i,x_j) \in (\mathcal{B}_t \times \mathcal{B}_s)} \end{cases} \tag{2}$$

With the Gegenbauer theorem [11], we have an approximate factorization of $\mathbb{G}^{\mathcal{B}_t, \mathcal{B}_s}$ if $\mathcal{B}_t$ and $\mathcal{B}_s$ are not neighbours, which means that they do not share any vertex.

 ISBN: 978-1-61208-237-0

$$\mathbb{G}^{\mathcal{B}_t,\mathcal{B}_s} \simeq \left(\mathbb{A}^{\mathcal{B}_t}\right)^\star \ \mathbb{T}^{\mathcal{B}_t,\mathcal{B}_s} \ \mathbb{A}^{\mathcal{B}_s}, \qquad \text{with } \mathcal{B}_s \notin \mathcal{V}(\mathcal{B}_t) \quad (3)$$

where:

- $\mathcal{V}(\mathcal{B}_t)$, the set of the neighbours of $\mathcal{B}_t$.
- $\mathbb{A}^{\mathcal{B}}$ is a $P \times N^{\mathcal{B}}$ matrix, called the aggregation matrix.
- $\mathbb{T}^{\mathcal{B}_t,\mathcal{B}_s}$ is a diagonal $P \times P$ matrix, called the translation matrix.
- $\mathbb{A}^\star$ is the conjugate transposition of $\mathbb{A}$.
- $N^{\mathcal{B}}$ is the number of elements in $\mathcal{B}$
- P is inversely proportional to the squared number of boxes, called the number of directions.

The factorization is valid only if $\mathcal{B}_t$ and $\mathcal{B}_s$ are not neighbours. So, we have to split the computation of $\vec{v}^{\mathcal{B}_t}$ in two parts:

$$\vec{v}^{\mathcal{B}_t} = \vec{v}^{\mathcal{B}_t}_{\text{far}} + \vec{v}^{\mathcal{B}_t}_{\text{near}} \quad (4)$$

$\vec{v}^{\mathcal{B}_t}_{\text{near}}$ is computed directly, but for $v^{\mathcal{B}_t}_{\text{far}}$, the factorization yields:

$$\vec{v}^{\mathcal{B}_t}_{\text{far}} \simeq \left(\mathbb{A}^{\mathcal{B}_t}\right)^\star \sum_{\mathcal{B}_s \notin \mathcal{V}(\mathcal{B}_t)} \mathbb{T}^{\mathcal{B}_t,\mathcal{B}_s} \ \mathbb{A}^{\mathcal{B}_s} \vec{u}^{\mathcal{B}_s} \quad (5)$$

$\mathbb{A}^{\mathcal{B}_s} \vec{u}^{\mathcal{B}_s}$ is computed once by $\mathcal{B}_s$, for every $\mathcal{B}_t$. This is the key point of the FMM.

With all these elements, we have a method to quickly compute the product.

B. Single-Level Multipole Method

First, we have to split our object in boxes. Then, the algorithm requires three steps:

1) Aggregation:

$$\vec{F}^{\mathcal{B}_s} = \mathbb{A}^{\mathcal{B}_s} \vec{u}^{\mathcal{B}_s}, \quad \forall \mathcal{B}_s \in \mathcal{P} \quad (6)$$

$\vec{F}^{\mathcal{B}_s}$ is a P vector, called the vector associated with box $\mathcal{B}_s$.

2) Translation:

$$\vec{N}^{\mathcal{B}_t} = \sum_{\mathcal{B}_s \notin \mathcal{V}(\mathcal{B}_t)} \mathbb{T}^{\mathcal{B}_s,\mathcal{B}_t} \ \vec{F}^{\mathcal{B}_s} \quad (7)$$

$\vec{N}^{\mathcal{B}_t}$ is a P vector, it represents the contribution on $\mathcal{B}_t$ from its non neighbours.

3) Disaggregation:

$$\vec{v}^{\mathcal{B}_t}_{\text{far}} = \left(\mathbb{A}^{\mathcal{B}_t}\right)^\star \ \vec{N}^{\mathcal{B}_t} \quad (8)$$

Finally, from (2) and (4, we obtain v:

$$\vec{v} = \sum_{\mathcal{B}_s \in \mathcal{P}} \vec{v}^{\mathcal{B}_s} \quad (9)$$

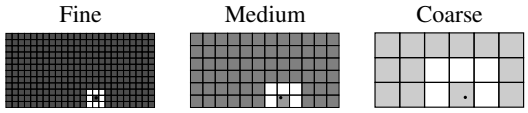

Figure 1. Translations to the gray boxes, of the box with the black point, depending on three partitionings. White boxes are direct calculations.

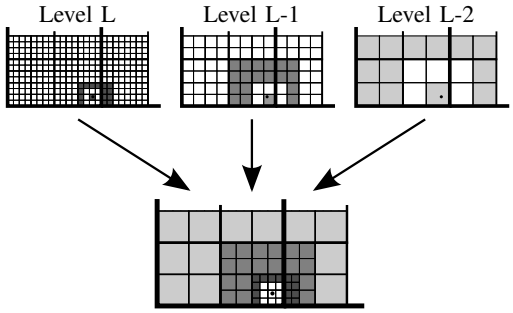

Figure 2. The translations in the ML-FMM.

C. Multi-Level Multipole Method

The most time consuming step in the FMM is the translation step. Indeed, there are many translations, their total amount being equal to the squared number of boxes. By increasing the size of the boxes, the amount of translations is decreased, but at the same time, the amount of direct computations is increased, Figure 1.

In order to reduce the total amount of translations, the Multi-Level Multipole Method (ML-FMM) uses several levels of interleaved partitions. The first level, called highest level is a partition of just one box. The next level is built by splitting the partition of the upper level. The last level is called the lowest one. We call the parent of a box, the box in the previous level, including this box. We define the children of a box reciprocally.

In the ML-FMM, the translation between two points is done in the lowest level, in which the boxes containing the points are not neighbours. Thus, the translations are carried out where they involve the smallest number of boxes. In Figure 2, the translations of the black point are done in three levels depending on the targets. The boxes are translated only to their non neighbour boxes that is to say the children of the neighbours of their parent. The explanation is simple: the non neighbour boxes of the father, and so their children too, are processed by the parent. We call these boxes far neighbours. Finally, the point is translated to each box besides the neighbours of the box, as in the SL-FMM.

The multilevel method involves knowing the vectors in the boxes at each level. A box can be computed by aggregating its child. That corresponds to the sum of all the child vectors multiplied with a shifting matrix $\mathbb{E}$. We remind that P, the size of the aggregated vector $\vec{F}$, depends on the number of boxes and so, on the level. Therefore, the size $\mathcal{P}^l$ of a child vector is lower than P^{l-1}, the size of its parent vector.

 ISBN: 978-1-61208-237-0

The father vector has to be interpolated, it is done by the multiplication with the interpolation matrix $\mathbb{I}$. These two operations are merged in the downward pass.

We also have to fetch the values from the higher levels for the disaggregation step. This operation is the upward pass.

In conclusion, the algorithm differs from the SL-FMM in the translation step, which is replaced by a loop on the levels of 3 steps: the downward pass, the translation and the upward pass. We define $\mathcal{B}^l$, a box on the level l, and $\mathcal{P}^l$, the partition of the level l. The highest level is level 1 and the lowest, level L. The algorithm has five types of operations:

1) Aggregation:

$$\vec{F}^{\mathcal{B}_s^L} = \mathbb{A}^{\mathcal{B}_s^L} \vec{u}^{\mathcal{B}_s^L}, \quad \forall \mathcal{B}_s^L \in \mathcal{P}^L \tag{10}$$

2) Upward pass: $\forall \mathcal{B}_s^l \in \mathcal{P}^l,\ L-1 \leq l \leq 3$,

$$\vec{F}^{\mathcal{B}_s^l} = \mathbb{I}^{l,l+1} \sum_{\mathcal{B}_s^{l+1} \subset \mathcal{B}_s^l} \mathbb{E}^{\mathcal{B}_s^l, \mathcal{B}_s^{l+1}} \vec{F}^{\mathcal{B}_s^{l+1}} \tag{11}$$

where:

- $\mathbb{E}^{\mathcal{B}^l, \mathcal{B}^{l+1}}$ is a diagonal $P^{l+1} \times P^{l+1}$ matrix.
- $\mathbb{I}^{l,l+1}$ is a $P^l \times P^{l+1}$ matrix.

3) Translation: $\forall \mathcal{B}_s^l \in \mathcal{P}^l,\ L \leq l \leq 3$,

$$\vec{N}_T^{\mathcal{B}_t^l} = \sum_{\mathcal{B}_s^l \in \mathcal{V}_{\text{far}}(\mathcal{B}_t^l)} \mathbb{T}^{\mathcal{B}_t^l, \mathcal{B}_s^l}\ \vec{F}^{\mathcal{B}_s^l} \tag{12}$$

4) Downward pass: $L-1 \leq l \leq 3$,

$$\vec{N}^{\mathcal{B}_t^{l+1}} = \begin{cases} \vec{N}_T^{\mathcal{B}_t^{l+1}} & \text{if } l = 3 \\ \vec{N}_T^{\mathcal{B}_t^{l+1}} + \left(\mathbb{E}^{\mathcal{B}_t^l, \mathcal{B}_t^{l+1}}\right)^\star \left(\mathbb{I}^{l,l+1}\right)^\star \vec{N}^{\mathcal{B}_t^l} & \\ & \text{if } l \in [\![4, L]\!] \end{cases} \tag{13}$$

5) Disaggregation: $\forall \mathcal{B}^L \in \mathcal{P}^L$,

$$\vec{v}_{\text{far}}^{\mathcal{B}_t^L} = \left(\mathbb{A}^{\mathcal{B}_t^L}\right)^\star \mathbb{N}^{\mathcal{B}_t^L} \tag{14}$$

III. Parallelization of the FMM

The parallelization of the FMM depends on the context. For example, for Laplace or Stokes kernels, good performances have been achieved with hundreds of thousands cores [12]. Unfortunately, in our context, the parallelization of the FMM is not so efficient. The reason is the size of vectors which increases when we go up in the tree. Thus, the amount of computations is nearly the same at each level unlike for Laplace.

That is why much work has been carried out on its parallelization since the 90s. Research works has been focused on the ML-FMM, but recent years, the SL-FMM has drawn attention with a better scalability. In addition to parallelization on many CPUs, there is a trend towards using GPUs to carry out calculations.

A. The ML-FMM

The first parallelization is based on the distribution of the boxes among the processors at each level .

Unfortunately, as we can see in [13], this parallelization on 16 processors gives poor results in terms of scalability. The reason is that at a given level, there are not enough boxes to share between processors so a good load balancing is impossible. It seems not to be important because it concerns only the boxes on the levels, which have less computations. But the problem is that, in the FMM applied to electromagnetism, each level needs the same amount of calculations, since P_l, the size of the vectors in (11) (12) (13), is in inverse proportion to the squared number of boxes. As a result, the bad parallelization of these levels has a great negative impact on the global scalability.

Velamparambil and Chew discovered [13] another strategy to overcome that. The previous distribution is kept for the fine levels, called the distributed layer. For the coarse levels, the vector attached to a box is split into blocks and distributed among the processors. The computations are carried out by block on each processor. In the levels with the new distribution, called the shared layer, data have to be replicated.

ErgÃ¼l and GÃ¼rel [14], presented another distribution called the hierarchical distribution. It consists of distributing the fields not only for the coarse levels but at each level . With this distribution 60% of efficiency can be achieved with 128 processors, whereas we have only 40% with the hybrid distribution and 20% with the simple distribution.

B. The SL-FMM

The ML-FMM has a better complexity but its parallelization is not efficient enough and all the possible strategies seem to have been considered. That is why Waltz et al. [15] take an interest in the SL-FMM in the context of the parallelization. Indeed, in the single-level method, the block of the vector linked to the boxes, can be computed independently. We just have to gather them at the end of the algorithm. So, the parallelization over the samples of field will be very efficient. It is not the same for the ML-FMM where the blocks have to be gathered for downward pass.

Moreover, in [16], Wagner et al. proposed the FMM-FFT, which reduces the complexity of the translation stage. The complexity becomes $O(N^{4/3} \log^{2/3} N)$, which represents an important improvement compared to the $O(N^{3/2})$ [15]. In [17], we have a proof of the efficiency of the FMM-FFT, with a perfect scalability up to 512 processors and a very good one for 1024.

Last year, Taboada et al. [18] combined the FMM-FFT and the ML-FMM. When the number of processors is bigger than the number of nodes, they use the FMM-FFT. For the next levels down, they use independent ML-FMM on each node. With this new method, they have solved a problem with 620 million unknowns.

 ISBN: 978-1-61208-237-0

C. With GPUs

Work have been done in the FMM, but only for Laplace, or other non oscillatory kernels. Good efficiencies have been achieved on GPUs thanks to the BLAS [19].

The important points for using a GPU in scientific applications are the consistency of the computations and enough computations compared to the data transfers. The data have to be well-sorted to benefit from the coalescing accesses.

IV. Our strategy

Many efforts have been done on the distribution of the computations in the last fifteen years. There has also been research on computation scheduling with tasks queues [20]. We have chosen the hierarchical distribution, and for the top level, a simple SLFFM, which can be upgraded in a FMM-FFT. We have decided to focus on computation scheduling because a good scheduling is the key in modern machines, with heterogeneous processing units. A good scheduling depends on the machine: speed of the processing units, bus speed, network speed, etc. The schedule has to fit the algorithm but also the machine, as the computations have to be adapted to the processing units. Another important point for a good scalability is to hide the communications by computations.

A. The dynamic scheduling

Our aim is to compute the FMM on a supercomputer with shared and distributed memory, thousands of CPUs and GPUs. For that we use the same distribution between the nodes as in the combination of the FMM-FFT and the ML-FMM. In a node, we use the dynamic scheduler StarPU [21]. It can handle the scheduling on CPUs and GPUs with different strategies: greedy, work stealing, minimal termination time, priority, etc. It automates transfers throughout heterogeneous machines and favours data locality. Tasks and data dependencies must be declared and StarPU does the rest.

The strategy, which has been considered, is the minimal termination time. It takes a task execution model and data transfer model into account to know where a task will end the soonest. The models can be provided for StarPU or it can build them.

B. Efficient operations

To be efficient, the tasks handled by the task scheduler should imply enough computations to hide the costs of the scheduling and of the data transfers. This is especially the case for the GPUs where the data transfers are more costly and because they can do simultaneous computations.

1) The data: Many computations have to be grouped in a same task. For that, the same strategy as in parallelization III-B is used: groups are made with many directions for many boxes.

To avoid calculation starvation and deadlocks, the granularity must be low. But high enough to permit the GPUs to be efficient. The number and the size of the blocks should be tuned depending on the machine and the input data. For that, we can set the number of tasks by level, depending on the numbers of computing workers.

2) Dependencies: All the operations except the translation and the upward pass can be done just by using the data contained in one direction block. Thus, there is no communication.

For the upward pass, the computation of a parent box needs all the directions of all its children. Transposed to our blocks, that becomes: a direction block needs all its child box blocks with all their direction blocks. This can be done by using one task for each direction block in the child level but by reducing the out data. StarPU can deal with reduce operations itself.

The translations to a box use all its far neighbours. Consequently, for translating all the boxes in a block, the far neighbours of all the boxes are needed. Although most of the far neighbours are in the same block, some are external to the block. Therefore, the translations in a block need data from other blocks. To limit the memory accesses between blocks, the data of the external far neighbours are copied to the block; see Figure 3. Thus, all the translations will be internal to the block. That adds synchronization because a translation can occur in a block only if all its neighbours have been computed. In fact, this is not a problem because the translations are useful only for the downward pass.

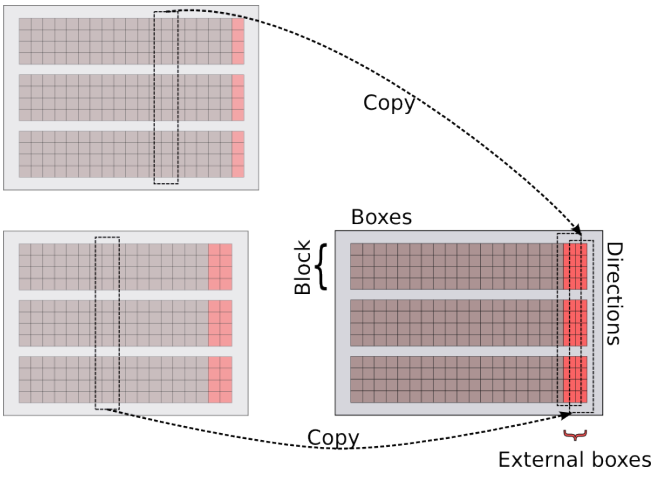

Figure 3. The distribution of the data

3) With GPUs: All the operations in the FMM are simple to implement on GPUs. There are mainly matrix vector multiplications. This is relatively efficient on the GPUs when the sizes of the matrix are not to small. We just have make use of coalescent accesses and avoid bank conflicts.

 ISBN: 978-1-61208-237-0

V. RESULTS

For the time being, the tests have only been done for the shared memory and for the GPUs but not for the distributed memory yet.

A. On shared memory

The test was done on a 2 Hexa-core Westmere Intel Xeon X5650 2.67 GHz (10.664 GFlops by core) with a sphere of 2 million points at 500 MHz. The results are presented in the Table I. The scalability is strong with 12 processors but

# cores	# blocks by level	Time (s)	Efficiency
1	1	80.1	100%
2	8	41.1	97%
4	8	21.4	94%
6	10	14.5	92%
8	10	10.9	92%
10	10	8.7	92%
12	10	7.3	91%

Table I
ON SHARED MEMORY WITH A 2 MILLION SPHERE

we have to do some tests on a machine with more CPUs. When we look at the scheduling, Figure 4, we find that the copies of the far neighbours (called block sharing) do not represent much time. Therefore, the cost of the parallelism is insignificant. The other important point is the waste time of the processors (called blocked): instead of executing a task, a processor is waiting for a new task.

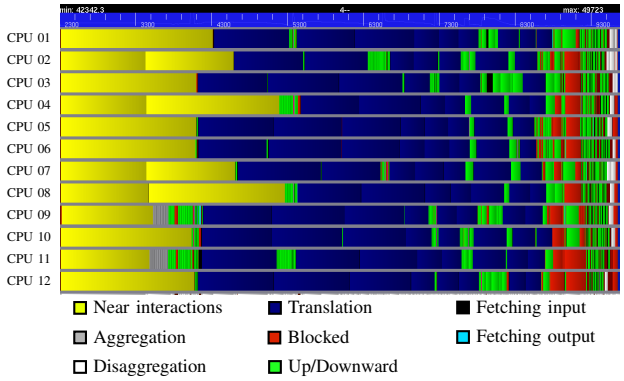

Figure 4. Gantt diagram for the execution on 12 cores

This is the result of an insufficient number of tasks, but here, if we increase the number of tasks, the cost of the parallelism becomes significant and the global time increases. Nevertheless, to avoid this kind of situation, we can favour the tasks that create other tasks and so parallelism. These tasks are the aggregations and the upward passes. In our scheduler, favouring a task can be done easily by adding a priority to this task.

B. With GPUs

The aim of our approach is to use GPUs. We have only done preliminary tests on the same machine with 3 NVIDIA Tesla M2070 (1 TFlops). 3 processors are dedicated to the handling of the 3 GPUs by StarPU. The test case is a 10 meter sphere with 2 million points at 1 GHz. At present we have only implemented the aggregation, the translations and the near interactions.

The aggregation is computed at the same speed on the GPUs, Figure 5. The near interactions are 7 times faster. The translation is 50 times faster. This last result is very good if we look at the flops of the processing units.

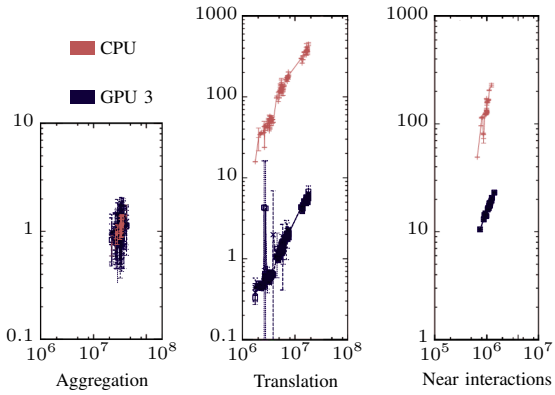

Figure 5. Execution time (μs) depending on the input size (B)

For having this efficiency, we must have enough directions (at least 100). But we have planned to develop kernels for small numbers of directions. StarPU will choose the better kernel depending on the size of the inputs.

The scheduling, Figure 6, is good on the GPUs but poor on CPUs. This is due to the fact that the blocks are too big for the CPUs. But if we decrease the size, we will loose our efficiency on the GPUs. That is why we want to create tasks with different sizes.

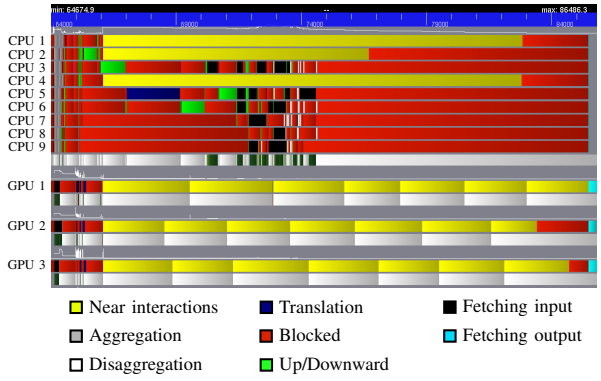

Figure 6. Gantt diagram for the execution on 9 cpu cores and 3 gpus

Without the GPUs, the global time of the computations is 4 times greater. It is not much as compared to the acceleration of the tasks separately. We hope for better results with

 ISBN: 978-1-61208-237-0

a better block creation and with the implementation of all the tasks on GPUs.

VI. Conclusion and future work

In this paper, we have studied the parallelization of the FMM, applied to scattering problems, with a dynamic scheduler. We have also seen how to arrange the computations in order to permit an efficient scheduling. Thus, on shared memory, the strong scalability is good. On GPUs,our first tests encourage us to continue. Our work is promising; but, to make conclusions we still have a lot of work to do: upward pass on GPUs, improvement of the kernels, strategies for the tasks, adaptive method, and MPI.

References

[1] D. Jones, "Acoustic and electromagnetic waves," *Oxford/New York, Clarendon Press/Oxford University Press, 1986, 764 p.*, vol. 1, 1986.

[2] J. Stratton, *Electromagnetic theory*. Wiley-IEEE Press, 2007, vol. 33.

[3] D. L. Colton and R. Kress, *Integral equation methods in scattering theory*, ser. Pure and Applied Mathematics (New York). New York: John Wiley & Sons Inc., 1983, a Wiley-Interscience Publication.

[4] L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations* 1," *Journal of Computational Physics*, vol. 73, no. 2, pp. 325–348, 1987.

[5] J. Song and W. Chew, "Multilevel fast-multipole algorithm for solving combined field integral equations of electromagnetic scattering," *Microwave and Optical Technology Letters*, vol. 10, no. 1, pp. 14–19, 1995.

[6] R. Coifman, V. Rokhlin, and S. Wandzura, "The fast multipole method for the wave equation: A pedestrian prescription," *Antennas and Propagation Magazine, IEEE*, vol. 35, no. 3, pp. 7–12, 2002.

[7] E. Darve, "The fast multipole method. I. Error analysis and asymptotic complexity," *SIAM J. Numer. Anal.*, vol. 38, no. 1, pp. 98–128 (electronic), 2000.

[8] X. Sun and N. Pitsianis, "A matrix version of the fast multipole method," *Siam Review*, vol. 43, no. 2, pp. 289–300, 2001.

[9] G. Sylvand, "La méthode multipôle rapide en électromagnétisme. Performances, parallélisation, applications," 2002.

[10] W. Chew, E. Michielssen, J. Song, and J. Jin, *Fast and efficient algorithms in computational electromagnetics*. Artech House, Inc. Norwood, MA, USA, 2001.

[11] M. Abramowitz and I. Stegun, *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables. National Bureau of Standards Applied Mathematics Series 55. Tenth Printing*, 1972.

[12] A. Rahimian, I. Lashuk, S. Veerapaneni, A. Chandramowlishwaran, D. Malhotra, L. Moon, R. Sampath, A. Shringarpure, J. Vetter, R. Vuduc, D. Zorin, and G. Biros, "Petascale direct numerical simulation of blood flow on 200k cores and heterogeneous architectures," *SC Conference*, pp. 1–11, 2010.

[13] S. Velamparambil and W. Chew, "Analysis and performance of a distributed memory multilevel fast multipole algorithm," *Antennas and Propagation, IEEE Transactions on*, vol. 53, no. 8, pp. 2719–2727, 2005.

[14] Ö. Ergül and L. Gürel, "Hierarchical parallelization strategy for multilevel fast multipole algorithm in computational electromagnetics," *Electron. Lett.*, vol. 44, pp. 3–5, Jan. 2008.

[15] C. Waltz, K. Sertel, M. Carr, B. Usner, and J. Volakis, "Massively parallel fast multipole method solutions of large electromagnetic scattering problems," *Antennas and Propagation, IEEE Transactions on*, vol. 55, no. 6, pp. 1810–1816, 2007.

[16] R. Wagner, J. Song, and W. Chew, "Monte Carlo simulation of electromagnetic scattering from two-dimensional random rough surfaces," *Antennas and Propagation, IEEE Transactions on*, vol. 45, no. 2, pp. 235–245, 2002.

[17] J. Mouriño, A. Gómez, J. Taboada, L. Landesa, J. Bértolo, F. Obelleiro, and J. Rodríguez, "High scalability multipole method. Solving half billion of unknowns," *Computer Science-Research and Development*, vol. 23, no. 3, pp. 169–175, 2009.

[18] J. Taboada, M. Araujo, J. Bertolo, L. Landesa, F. Obelleiro, and J. Rodriguez, "Mlfma-fft parallel algorithm for the solution of large-scale problems in electromagnetics," *Progress In Electromagnetics Research*, vol. 105, pp. 15–30, 2010.

[19] N. Gumerov and R. Duraiswami, "Fast multipole methods on graphics processors," *Journal of Computational Physics*, vol. 227, no. 18, pp. 8290–8313, 2008.

[20] G. Sylvand, "Performance of a parallel implementation of the FMM for electromagnetics applications," *International Journal for Numerical Methods in Fluids*, vol. 43, no. 8, pp. 865–879, 2003.

[21] C. Augonnet, "Scheduling Tasks over Multicore machines enhanced with Accelerators: a Runtime System's Perspective," Ph.D. dissertation, Université Bordeaux 1, 351 cours de la Libération — 33405 TALENCE cedex, Dec. 2011.

 ISBN: 978-1-61208-237-0