# Scalable Fast Multipole Method for Electromagnetic Simulations

Nathalie Möller[1]✉, Eric Petit[2], Quentin Carayol[1], Quang Dinh[1] and William Jalby[3]

[1] Dassault Aviation, France
{nathalie.moller,quentin.carayol,quang.dinh}@dassault-aviation.com
[2] Intel, France
eric.petit@intel.com
[3] LI-PaRAD, University of Versailles, France
william.jalby@uvsq.fr

**Abstract.** To address recent many-core architecture design, HPC applications are exploring hybrid parallel programming, mixing MPI and OpenMP. Among them, very few large scale applications in production today are exploiting asynchronous parallel tasks and asynchronous multithreaded communications to take full advantage of the available concurrency, in particular from dynamic load balancing, network, and memory operations overlapping. In this paper, we present our first results of ML-FMM algorithm implementation using GASPI asynchronous one-sided communications to improve code scalability and performance. On 32 nodes, we show an 83.5% reduction on communication costs over the optimized MPI+OpenMP version.

**Keywords:** CEM, MLFMM, MPI, PGAS, TASKS

## 1 Introduction

The stability of the architecture paradigm makes more predictable the required effort to port large industrial high performance codes from a generation of supercomputers to another. However, recent advances in hardware design result in an increasing number of nodes and an increasing number of cores per node: one can reasonably foresee thousands of nodes mustering thousands of cores, with a subsequent decrease of memory per core. This shift from latency oriented design to throughput oriented design requires application developers to reconsider their parallel programming usage outside the current landscape of production usage. Due to the large and complex code base in HPC, this tedious code modernization process requires careful investigation. Our objective is to expose efficiently fundamental properties such as concurrency and locality.

In this case-study, this is achieved thanks to asynchronous communication and overlapping with computation. We demonstrate our methodology on a Dassault Aviation production code for Computational Electromagnetism (CEM)

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_47

implementing the Multi-Level Fast Multipole Method (ML-FMM) algorithm presented in section 2. This complex algorithm is already using a hybrid MPI and OpenMP implementation which provides state of the art performance compared to similar simulations [6], as discussed in related work section 2.2. To put priorities in the modernization process, we evaluate the potential of our optimization with simple measurements that can be reproduced on other applications. This evaluation is presented in section 3. With this algorithm, the three problems to address are load-balancing, communication scalability, and overlapping. A key common aspect of these issues is the lack of asynchronism in all levels of parallelism. In this paper, we will focus on exploring the impact of off-line load-balancing strategies in section 4 and introducing asynchronism in communications in section 5.

The result section 6 shows, on 32 nodes, an 83.5% improvement in communication time over the optimized MPI+OpenMP version. Further optimizations to explore are discussed in future work. To demonstrate our load balancing and communications improvement for ML-FMM, we are releasing *FMM-lib* library [1] under LGPL-3 license.

## 2 SPECTRE and MLFMM

SPECTRE is a Dassault Aviation simulation code for electromagnetic, and acoustic applications. It is intensively used for RCS (Radar Cross-Section) computations, antenna design and external acoustic computations. These problems can be described using the Maxwell equations. With a Galerkin discretization, the equations result in a linear system with a dense matrix: the Method of Moments (MoM). Direct resolution leads to an O($N^3$) complexity, where N denotes the number of unknowns. A common approach presented in section 2.1 to solve larger systems is to use the Multi-Level Fast Multipole Method (MLFMM) to reduce the complexity to O($NlogN$)[10]. In SPECTRE, the MLFMM is implemented with hybrid MPI + OpenMP parallelism. For the time being, all MPI communications are blocking.

### 2.1 The FMM algorithm

The FMM (Fast Multipole Method) has been introduced in 1987 by L.Greengard and V. Rohklin[1] and is part of the 20th century top ten algorithm [5]. The algorithm relies on the properties of the Green kernel. Let us consider a set of n points $x_p$ and a function $f$ with known values at each of these points. The Fast Multipole Method (FMM) is an algorithm which allows, for all $p \leq n$, fast computation of the sums :

$$\sigma(p) = \sum_{q \leq n \ q \neq p} G(x_p - x_q) f(x_q),$$

where G(.) is the Green kernel. In two dimensions, a naive computation of these sums would require a large number of operations in $O(n^2)$, whereas FMM yields a result in $O(n)$ operations.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_47

The FMM relies on an accurate approximation of the Green kernel, hierarchical space decomposition and a criterion of well separation. The rationale is to model the distant point to point interactions by hierarchically grouping the points into a single equivalent point. Therefore the FMM relies on the accuracy of this far-field low-rank approximation. To make the approximation valid, the group of points has to be far enough from the target point: this is the *well-separated* property.

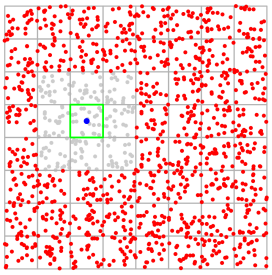

**Fig. 1.** Near Field and Far Field characterization in 2D.

Figure 1 shows a quadtree covering a two-dimensional space and defining the near field and the far field of the particles in the green square. The red particles are *well-separated*: they form the far field and interactions are computed using the FMM. The grey particles outside the green square are too close: they form the near field and interactions are computed using the MoM.

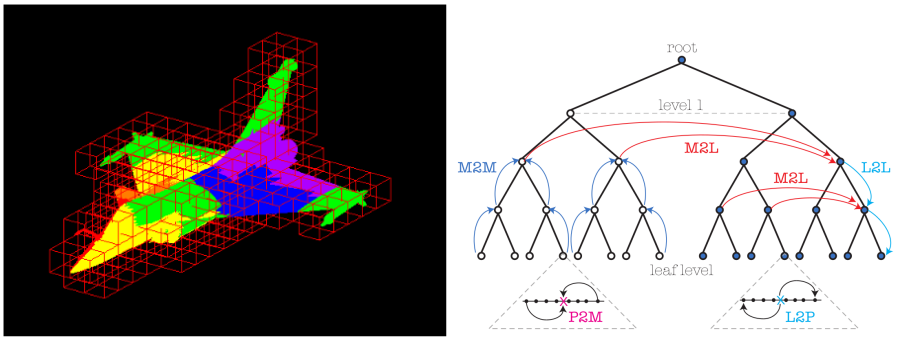


**Fig. 2.** Left: Hierarchically 3D partitioning, Right: 2D quadtree with FMM operators

As illustrated in figure 2, the 3D computational domain is hierarchically partitioned using octrees, and the FMM algorithm is applied using a two-step tree traversal. The upward pass aggregates children's contributions into larger parent

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_47

nodes, starting from the leaf level. The downward pass collects the contributions from same level source cells, which involves communications, and translates the values from parents down to children. The operators are commonly called P2M (Particle to Multipole) and M2M (Multipole to Multipole) for the first phase and M2L(Multipole to Local), L2L (Local to Local) and L2P (Local to Particle) for the second one.

Compared to n-body problems, electromagnetic simulation introduces a major difference: at each level, the work to compute a multipole is doubling, which leads the complexity to O($NlogN$) instead of O($N$). Moreover, FMM is referred to as MLFMM, for Multi-Level FMM.

### 2.2 Related Work

**Parallel programming and core optimization** At node level, [11], Chandramowlishwaran et al. have done extensive work to optimize the computations with manual SIMD, data layout modifying to be vectorization friendly, replacement of matrix storage by on-the-fly computations, use of FFTs and OpenMP parallelization. These optimizations have been implemented in the KIFMM code. In [23], Yokota et al. also achieve parallelisation through OpenMP and vectorization with inline assembly. In [18] and [16], they discuss data driven asynchronous versions using both tasks, at node level with Quark, and on distributed systems with TBB and sender-initiated communications using Charm++. Milthorpe et al. use the X10 programming language [19] which supports two levels of parallelism: a partitioned global address space for communications and a dynamic tasks scheduler with work stealing. In [8] and [9], Agullo et al. automatically generate a directed acyclic graph and schedule the kernels with StarPU within a node that may expose heterogeneous architectures. Despite being impressive demonstrators opening a lot of opportunities, to our best knowledge, none of these codes has been integrated with all their refinements in large production codes to solve industrial use-cases. Furthermore, they require adaptation to match the specific properties of electromagnetic simulation with doubling complexity at each level of the tree.

**Load Balancing and communication pattern** In the literature [16], two main methods are identified for fast N-Body partitioning, and can be classified into Orthogonal Recursive Bisection (ORB) or Hashed Octrees (HOT). ORB partitioning consists in recursively cutting the domain into two sub-domains. This method creates a well-balanced binary tree, but is limited to power of two numbers of processors. Hashed octrees partition the domain with space-filling curves. The most known are Morton and Hilbert. Efficient implementation relies on hashing functions and are, therefore, limited to 10 levels depth with 32 bits or 21 levels with 64 bits.

In [17], Lakshuk et al. propose to load balance the computational work with a weighted Morton curve. Weights, based on the interaction lists, are computed and assigned to each leaf. In a similar approach, Milthorpe et al. [19] propose a

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_47

formula to evaluate at runtime the computational work of the two main kernels: P2P and M2L. Nevertheless current global approaches on space filling curve do not consider the communication costs.

A composite approach is proposed by Yokota et al. [16]. They use a modified ORB (Orthogonal Recursive Bisection) method, combined with a local Morton key so that the bisections align with the tree structure. For the local Morton key, particles are weighted considering computation and communication using ad-hoc formula. This is complemented by another work of the same author in [24] and [15] about FMM communication complexity. However, their model cannot be generalized to our use-cases.

In [12], Cruz et al. elaborate a graph-based approach to load balance the work between the nodes while minimizing the total amount of communications. They use vertices' weights proportional to the computational work and edges proportional to communications' volume. They compute the partition using PARMETIS [3] . First result demonstrate interesting strong scaling results on small clusters.

## 3 Profiling

The FMM's behavior has been examined in terms of execution time, scalability, communications, load-balance, and data locality. To this end, two test cases have been used, as well as different profiling tools like ScoreP [4] and Maqao [2]. The test cases are a generic metallic UAV (Unmanned Aerial Vehicle) [7], with 95 524 nodes, 193 356 elements and 290 034 dofs, and a Dassault Aviation Falcon F7X airplane with 765 187 nodes, 1 530 330 elements and 2 295 495 dofs. All experiments, for profiling or results, are run on a Dassault Aviation cluster, composed by 32 nodes of two Intel Sandy Bridge E5-2670 (8 cores@2.60GHz) interconnected with InfiniBand FDR. Binaries are compiled with Intel 2015 and run with bullxmpi-1.2.9.1.

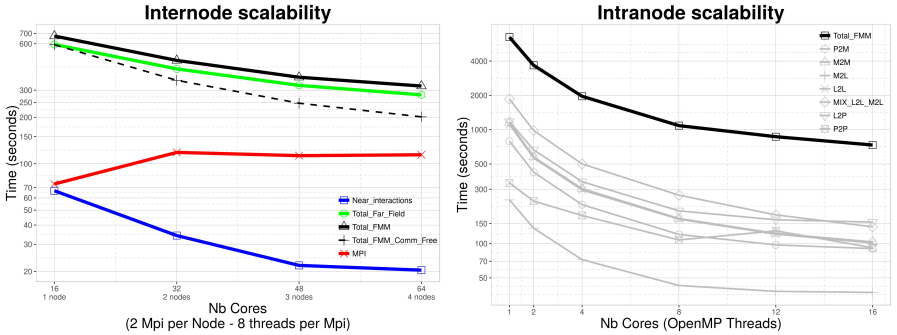


**Fig. 3.** Internode and intranode strong scaling analysis on UAV

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_47

*Scalability* In the FMM algorithm, a common difficulty is the scalability of the distributed memory, due to the large number of communications required. Therefore, we are interested in separating communication and computational costs. In order to evaluate the potential gain by overlapping the communications with computations without other changes in the algorithm, we create and measure an artificial communication-free *Total FMM Comm Free* version of the code. Of course this code does not converge numerically anymore, but the performance of a significant number of iteration can be compared with the same number of original iterations. In the current implementation, the communications consist in exchanging the vector of unknowns and far-field contributions before and after the FMM tree traversal. Unknowns are exchanged via MPI broadcast and allreduce communications, while far fields are sent via MPI two-sided point to point communications and one allreduce at the top of the tree. The left part of figure 3 shows the strong scaling analysis. We use 4 nodes, and according to the best performing execution mode of SPECTRE/MLFMM, we launch one MPI process per socket with 8 OpenMP threads. Communications are not scaling: with 8 processes, the gap (log scale) between the *Total FMM* and *Total FMM comm-free* represents 35%. Further experiments with the larger F7X test case, run on 32 nodes, with one MPI process per node, show that the time spent in the communications grows considerably and reaches 59%. The right part of figure 3 focuses on intranode scalability. It highlights the lack of shared memory parallelism: over eight threads, parallel efficiency falls under 56%. In its current status, this last measurement prevents efficient usage of current manycore architectures and is a risk, for the future increase in the number of cores, which must be addressed.

*Data Locality* Figure 4 shows the communication matrix of the original application, which reflects the quantity of data exchanged. A gradient from blue to red towards the diagonal would represent a good locality pattern. Despite communications being more important around the diagonal, vertical and horizontal lines are noticeable in the right and the bottom part of the matrix. They denote a set of processors having intensive communication with almost all the others, resulting in large connection overhead and imbalance in communication costs. Measurements with the Maqao tool expose a load balance issue between threads within an MPI worker, and between MPI workers, with respective 17% and 37% idle/waiting time.

## 4 Load Balancing

In the FMM algorithm, load-balancing consists in evenly distributing the particles among the processes, while minimizing and balancing the size and number of communications of the far-fields. There is no general and optimal solution to this problem. Each application of the FMM algorithm and more specifically each use case, may benefit from different strategies. In the case of CEM for structures, the particles are Gauss points that are not moving, and therefore one can pay

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_47

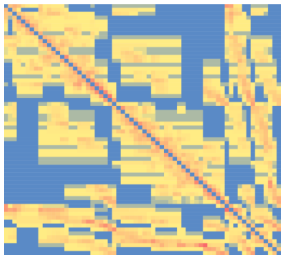

**Fig. 4.** Communication matrix, UAV test case

for off-line elaborated approaches. In our library, we propose two load balancing strategies using Morton space-filling curves and Histograms. Both methods compute separators matching the underlying octree grid. Having the separator frontier aligned on the regular octree is of prime importance for the cutoff distance test computation in the remaining of the FMM algorithm. The Morton version relies on a depth-first traversal, and isn't limited by any depth, and the Histogram version is an ameliorated ORB with multi-sections. Our implementations are generic and freely available under LGPLv3.0 [1].

### 4.1 SPECTRE's Load Balancing

In the initial version of SPECTRE, the load balancing relies on distributing an array of leaves among the processors. The array is sorted in a breadth-first way. Thus, this method results in being equivalent to a Morton ordering.

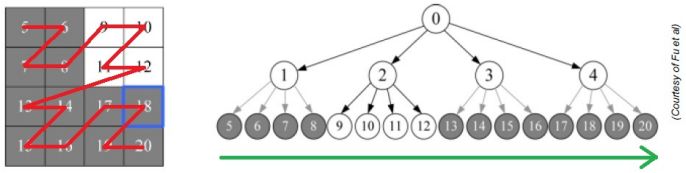


**Fig. 5.** Morton tree traversal

As shown in figure 5, drawing a Morton curve on the quadtree grid represented on the left side corresponds to picking the cells in ascending order, i.e. a depth-first tree traversal. While considering only leaves, breadth-first traversal results in the same ordering.

### 4.2 Morton

The study of the existing load balancing scheme implemented in SPECTRE has left little room for improvement using the Morton load balancing strategy. Nevertheless, Morton ordering benefits of good locality in the lower levels, but on the

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_47

highest level, the first cut can cause spatial discontinuities and therefore generate communications. In order to observe the results obtained with the different load balancing strategies, an interactive visualization tool, based on OpenGL, has been developed to produce the views. In figure 6, each process displays its own scene: the vertical plane cuts the UAV in several pieces, generating neighboring and communications.

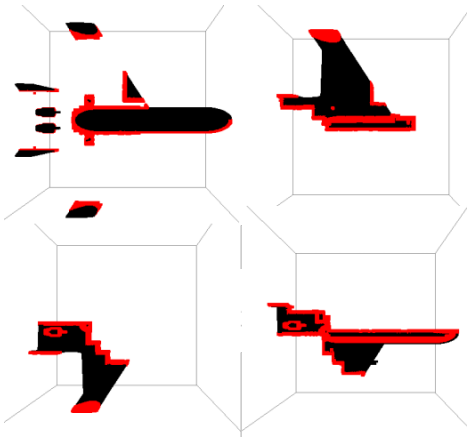

**Fig. 6.** UAV cut into discontiguous pieces with Morton ordering

Our implementation of Morton ordering uses a depth-first traversal on the underlying regular octree taking into account the number of elements present in each node. To avoid over-decomposition, a tolerance threshold has been introduced to tune the load balancing precision. While the depth-first traversal algorithm progresses down the octree, nodes become smaller, and the load balancing precision improves. The algorithm stops as soon as a separator meeting the precision threshold is computed.

In 3D, rotating the axis order results in eight different possibilities, which we have implemented and tested. Figure 7 shows the communication matrices obtained for each case. One can see that the most interesting communication matrices result from Z-axis first orderings. However, from 1 to 64 processes with our previous experimental setup, Morton ZYX ordering, obtains a limited 5% to 10% performance improvement compared with the original ordering.

### 4.3 Histograms (Multisection Kd-tree)

Multi-sections allow to overpass the ORB limit of power of two numbers of processors: the targeted number of domains is decomposed into prime numbers and the multi-sections are computed with global histograms. This is a costly and complex algorithm. This method, also referred as *implicit kd-tree*, has already been explored in other domains such as ray tracing [13]:

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_47

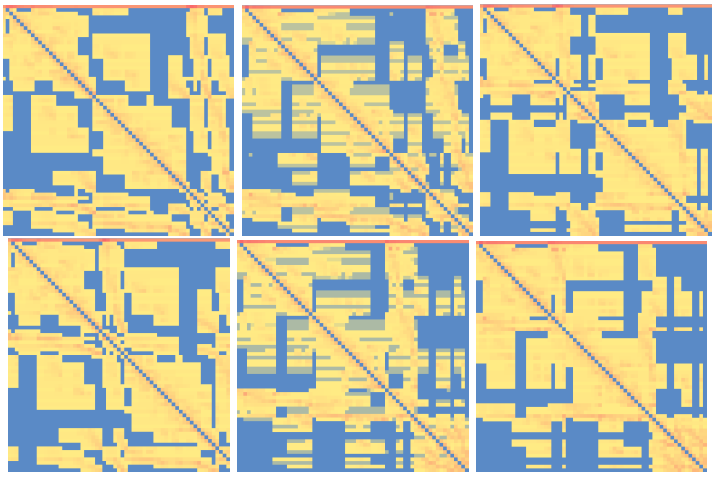

**Fig. 7.** Axis order influence on communication matrices. First column: X-axis first: XYZ and XZY - Second column: Y-axis first: YXZ and YZX, - Third column: Z-axis first : ZXY and ZYX.

We developed two scalable distributed parallel versions: the first one is a complete computation of the multi-sections leading to an exact distribution, and the second one is an approximation where the separator is rounded to match the octree grid. The second method is less accurate, but requires less computation and communication. Our implementation is fully distributed. It can handle very large inputs, which do not fit in a single node, by managing the particle exchanges, to produce the final sorted set.

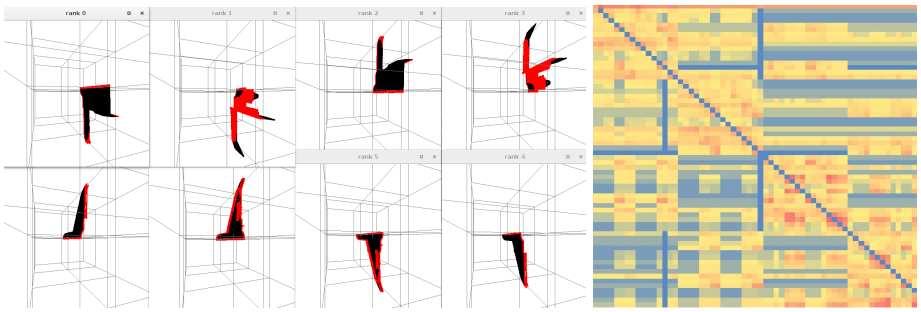

**Fig. 8.** Left : OpenGL visualization of histogram load balancing, UAV, 8 domains - Right: Communication matrix on UAV with 64 domains

Left part of figure 8 shows the UAV distributed among eight processes with red points representing the neighboring cells. One can see that they represent a large part of the points and come from the vertical cut along the Z-axis. The right

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_47

part of figure 8 shows the resulting communication matrix with 64 processes: the communication locality has been worsened, resulting in an execution time 1.7 times longer than the original version.

In electromagnetic simulations, another commonly used load balancing method consists in cutting the airplane in slices along its length [22]. The histogram algorithm allows to easily test this method by realizing all the cuts along only one axis, but the experiments did not show any improvement.

### 4.4 Load balancing future work

Load balancing the work is critical between and inside the nodes. For the work distribution among the computation nodes, we tested different classical methods without achieving any significant performance improvement. Further investigation is required. A good distribution should balance the work but also minimize the neighboring. Nonetheless, a blocking bulk-synchronous communication model induces many barriers, which could tear down any load balancing effort. Therefore, introducing asynchronism, overlapping, and fine grain task parallelism inside the nodes may influence our conclusion and must be fixed before exploring new load-balancing strategies.

## 5 Communications

The FMM's communications consist of two-sided point to point exchanges of far-field terms. In the current version, all communications are executed at the top of the octree when the upward pass is completed. The highest level of the tree is exchanged via a blocking MPI_Allreduce call. The other level communications are executed with blocking MPI_Send, MPI_Recv or MPI_Sendrecv. They are ordered and organized in rounds. At each round, pair of processes communicate and accumulate the received data into their far-field array. The computation continues only once the communication phase is completed.

We aim at proposing a completely asynchronous and multithreaded version of these exchanges. Efficient asynchronous communications consist in sending the available data as soon as possible and receiving it as late as possible. Messages are sent at the end of each level, during the upward pass, instead of waiting to reach the top of the tree. In the same way, receptions are handled at the beginning of each level during the downward pass. We have developed different versions based on non-blocking MPI and one-sided GASPI, a PGAS programing model [21]. Our early results have been presented in [20].

PGAS (Partitioned Global Address Space) is a programming model based on a global memory address space partitioned among the distributed processes. Each process owns a local part and directly accesses to the remote parts both in read and write modes. Communications are one-sided: the remote process does not participate. Data movement, memory duplications and synchronizations are reduced. Moreover, PGAS languages are well suited to clusters exploiting RDMA

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_47

(Remote Direct Memory Access) and letting the network controllers handle the communications. We use GPI, an implementation of the GASPI API [21].

The FMM-lib library entirely handles the GASPI communications outside the Fortran original code. At the initialization step, the GASPI segments are created. All necessary information to handle the communications needs to be precomputed. Indeed, when a FMM level is locally terminated, the corresponding process sends the available information by writing remotely into the recipient's memory at a pre-computed offset without requiring action from the the recipient. This write is followed by a notification containing a notifyID and a notifyValue which are used to indicate which data has been received. When the recipient needs information to pursue its computation, it checks for notifications and waits only if necessary.

A similar pattern can be built using the MPI 2-sided non-blocking communications via MPI_Isend and MPI_Irecv. These calls return immediately, and let the communication complete while the computation pursues. However, the lack of communication progression is a well-known problem. Some methods, like manual progression, help to force the completion. It consists in calling MPI_Test on the communication corresponding MPI_Request. The MPI standard ensures that the MPI_Test calls trigger the communications [14].

## 6 Communication results

We use the generic metallic UAV, place one MPI/GASPI process per node, and increase the number of nodes from 1 to 32. Each node is fully used with 16 OpenMP threads.

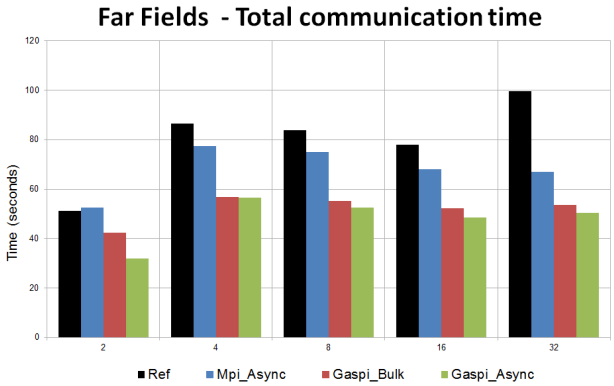


**Fig. 9.** Far fields communication time

Figure 9 shows the execution time of the different MPI and GASPI versions. We compare four different versions: *Ref*, *MPI Async*, *Gaspi Bulk*, and *Gaspi*

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_47

*Async*. The first three ones handle all the exchanges at the top of the tree. The idea is to measure the improvement, without any algorithmic modification. *Ref* uses blocking MPI calls, *MPI Async* uses non-blocking MPI calls and manual progression, and *Gaspi Bulk* uses one-sided GASPI writes. The *Gaspi Async* version sends data as soon a level is complete, receives as late as possible, and relies on hardware progression. One can see that, without introducing any overlapping, on 32 nodes, the *Gaspi Bulk* version already reaches 46% speedup over *Ref*. MPI Async version achieves 36% speedup, but still remains slower than the synchronous *Gaspi Bulk* version. Finally, introducing overlapping enables the *Gaspi Async* version to gain three more percentage points over the *Ref* version, with a total of 49% speedup on communications.

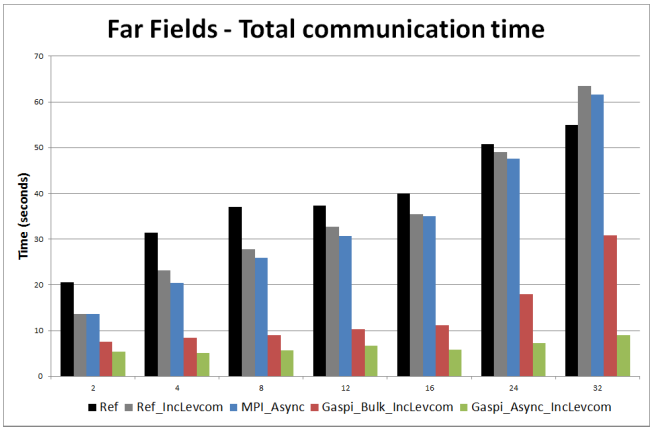


**Fig. 10.** Far fields communication time, after eliminating the Allreduce.

The allreduce at the top of the tree is very sparse. Therefore, we tried to replace it by more efficient point to point exchanges. Figure 10 shows the results on the larger F7X test case. The graph presents five versions: the reference, and all four precedent versions modified by suppressing the allreduce. All GASPI versions take more benefit more from this modification than the MPI ones. *Gaspi Async* reaches 83.5% speedup on communication over *Ref*, resulting in 29% speedup for the complete FMM algorithm.

## 7 Conclusion and Future Work

In this paper, we are investigating methodologies to introduce load balancing and asynchronous communications, in a large industrial Fortran MLFMM application. Applying the different load balancing strategies has demonstrated that it is possible to improve the communication pattern, but it would require more refined options in the future work to further improve the solution. Furthermore,

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_47

since our load balancing may be harmed by the bulk synchronous communication scheme, based on blocking MPI communications, we prioritized the implementation of a fully asynchronous communication model. We are exploring the use of non-blocking MPI and multithreaded asynchronous one-sided GASPI, and have already obtained a significant 83.5% speedup.

At the present time, we are working on optimizing the intranode scalability introducing fine grain parallelism with the use of tasks. The next step is to make the algorithm fully asynchronous to expose the maximum parallelism: all the barriers between levels will be broken into fine grain task dependencies management.

**Acknowledgements** The optimized SPECTRE application described in this article is the sole property of Dassault Aviation.

## References

1. Fmm-lib. https://github.com/EXAPARS/FMM-lib.
2. Maqao. https://www.maqao.org.
3. Parmetis. http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview.
4. Score-p. https://www.vi-hps.org/projects/score-p/.
5. Top 10 algorithm. https://archive.siam.org/pdf/news/637.pdf.
6. Workshop-em-isae. https://websites.isae-supaero.fr/workshop-em-isae-2018/workshop-em-isae-2018.
7. Workshop-em-isae. https://websites.isae-supaero.fr/workshop-em-isae-2016/accueil.
8. E. Agullo, B. Bramas, O. Coulaud, E. Darve, M. Messner, and T. Takahashi. Task-based FMM for multicore architectures. Technical Report RR-8277, INRIA, Mar. 2013.
9. E. Agullo, B. Bramas, O. Coulaud, E. Darve, M. Messner, and T. Takahashi. Task-based FMM for heterogeneous architectures. Research Report RR-8513, Inria, Apr. 2014.
10. Q. Carayol. *Development and analysis of a multilevel multipole method for electromagnetics.* PhD thesis, Paris 6, 2002.
11. A. Chandramowlishwaran, K. Madduri, and R. Vuduc. Diagnosis, tuning, and redesign for multicore performance: A case study of the fast multipole method. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pages 1–12, Washington, DC, USA, 2010. IEEE Computer Society.
12. F. A. Cruz, M. G. Knepley, and L. A. Barba. Petfmm–a dynamically load-balancing parallel fast multipole library. *CoRR*, abs/0905.2637, 2009.
13. M. Groß, C. Lojewski, M. Bertram, and H. Hagen. Fast implicit kd-trees: Accelerated isosurface ray tracing and maximum intensity projection for large scalar fields. In *Proceedings of the Ninth IASTED International Conference on Computer Graphics and Imaging*, CGIM '07, pages 67–74, Anaheim, CA, USA, 2007. ACTA Press.
14. T. Hoefler and A. Lumsdaine. Message Progression in Parallel Computing - To Thread or not to Thread? IEEE Computer Society, Oct. 2008.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_47

15. H. Ibeid, R. Yokota, and D. Keyes. A performance model for the communication in fast multipole methods on HPC platforms. *CoRR*, abs/1405.6362, 2014.
16. M. A. Jabbar, R. Yokota, and D. Keyes. Asynchronous execution of the fast multipole method using charm++. *CoRR*, abs/1405.7487, 2014.
17. I. Lashuk, A. Chandramowlishwaran, H. Langston, T.-A. Nguyen, R. Sampath, A. Shringarpure, R. Vuduc, L. Ying, D. Zorin, and G. Biros. A massively parallel adaptive fast-multipole method on heterogeneous architectures. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 58:1–58:12, New York, NY, USA, 2009. ACM.
18. H. Ltaief and R. Yokota. Data-driven execution of fast multipole methods. *CoRR*, abs/1203.0889, 2012.
19. J. Milthorpe, A. P. Rendell, and T. Huber. Pgas-fmm: Implementing a distributed fast multipole method using the x10 programming language. *CCPE*, 26(3):712–727, 2014.
20. N. Moller, E. Petit, Q. Carayol, Q. Dinh, and W. Jalby. Asynchronous One-Sided Communications for Scalable Fast Multipole Method in Electromagnetic Simulations, Aug. 2017. Short Paper presented at COLOC workshop, Euro-Par 2017, Santiago de Compostela, August 29, 2017.
21. C. Simmendinger, J. Jägersküpper, R. Machado, and C. Lojewski. A pgas-based implementation for the unstructured cfd solver tau. *PGAS11, USA*, 2011.
22. G. Sylvand. *La méthode multipôle rapide en électromagnétisme. Performances, parallélisation, applications*. Theses, Ecole des Ponts ParisTech, June 2002.
23. R. Yokota and L. A. Barba. A tuned and scalable fast multipole method as a preeminent algorithm for exascale systems. *CoRR*, abs/1106.2176, 2011.
24. R. Yokota, G. Turkiyyah, and D. Keyes. Communication complexity of the fast multipole method and its algebraic variants. *CoRR*, abs/1406.1974, 2014.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_47