Fast Multipole Method for particle interactions: an open source parallel library component.

Felipe A. Cruz[a*], L. A. Barba[a†] and Matthew G. Knepley[b]

[a]Department of Mathematics, University of Bristol,
University Walk, Bristol, BS8 1TW, United Kingdom

[b]Mathematics and Computer Science Division, Argonne National Laboratory,
9700 S. Cass Avenue, Argonne, IL 60439 U.S.A.

The fast multipole method is used in many scientific computing applications such as astrophysics, fluid dynamics, electrostatics and others. It is capable of greatly accelerating calculations involving pair-wise interactions, but one impediment to a more widespread use is the algorithmic complexity and programming effort required to implement this method. We are developing an open source, parallel implementation of the fast multipole method, to be made available as a component of the PETSc library. In this process, we also contribute to the understanding of how the accuracy of the multipole approximation depends on the parameter choices available to the user. Moreover, the parallelization strategy provides optimizations for automatic data decomposition and load balancing.

1. INTRODUCTION

The advantages of the fast multipole method (FMM) for accelerating pair-wise interaction or N-body problems are well-known. In theory, one can reduce an $\mathcal{O}(N^2)$ calculation to $\mathcal{O}(N)$, which has a huge impact in simulations using particle methods. Considering this impact, it is perhaps surprising that the adoption of the FMM algorithm has not been more widespread. There are two main reasons for its seemingly slow adoption; first, the scaling of the FMM can really only be achieved for simulations involving very large numbers of particles, say, larger than 10^4. So only those researchers interested in solving large problems, and with access to moderately large computing resources, will see an advantage with the method. More importantly, perhaps, is the fact that the FMM requires considerable extra programming effort, when compared with other algorithms like particle-mesh methods, or treecodes providing $\mathcal{O}(N \log N)$ complexity.

One could argue that a similar concern has been experienced in relation to most advanced algorithms. For example, when faced with a problem resulting in a large system of algebraic equations to be solved, most scientists would be hard pressed to have to program a modern iterative solution method, such as a generalized minimum residual (GMRES) method. Their choice, in the face of programming from scratch, will most likely be direct Gaussian elimination, or if attempting an iterative method, the simplest to implement but slow to converge Jacobi method. Fortunately, there is no need to make this choice, as we nowadays have available a

*Acknowledges support from: SCAT project via EuropeAid contract II-0537-FC-FA, `www.scat-alfa.eu`, and BAE/Airbus under Agreement ACAD 01478

†Acknowledges support from EPSRC under grant contract EP/E033083/1.

wealth of libraries for solving linear systems with a variety of advanced methods. What's more, there are available parallel implementations of many libraries, for solving large problems in a distributed computational resource. One of these tools is the PETSc library for large-scale scientific computing [1]. This library has been under development for more than 15 years and offers distributed arrays, and parallel vector and matrix operations, as well as a complete suite of solvers, and much more. We propose that a parallel implementation of the FMM, provided as a library component in PETSc, is a welcome contribution to the diverse scientific applications that will benefit from the acceleration of this algorithm.

In this paper, we present an ongoing project which is developing such a library component, offering an open source, parallel FMM implementation, which furthermore will be supported and maintained via the PETSc project. In the development of this software component, we have also investigated the features of the FMM approximation, to offer a deeper understanding of how the accuracy depends on the parameters. Moreover, the parallelization strategy involves an optimization approach for data decomposition among processors and load balancing, that should make this a very useful library for computational scientists.

2. CHARACTERIZATION OF THE MULTIPOLE APPROXIMATION

2.1. Overview of the algorithm

The FMM is based on the idea that the influence of a cluster of particles can be approximated by an agglomerated quantity, when such influence is evaluated far enough away from the cluster itself. The method works by dividing the computational domain into a *near-domain* and a *far-domain*:

Near domain: contains all the particles that are near the evaluation point, and is usually a minor fraction of all the N particles. The influence of the near-domain is computed by directly evaluating the pair-wise particle interactions. The computational cost of directly evaluating the near domain is not dominant as the near-domain remains small.

Far domain: contains all the particles that are far away from the evaluation point, and ideally contains most of the N particles of the domain. The evaluation of the far domain will be sped-up by evaluating the approximated influence of clusters of particles rather than computing the interaction with every particle of the system.

The approximation of the influence of a cluster is represented as Multipole Expansions (MEs) and as Local Expansions (LEs), these two different representations of the cluster are the key ideas behind the FMM. The MEs and LEs are Taylor series that converge in different subdomains of space. The center of the series for an ME is the center of the cluster of source particles, and it only converges outside the cluster of particles. In the case of an LE, the series is centered near an evaluation point and converges locally.

The first step of the FMM is to hierarchically subdivide space in order to form the clusters of particles; this is accomplished by using a tree structure, illustrated in Figure 1, to represent each subdivision. In a one-dimensional example: level 0 is the whole domain, which is split in two halves at level 1, and so on up to level l. In two dimensions, instead each domain is divided in four, to obtain a quadtree, while in three dimensions, domains are split in 8 to obtain an oct-tree. In all cases, we can make a flat drawing of the tree as in Fig. 1, with the only difference being the number of branches coming out of each node.

The next step is to build the MEs for each node of the tree; the MEs are built first at the deepest level, level l, and then translated to the center of the parent cell. This is referred to as the *upward sweep* of the tree. Then, in the *downward sweep* the MEs are first translated into

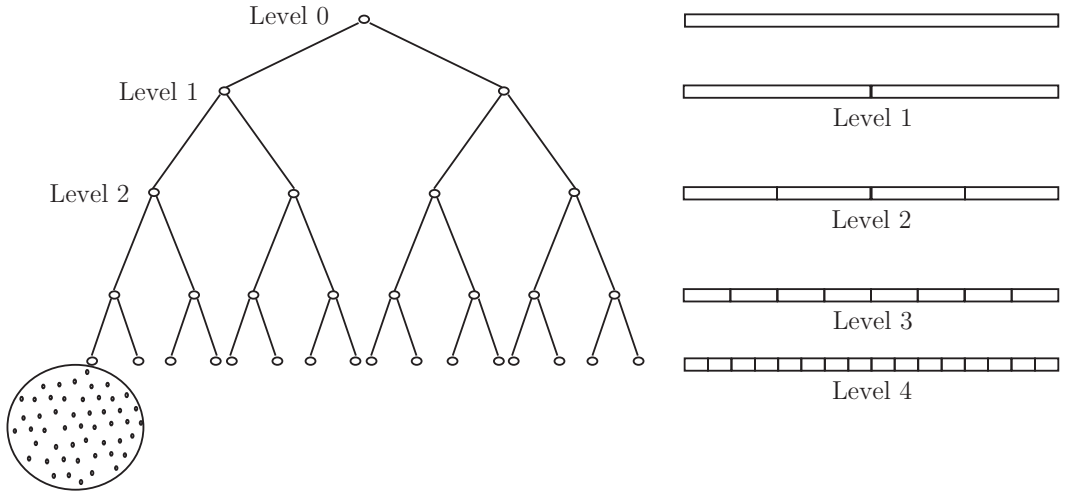


Figure 1. Sketch of a one-dimensional domain (right), divided hierarchically using a binary tree (left), to illustrate the meaning of levels in a tree and the idea of a final leaf holding a set of particles at the deepest level.

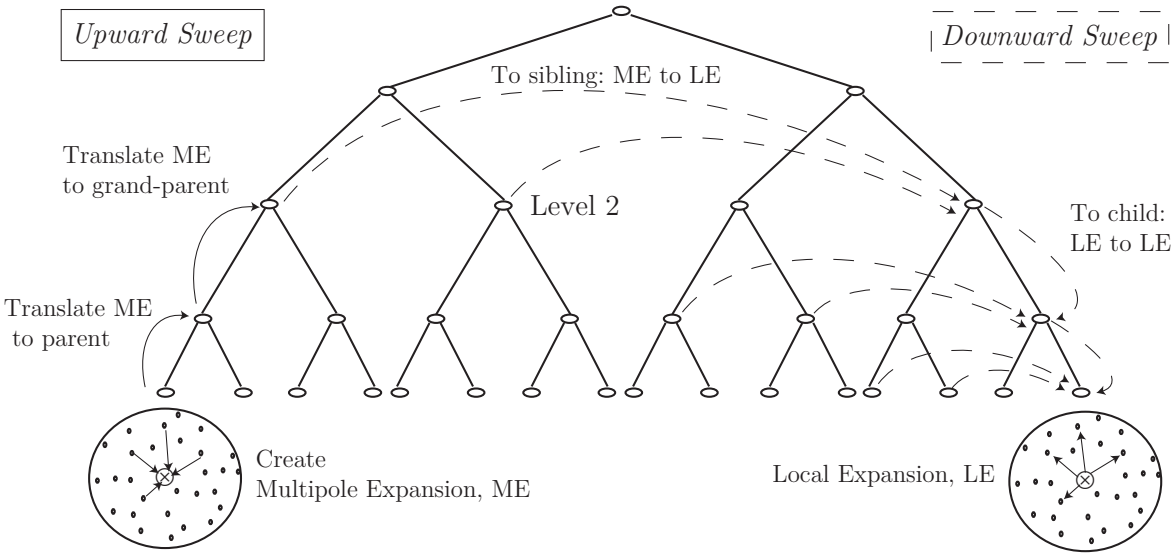
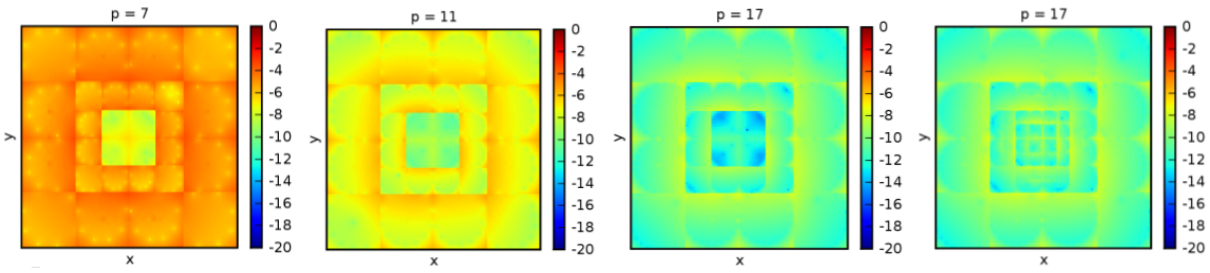


Figure 2. Illustration of the *upward sweep* and the *downward sweep* of the tree. The multipole expansions (ME) are created at the deepest level, then translated upwards to the center of the parent cells. The MEs are then translated to a local expansion (LE) for the siblings at all levels deeper than level 2, and then translated downward to children cells. Finally, the LEs are created at the deepest levels.

LEs for all the boxes in the *interaction list*. At each level, the interaction list corresponds to the cells of the same level that are in the far field for a given cell. Finally, the LEs of upper levels are added up to obtain the complete far domain influence for each box at the leaf level of the tree. These ideas are better visualized with an illustration, as provided in Figures 2. For more details of the algorithm, we cite [2].

2.2. Results with the serial version of the FMM code

We have developed a prototype of the FMM algorithm in a Python code, which has been used to produce a methodical study of the approximation error, with respect to the available parameters: p, the truncation level of the multipole expansion; l, the deepest level of the tree; and N, the number of particles. The application of the FMM that was chosen was the calculation of the velocity of N particles of vorticity, using the Biot-Savart law of vorticity dynamics. The results of more than 900 runs with this code are being compiled for another publication. We present here an example which shows the variation of error in space for different truncation p and an additional level of the tree. See Figure 3.



(a) $l = 3$, $p = 7$ (b) $l = 3$, $p = 11$ (c) $l = 3$, $p = 17$ (d) $l = 4$, $p = 17$

Figure 3. Logarithm of the error incurred when using FMM in experiments with $N = 10201$ in a velocity evaluation with vortex particles.

3. PARALLELIZATION STRATEGY

In order to partition work among processes, we cut the multipole tree at a certain level k, dividing it into a *root* tree and 2^{dk} *local* trees. By assigning multiple trees to any given process, we can achieve both load balance and minimize communication. In order to partition the trees, we assemble a graph whose vertices are the sub-trees with edges (i, j) indicating that a cell c in sub-tree j is in the interaction list of cell c' in sub-tree i. Then weights are assigned to each vertex i, indicating the amount of computational work performed by the sub-tree i, and to each edge (i, j) indicating the communication cost between sub-trees i and j. Now that we have a weighted graph representation of the fast algorithm, the graph can be partitioned, for instance using ParMetis [3], and the tree information can now be distributed to the relevant processes. Advantages of this approach, over a space-filling curve partitioning for example, include its simplicity, reuse of the serial tree data structure, and reuse of existing partitioning tools. Moreover, purely local data and data communicated from other processes are handled using the same sieve section structure. Use of the sieve framework reduces the parallelism to a single operation for neighbor and interaction list exchange.

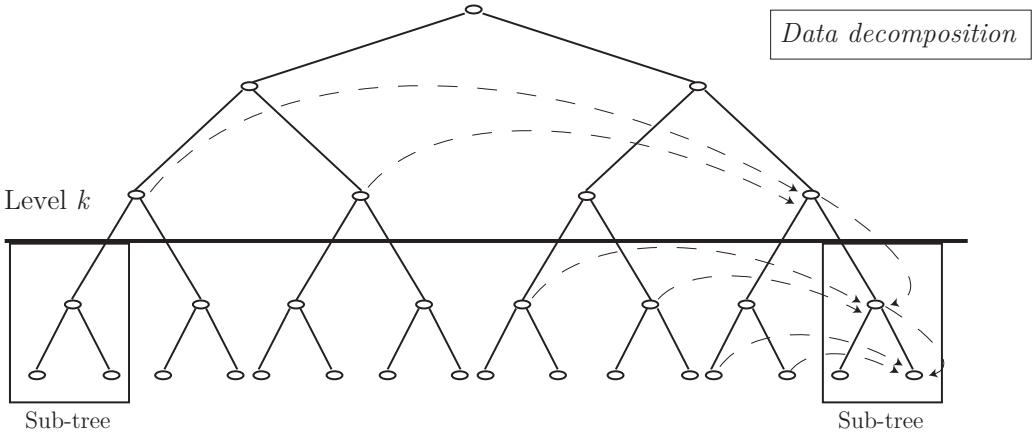


Figure 4. Sketch to illustrate the parallelization strategy. The tree is "cut" at a chosen level k, and all sub-trees branching from that level are assigned to processors.

4. CONCLUDING REMARKS

At this point, we have implemented the parallel version of the FMM code and it is being integrated to the PETSc library. Experiments are being performed for small numbers of processors, and initial results agree with serial runs. Optimization of the parallel code is underway. By this, we mean incorporating the use of the graph partitioner with an intelligent weighting of the work to be done at each node of the tree, and automatic optimal load balancing. Further optimizations are under investigation. For the Parallel CFD conference, we are planning to have speed-up results and to demonstrate the optimization strategy. PETSc integration will reach beta stage at some point later in the year.

REFERENCES

1. S. Balay, K. Buschelman, W. D. Gropp, D. Kaushik, M. Knepley, L. Curfman-McInnes, B. F. Smith, and H. Zhang. PETSc User's Manual. Technical Report ANL-95/11 - Revision 2.1.5, Argonne National Laboratory, 2002.
2. L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *J. Comput. Phys.*, 73:325–348, 1987.
3. George Karypis and Vipin Kumar. A parallel algorithm for multilevel graph partitioning and sparse matrix ordering. *Journal of Parallel and Distributed Computing*, 48:71–85, 1998.