Asynchronous One-Sided Communications for Scalable Fast Multipole Method in Electromagnetic Simulations

Nathalie Moller, Eric Petit, Quentin Carayol, Quang Dinh, William Jalby

Asynchronous One-Sided Communications for Scalable Fast Multipole Method in Electromagnetic Simulations

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

[1] Dassault Aviation, France
firstname.lastname@dassault-aviation.com,
[2] Intel, France
eric.petit@intel.com
[3] LI-PaRAD, University of Versailles, France
william.jalby@uvsq.fr

Abstract. A common approach in HPC applications is to use MPI and OpenMP programming models to express the parallelism. Refined solutions are using asynchronous communications to take advantage from overlapping. In this paper we propose an early implementation of the ML-FMM algorithm using GASPI asynchronous one-sided communications to demonstrate how PGAS ad task based programming can impact the code performance. Early results, on 32 nodes, show a 49% improvement on communications over the optimized MPI+OpenMP version.

Keywords: CEM MLFMM, communications, Gaspi

1 Introduction

SPECTRE is a Dassault Aviation in-house simulation code for Electromagnetic and Acoustic applications. It is intensively used for RCS (Radar Cross Section) computations. These problems can be described using the Maxwell equations. When approximating with the Method of Moments, they result in a dense linear system. Direct resolution leads to an O(N^3) complexity, where N denotes the number of unknowns. A common approach to solve larger systems is to use the Multi-Level Fast Multipole Method (MLFMM) to reduce the complexity to O($NlogN$)[1].

The MLFMM relies on an accurate approximation of pairwise interactions in the far field [2]. The rationale is to model the distant point to point interactions by hierarchically grouping the points into a single equivalent point. Therefore the MLFMM relies on the accuracy of this far field low rank approximation. To make the approximation valid, the group of point have to be far enough to the target point: this is the well-separated property. The interaction with the points that are not well separated, the near field, are done using the method of moment.

A common way to build the group is to use a hierarchical decomposition based on an octree. Then the MLFMM traverses the tree forward and backward. The upward pass aggregates the children's contributions into larger parent nodes, and the downward pass collects the contributions from the same level source cells, which involves communications, and translates the values from parents down to children. These operations are organized into MLFMM operators.

In SPECTRE, MLFMM has been implemented with a hybrid MPI + OpenMP parallelism. A common difficulty in the MLFMM algorithm is its distributed memory implementation scalability due to the large number of communications required. Furthermore, despite many existing work are optimizing the shared memory parallelization of the FMM algorithm for n-body problems, the electromagnetic simulation is introducing a major difference: at each level of the tree the work to be done to compute the multipole is doubling. This explains the O($NlogN$) complexity compared to the O(N) of the standard algorithm. The efficient parallelization within a multipole and how it integrates with the rest of the algorithm is not widely devised in the literature. An in-depth profiling of SPECTRE confirmed our analysis. Firstly, the time spent in the communications grows considerably with the number of processes and reaches 29% of total execution time with 32 processes. Secondly, the scalability in shared memory is improvable, over eight threads parallel efficiency falls under 75% for each operator. In this work, we focus on the optimization of the communications. This paper presents our preliminary results and future work in introducing asynchronous one-sided communications to improve communication overlapping and scalability, and the introduction of fine grain task parallelism for multipole computations and how it integrates with the communication layer.

2 Optimizing the Communications

For the N-body problem, the FMM communication complexity was studied in [8] and [4]. Global complexity is demonstrated to be O($(n/p)^{\alpha} + log\ p$), n being the problem size, p the number of processes and α equaling 2/3 in the 3D case. The PGAS programing model efficiency has been demonstrated in [5], [6].

The MLFMM'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 upwards pass is completed. They are ordered and organized in rounds. At each round, pair of processes communicate and accumulate the received data into their far field array. Messages are sent via blocking MPI_send, MPI_recv or MPI_sendrecv. The computation continues only once the communication phase is terminated. We aim at proposing a completely asynchronous and multithreaded version of these exchanges. A first step consists in desynchronizing the communications by sending the available data as soon as possible and receiving it as late as possible. We have developed different versions based on non-blocking MPI and one-sided GASPI.

The MPI standard offers non-blocking communications via MPI_Isend and MPI_Irecv. These calls return immediately, and let the communication complete

while computation pursues. However, the lack of communication progression is a well known problem. Some methods, like manual progression, help to force the completion[3]. Manual progression consists in calling MPI_Test on the communication's corresponding MPI_Request. The MPI standard ensures that MPI_Test calls trigger the communications.

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 (Remote Direct Memory Access) and letting the network controllers handle the communications. We use GPI, a GASPI API, which implements the PGAS standard, developed by Fraunhofer ITWM [7].

The FMM-lib library, entirely handles the GASPI communications. It is called by the Fortran application, which provides pointers to the structures containing the data to exchange. At the initialization step, the GASPI segments are created and, since the communications are one-sided, some first exchanges are required. All necessary informations to handle the communications are pre-computed. When a level is terminated, the source process sends the available information by writing remotely into the receiver's memory at a pre-computed offset. This write is followed by a notification containing a `notifyID` and a `notifyValue` which are used to indicate the source's rank and the octree level. When the receiver needs the information to poursue its computation, it checks for the expected notifications and waits only if necessary. The library is available under LGPL licence at https://github.com/EXAPARS/FMM-lib.

3 Preliminary Results

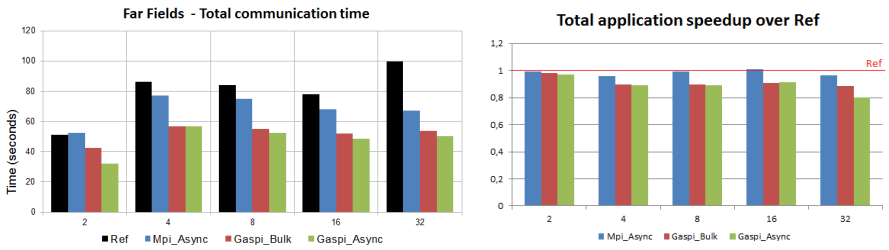

Fig. 1. Left: Time spent in far field communications, Right: Total speedup over Ref

Figure 1 shows the execution time of the different MPI and GASPI and the gain on the total application. The experiences are run on an in-house cluster at

Dassault Aviation, composed by 32 nodes of two Intel Sandy Bridge E5-2670 (8 cores@2.60GHz). SPECTRE and the FMM library are compiled with Intel 2015 and run with bullxmpi-1.2.9.1. The test case is a generic metallic UAV, with 95 524 nodes, 193 356 elements and 290 034 dofs. We place one MPI/GASPI process per node and increase the nodes count from 1 to 32. Each node is fully used with OpenMP threads.

We compare four different versions : `Ref`, `MPI Async` and `Gaspi Bulk` handle all the exchanges at the top of the tree. `Ref` uses blocking MPI calls, `MPI Async` uses non blocking MPI calls and manual progression and `Gaspi Bulk` one-sided Gaspi writes. `Gaspi Overlap` sends 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 Overlap` version to gain 3 more percentage points over the `Ref` version, with a total of 49% speedup on communications and 20% on the complete application.

4 Conclusion and Future Work

In this article we investigate a way of desynchronizing an industrial Fortran MLFMM kernel based on blocking MPI communications. The communication phase has been optimized with the use of non blocking MPI and one-sided Gaspi and a significant speedup has been obtained. At the present time, we are working on optimizing the intranode scalability with the use of tasks. The communications will be multithreaded and be integrated into the tasks.

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

References

1. Carayol, Q.: Development and analysis of a multilevel multipole method for electromagnetics. Ph.D. thesis, Paris 6 (2002)
2. Greengard, L., Rokhlin, V.: A fast algorithm for particle simulations. J. Comput. Phys. 73(2), 325–348 (Dec 1987), http://dx.doi.org/10.1016/0021-9991(87)90140-9
3. Hoefler, T., Lumsdaine, A.: Message Progression in Parallel Computing - To Thread or not to Thread? IEEE Computer Society (Oct 2008)
4. Ibeid, H., Yokota, R., Keyes, D.: A performance model for the communication in fast multipole methods on HPC platforms. CoRR abs/1405.6362 (2014)
5. Jabbar, M.A., Yokota, R., Keyes, D.: Asynchronous execution of the fast multipole method using charm++. CoRR abs/1405.7487 (2014)
6. Milthorpe, J., Rendell, A.P., Huber, T.: Pgas-fmm: Implementing a distributed fast multipole method using the x10 programming language. CCPE 26(3), 712–727 (2014)
7. Simmendinger, C., Jägersküpper, J., Machado, R., Lojewski, C.: A pgas-based implementation for the unstructured cfd solver tau. PGAS11, USA (2011)
8. Yokota, R., Turkiyyah, G., Keyes, D.: Communication complexity of the fast multipole method and its algebraic variants. CoRR abs/1406.1974 (2014)