Massively Parallel Fast Multipole Method Solutions of Large Electromagnetic Scattering Problems

Caleb Waltz, Student Member, IEEE Kubilay Sertel, Member, IEEE Michael A. Carr, Member, IEEE
Brian C. Usner, Member, IEEE and John L. Volakis, Fellow, IEEE

Abstract—We describe a massively parallel version of the single-level fast multipole method that employs the fast Fourier transform for the translation stage. The proposed FMM-FFT method alleviates the communication bottleneck and has a lower complexity, $O(N^{3/2} \log^{2/3} N)$, as compared to the conventional single level fast multipole method which scales as $O(N^{3/2})$, where $N$ is the number of unknowns. Through numerical examples we demonstrate that the proposed parallel fast multipole method yields a faster solution time than its multilevel counterpart for very large problems in a distributed memory parallel setting.

Index Terms—fast methods, FMM, FFT, parallel, scalable, multilevel

I. INTRODUCTION

Massively scalable distributed computing on relatively inexpensive computer clusters and multi-core processors continues to be the method of choice for solving modern engineering and science problems arising from extremely complicated real-life applications. The last decade witnessed significant developments in both computer hardware capabilities and implementations of fast and efficient algorithms. Concurrent advances in these two areas have enabled faster, more efficient and accurate numerical solutions for a vast range of problems in all areas of pure and applied sciences. As a result (of these algorithmic breakthroughs and hardware developments), computational electromagnetics (CEM) commanded growing attention within the electromagnetics community. The last several years have concurrently witnessed an unprecedented surge in the development of computer codes, some of which have been used for automated designs of complex EM problems. In turn, complicated design requirements are fueling the need to solve larger and more complicated problems. Inevitably, the complexities and electrical sizes of CEM problems motivate the development of codes for massively parallel distributed computers.

The obvious bottleneck with many algorithms employed for the solution of integral equations relates to the $O(N^3)$ solution time and $O(N^2)$ memory storage requirements of direct solution. Iterative solvers [1]–[3] reduce the solution time down to $O(N_{it}N^2)$, with $N_{it}$ being the number of iterations to achieve a desired accuracy. To further reduce the overall time and memory requirements, fast algorithms such as the conjugate gradient fast Fourier transform (CG-FFT), adaptive integral method (AIM), and fast multipole method (FMM), can be employed [4]–[12]. These algorithms reduce the CPU time for carrying out the matrix-vector product (MVP) in the iterative loop from $O(N^2)$ down to $O(N^{3/2})$ and $O(N \log N)$. The FMM, arguably the most popular of these methods, was further improved via the multilevel fast multipole algorithm (ML-FMA), which boasts an $O(N \log N)$ MVP time [13], [14]. While substantially more difficult to implement, the ML-FMA has become the algorithm of choice when solving large-scale electromagnetics scattering problems. Therefore, the parallelization of ML-FMA on distributed memory computing clusters is of great interest.

Parallelization of ML-FMA has been considered by several researchers on both distributed memory computing clusters [15]–[24] and shared memory multiprocessors [25] (Ch. 4 of [17] describes the parallelization of ML-FMA and includes more references on the subject). Here, we concern ourselves with only distributed memory parallel algorithms as this platform is generally less expensive and more readily available (even though shared memory architectures are preferred for parallelization). While some success in parallelizing ML-FMA in distributed memory environments has been demonstrated (see e.g. [17], [22]–[24]), it requires sophisticated load distribution strategies involving shared and distributed partitions of the ML-FMA tree. Thus, parallel scaling of the ML-FMA algorithm has been limited to a handful of processors. On the other hand, the single-level FMM (since it does not employ the tree structure of ML-FMA) does not suffer from poor parallel scaling. Consequently, it is a more eligible candidate for parallelization despite its higher $O(N^{3/2})$ complexity. An FFT extension of the conventional FMM, known as FMM-FFT [26], dramatically lowers the $O(N^{3/2})$ MVP time requirement of the conventional algorithm down to below $O(N^{4/3} \log^{2/3} N)$, while preserving the propensity for parallel scaling of the single level FMM. In this paper, we demonstrate that a parallel FMM-FFT algorithm is quite attractive (when compared to the ML-FMA) in the context of massively parallel distributed memory machines.

We first show how the FMM-FFT dramatically lowers the complexity of the conventional FMM with respect to CPU time for a single MVP. We then show results of our attempts to parallelize the ML-FMA. Finally, a comparison between the parallel FMM-FFT and ML-FMA is given along with some RCS results of the parallel FMM-FFT.

II. COMPUTATIONAL COMPLEXITY OF THE FMM-FFT

The FMM-FFT method, originally proposed in [26], was shown to scale as $O(N \log N)$ in the context of geometries...
of small height (i.e. almost planar geometries) by exploiting the Toeplitz property of the regular group spacing of the FMM to reduce the complexity of the translation stage from $O(KM^2)$ down to $O(KQ \log Q)$, where $M$ is the number of non-empty groups, $Q$ is the total number of groups, and $K$ is a constant representing the number of $k$-space samples. This distinction between $Q$ and $M$ in the context of three-dimensional surface codes is the reason that the FMM-FFT defining the pertinent parameters: FFT algorithm in a three-dimensional context, we begin by choosing the number of groups in a manner that achieves improvement in complexity over the conventional FMM. This, combined with its propensity for scalable implementation, makes it a prominent candidate for solving very large problems in a scalable environment.

In the conventional FMM algorithm, care must be taken to choose the number of groups in a manner that achieves $O(N^{3/2})$ complexity. To analyze the complexity of the FMM-FFT algorithm in a three-dimensional context, we begin by defining the pertinent parameters:

- $K =$ number of $k$-space samples,
- $N =$ number of unknowns in the problem,
- $M =$ number of non-empty FMM groups,
- $Q =$ total number of FMM groups,
- $D =$ maximum group dimension, assuming groups are perfect cubes.

As mentioned earlier, the distinction between $Q$ and $M$ is very important in the context of three-dimensional surface formulations. This stems from the fact that $M$ increases proportionally to $N$ in both surface and volume codes, whereas $Q$ increases in proportion to $N^{3/2}$ in surface codes and to $N$ in volume codes. Table I summarizes the proportionality relationships of $M$ and $Q$ versus $N$ and $D$.

<table>
<thead>
<tr>
<th>Surface Codes</th>
<th>Volume Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M \propto \frac{N}{M}$</td>
<td>$M \propto Q \propto \frac{N}{M}$</td>
</tr>
<tr>
<td>$Q \propto \frac{N}{M}$</td>
<td>$Q \propto M^{3/2}$</td>
</tr>
</tbody>
</table>

**TABLE I**  
PROPORTIONALITY OF NON-EMPTY GROUPS USED ($M$) AND TOTAL GROUPS ($Q$) WITH RELATION TO unknowns ($N$) AND GROUP SIZE ($D$) FOR BOTH SURFACE AND VOLUME IMPLEMENTATIONS.

In the conventional FMM, the computational cost of aggregations/disaggregations is proportional to $KN$, and the cost of translations is proportional to $KM^2$. Therefore, to optimize the complexity, the choice of $M \propto \sqrt{N}$ is usually made to arrive at a final cost that is proportional to $KN$. Since $K$ is proportional to $D^2$, which is in turn proportional to $N/M$, the final order of complexity becomes $O(N^{3/2})$.

Likewise, in the context of FMM-FFT, the cost of aggregations/disaggregations remains proportional to $KN$. However, the cost of translations becomes proportional to $KQ \log Q$. If we choose $Q \log Q \propto N$, then the final cost is proportional to $KN$. Further, since $K \propto D^2$, and $D^2 \propto \frac{N}{Q^{2/3}}$ (see Table I), the overall FMM-FFT complexity is proportional to $\frac{N^2}{Q^{2/3}} Q$ or $N^{4/3} \log^{2/3} Q$. Since $Q \log Q \propto N$ is a transcendental relationship, we cannot express $Q$ explicitly in terms of $N$. However, if we choose $Q \log Q = CN$, where $C$ is a constant less than unity, then

$$\log Q = \log C + \log N - \log \log Q < \log N.$$  

Using this, we can conclude that the final complexity of FMM-FFT is bounded by $O(N^{4/3} \log^{2/3} N)$. For a three-dimensional volume formulation the corresponding complexity would be $O(N \log N)$, since $Q \propto M$ in three-dimensional volume discretizations (see Table I).

Fig. 1 demonstrates the performance improvement of the FMM-FFT algorithm over the conventional FMM. As seen, the FMM-FFT performs much better than the conventional FMM for problems having more than 1,000 unknowns. Curves depicting complexity orders are also included for reference.

![Fig. 1. Comparison between MVP CPU times for FMM-FFT and the conventional FMM referring to the discretization of an ogive geometry. Simulations were performed using an AMD Opteron at 1.8GHz.](image)

To further illustrate, Tables II and III give the maximum group dimension ($D$) when considering the ogive geometry. The same geometry was also used to generate Fig. 1. The corresponding MVP times of Fig. 1 are shown in the rightmost columns. We remark that the group sizes for the FMM-FFT are substantially smaller than those for the conventional FMM since the translation stage is performed more quickly in the former, meaning that we can have more groups that are smaller. However, performing translations with FFT is much faster and this is despite the large number of empty groups. From Table III we observe that when the 70,000 unknown problem (ogive geometry) is divided into groups having a maximum dimension of $kD = 2.155$, only 16% of the groups are non-empty. Nevertheless, despite this apparent inefficiency, the $O(Q \log Q)$ complexity of the FFT is more efficient than the $O(M^2)$ complexity of the conventional FMM translation stage.

We proceed to demonstrate that the actual complexity is somewhat better than $O(N^{4/3} \log^{2/3} N)$ if we carefully choose the group size to optimize memory requirements. To
do this, let us assume the following memory requirements for the FMM-FFT,
- Near terms – \( O(N^2/M) \)
- Signature terms – \( O(KN) \)
- Translation operator – \( O(KQ) \).

where (as usual) \( N \) is the number of unknowns, \( K \) is the number of \( k \)-space sample points, \( M \) is the number of non-empty FMM groups, and \( Q \) denotes the total number of groups. We can combine these memory requirements with the relationships from Table I to obtain the total memory requirements as a function of group size \( D \).

\[
MEM_{tot} = C_{near}D^2N + C_{sig}D^2N + C_{trans}N^{3/2} \frac{M}{D}. \tag{3}
\]

Since the coefficients \( C_{near}, C_{sig}, \) and \( C_{trans} \) are constant with respect to the cell size \( D \), we can estimate their values and minimize \( MEM_{tot} \) with respect to \( D \) using a simple iterative scheme such as the Newton method. Furthermore, since these memory multipliers are also constant with respect to problem size or \( N \) for a given geometry, we can extrapolate to any problem size. This is done by approximating the memory multipliers up to a feasible problem size and minimizing (3) for arbitrarily large \( N \). While we showed that the complexity of FMM-FFT is bounded by \( O(N^{4/3}\log^{2/3}N) \), our observations for reasonably large problems displayed a significantly lower complexity.

Fig. 2 shows the results of a study where we present the scaling of the memory requirements for several discretizations of the ogive geometry. As seen, the memory requirement of the FMM-FFT scale much better than the upper bound of \( O(N^{4/3}\log^{2/3}N) \). We should remark, however, that while

![Fig. 2. Theoretical memory requirements of an ogive geometry using FMM-FFT. The middle curve is the actual memory requirement. It is bounded by the \( O(N^{6/3}) \) and \( O(N^{3/3}\log^{2/3}N) \) curves.](image)

### III. Performance of Parallel ML-FMA

The ML-FMA is very effective in reducing the complexity of the fast multipole method down to \( O(N\log N) \). Its overall complexity is reduced by decreasing the complexity of the translation stage of the FMM MVP from \( O(KM^2) \) down to \( O(KM\log M) \) via a nested tree structure algorithm, where \( K \) is the number of \( k \)-space samples and \( M \) is the number of non-empty groups. However, this tree structure leads to significant inter-processor communication in a distributed memory parallel setting. Thus, parallelization of ML-FMA is a challenge. To assess the parallel scaling of parallel ML-FMA, we developed a cell-based load distribution approach. Fig. 3 shows an example partitioning of such a tree across four processors. Note that this is a very general, arbitrary grouping scenario. The actual CPU cost, sometimes referred to as the “mass” of the branches, is determined by several factors, including the number of basis functions in each group, the non empty children from each “parent” and so on. A good load balancing requires an optimum partitioning of the tree leading to equal division of the computational cost among processors. However, even good load balancing may not lead to best CPU performance since inter-processor communication will inevitably be a controlling factor. 

Unfortunately, previous approaches yielded poor scaling performance, as evidenced by the data shown in
Fig. 3. Example processor assignments of the cells for the ML-FMA tree.

Fig. 4 [18] (also see [19]). While more elaborate implementations of the ML-FMA, such as those given in [15], [17], [22]–[24] have been developed, efficient scaling was demonstrated for only a handful of processors. The complexity of implementing such sophisticated load distribution schemes suggests that the simpler FMM-FFT approach outlined in this paper may be an attractive alternative for very large-scale implementations of massively parallel integral equation solvers.

IV. PERFORMANCE OF MASSIVELY PARALLEL FMM-FFT

In the conventional FMM and modified FMM-FFT, samples in $k$-space are completely independent of one another. Therefore, by distributing the $k$-space samples across processors, the only need for communication is at the very end of the matrix-vector product when the signatures over $k$-space need to be numerically integrated. This efficient distribution was suggested in [24] for the shared levels of the ScaleME implementation. In this paper, we integrate this approach into the FFT accelerated FMM. Implementation of this parallelization is very simple and allows the parallel implementation of FMM-FFT to scale well. Using this approach, Fig. 5 shows the speedup of sparse matrix fill times (top) and MVP times (bottom) and Fig. 6 shows the corresponding parallel efficiency when solving for the radar cross section (RCS) of a 513,274 unknown PEC sphere using FMM-FFT. As we can see, the MVP time efficiency for a 513,274 sphere surpasses the 90% mark for 64 processors and over 80% efficiency is obtained for 128 processors. Fig. 7 shows the MVP times for each of the 128 processors for the parallel FMM-FFT in solving the same 513,274 unknown sphere problem. It is seen that the computational load is almost perfectly balanced across such a large number of processors, providing the excellent parallel scalability demonstrated in Figs. 5 and 6. Also, Fig. 8 shows a similar speedup plot for an ogive with 247,972 unknowns.

It is clear from Figs. 5 and 8 that the FMM-FFT scales better than all reported ML-FMA implementations [15]–[24] for massively distributed computers. This excellent scaling, combined with a low complexity order of $O(N^{4/3} \log^{2/3} N)$, makes the FMM-FFT algorithm an attractive choice when solving large problems using integral equation methods in a parallel setting.

Fig. 9 shows how the parallel FMM-FFT algorithm compares against the serial ML-FMA in terms of CPU time. For
curves in Fig. 9, assumed $O(N^{6/5})$ complexity for FMM-FFT, and $O(N \log N)$ complexity for ML-FMA. The timings were extrapolated up to 20 million unknowns for up to 64 processors. As seen in Figs. 9 and 10, for a large number of processors, the FMM-FFT is faster and uses less memory than the serial ML-FMA. It is important to note here that a parallel implementation of ML-FMA will provide excellent parallel scaling of memory requirements. However, since the time requirements do not scale as well, the ML-FMA algorithm is not attractive for such a large number of unknowns.

Referring to Fig. 9, we note that FMM-FFT performs a MVP with 5 million unknowns on 64 processors in approximately the same time as the serial ML-FMA with 100,000 unknowns and the 8x ML-FMA with 1 million unknowns. Tables IV and V give the actual MVP times and memory requirements shown in Figs. 9 and 10.

<table>
<thead>
<tr>
<th>Unknowns</th>
<th>Serial ML-FMA</th>
<th>FMM-FFT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>300</td>
<td>0.120</td>
<td>0.070</td>
</tr>
<tr>
<td>2120</td>
<td>0.970</td>
<td>0.410</td>
</tr>
<tr>
<td>26996</td>
<td>23.210</td>
<td>11.770</td>
</tr>
<tr>
<td>247972</td>
<td>55.411</td>
<td>11.940</td>
</tr>
</tbody>
</table>

**TABLE IV**

MVP TIMES (SECONDS) FOR PEC SPHERES USING A 2.4 GHZ INTEL P4 PROCESSORS.

As an example, Fig. 11 shows the RCS for a 141,558 unknown ogive solved using FMM-FFT on 4 processors. The iterative solution required 93 iterations using a BI-CGSTAB solver [3]. The total solve time was 68 minutes, yielding an average MVP time of 22 seconds. The corresponding memory was less than 600 MBytes per processor.

Fig. 12 shows the RCS for a 1,138,368 unknown sphere of...
diameter 40\(\lambda\) calculated using FMM-FFT. This calculation was performed using 32 processors (2.4GHz Intel P4 Xeon). The iterative solution of this system required 711 iterations using BI-CGSTAB (which requires two MVPs per iteration). The total solve time was 15.26 hours, yielding an average MVP time of 38.65 seconds. Memory requirements were close to 1 GB per processor.

V. CONCLUSIONS

We demonstrated that the FMM-FFT algorithm can be optimized to scale lower than \(O(N^{4/3} \log^{2/3} N)\) in both memory and CPU time by carefully choosing the group size \(D\). While algorithmically not as efficient as the \(O(N \log N)\) complexity of the ML-FMA, it has the advantage of much better scaling for massively parallel distributed memory implementations. In contrast, the parallelization of ML-FMA is rather difficult, requiring significant experience in parallel programming. Given the availability of a large number of processors at our disposal, we conjecture that a FMM-FFT algorithm is a viable alternative to previously reported ML-FMA implementations.

REFERENCES


Fig. 11. Bistatic RCS for a 40\(\lambda\) long and 8\(\lambda\) thick PEC ogive using the parallel FMM-FFT with 140,000 unknowns excited by a plane wave traveling in the +z direction; shown is the \(\phi = 0^\circ\) cut (top), and the \(\phi = 90^\circ\) cut (bottom).

Fig. 12. Bistatic RCS for a 40\(\lambda\) diameter PEC sphere using a parallel FMM-FFT with 1,138,368 unknowns excited by a plane wave traveling in the +z direction; shown is the \(\phi = 0^\circ\) cut (top), and the \(\phi = 90^\circ\) cut (bottom).


