High Performance Evaluation of Helmholtz Potentials Using the Multi-Level Fast Multipole Algorithm

Michael P. Lingg, Stephen M. Hughey, Balasubramaniam Shanker, *Fellow, IEEE*, and Hasan Metin Aktulga

Abstract—Evaluation of pair potentials is critical in a number of areas of physics. The classical N-body problem has its root in evaluating the Laplace potential, and has spawned tree-algorithms, the fast multipole method (FMM), as well as kernel independent approaches. Over the years, FMM for Laplace potential has had a profound impact on a number of disciplines as it has been possible to develop highly scalable parallel versions of these algorithms. This is in stark contrast to parallel algorithms for oscillatory potentials such as the Helmholtz potential. The principal bottlenecks to scalable parallelism are the computation and communication costs of operations necessary to traverse up, across, and down the tree. In this article, we analyze asymptotic costs for both computation and communication in a parallel implementation, and describe techniques to overcome bottlenecks and achieve high performance evaluation of the Helmholtz potential for different distributions of particles. We demonstrate that the resulting implementation has a load balancing effect that significantly reduces the time-to-solution and enhances the scale of problems that can be treated using full wave physics.

Index Terms—Helmholtz equation, multilevel fast multipole method, tree algorithm, electromagnetic interaction, global interpolation

1 Introduction

Physics described by hyperbolic partial differential equations (PDEs) form the backbone of a wide array of modern technologies. Solutions to PDEs governing electromagnetics and acoustics have enabled technologies that have had, and will continue to have, a broad and profound effect on our daily lives. Our interest herein, is solution to the Helmholtz equation for radiation and scattering, posed in terms of convolution with a Green's function [1]. The common thread for the wide range of phenomena described by the Helmholtz equation is understanding and manipulation of wave physics at multiple length scales. This task is increasingly challenging given the increase in geometric complexity (smaller and more complex features) and wider range of operating frequencies (requiring more precision and detail to achieve optimal performance at all frequency bands). Advances in these technologies, and engineering sophisticated yet robust systems, are intimately tied to a detailed understanding of the underlying wave physics. Today, more often than not, such insight requires simulations via high-fidelity computational tools.

One of the main challenges in developing such tools is computing fields on electrically large objects. Here, electrical length is measured in terms of the largest linear dimension in terms of wavelengths. For several emerging problems, the electrical length can be several thousand wavelengths. For some perspective, the number of degrees of freedom typically required to model the physics (unless dictated by geometric constraints) scales as 10-15 per wavelength in a single dimension. The typical solution to the requisite integral equation proceeds via solving dense matrix equation iteratively. And, herein lies the problem–the dense matrix arises from evaluation of the Green's kernel that corresponds to an N-body problem. The Fast Multipole Method for Helmholtz equations (H-FMM) reduces the $\mathcal{O}(N_s^2)$ cost of direct potential evaluation to $\mathcal{O}(N_s \log N_s)$ [2] for surface distributions. Here, N_s denotes the number of degrees of freedom. This algorithm bears a strong similarity to that developed for Laplace equations (L-FMM), i.e., for non-oscillatory potentials such as gravitational or electrostatic fields [3]. The literature on the intricacies of both L-FMM and H-FMM (and their close cousins, tree-codes) are extensive; see [4], [5], [6], [7], [8], [9]. As is evident from these review papers, applications of these algorithms is extensive and cross-cutting in terms of the number of disciplines that it has benefited.

Given the wide applicability of FMM, a number of parallel algorithms and parallel implementations have been developed. Those developed for L-FMM have indeed been highly successful in terms of their performance and scalability. Indeed, several Gordon-Bell awards have gone to scalable L-FMM algorithms [10], [11], [12], [13]. This is in contrast to the development of parallel algorithms for H-FMM, despite sustained efforts [14], [15], [16], [17]. The

- *Michael P. Lingg and Hasan Metin Aktulga are with the Department of Computer Science and Engineering, Michigan State University, East Lansing, MI 48824 USA. E-mail: {linggmic, hma}@msu.edu.*
- *Stephen M. Hughey and Balasubramaniam Shanker are with the Department of Electrical and Computer Engineering, Michigan State University, East Lansing, MI 48824 USA. E-mail: {hugheyst, bshanker}@msu.edu.*

Manuscript received 28 June 2021; revised 31 Jan. 2022; accepted 4 Apr. 2022. Date of publication 7 Apr. 2022; date of current version 11 July 2022.
This work was supported in part by the NSF under Grant CCF-1822932, in part by the National Energy Research Scientific Computing Center (a DOE Office of Science User Facility), in part by the Office of Science of the U.S. Department of Energy under Grant DE-AC02-05CH11231, and in part by High Performance Computing Center at Michigan State University.
(Corresponding author: Michael P. Lingg.)
Recommended for acceptance by A. Bhatele.
Digital Object Identifier no. 10.1109/TPDS.2022.3165649

challenges to developing efficient parallel algorithms for H-FMM arise due to (a) the varying structure and cost of the computational workflow over the underlying tree representation, and (b) accuracy requirements of the target applications. To understand these issues better, we next present the nuances of H-FMM in comparison to L-FMM, summarize the existing literature on parallel algorithms for H-FMM and contributions of this paper that overcome some of the bottlenecks.

2 Problem Statement and Background

To set the stage for this work, we just consider the evaluation of the Green's kernel and not an an integral equation. Consider a collection of N_s point sources with intensity $u_n \in \mathbb{C}$ located at points $\mathbf{r}_n \in \mathbb{R}^3$, $n = 1, \ldots, N_s$. The potential $\Phi(\mathbf{r})$ due to these sources at some observation point $\mathbf{r}$ is then given by

$$\Phi(\mathbf{r}) = \sum_{n=1}^{N_s} g(\mathbf{r} - \mathbf{r}_n) u_n, \tag{1}$$

Here, the Green's function $g(\mathbf{r})$ for the Helmholtz equation is given by

$$g(\mathbf{r}) = \frac{e^{-jk|\mathbf{r}|}}{4\pi|\mathbf{r}|}, \tag{2}$$

where $k = 2\pi/\lambda$ denotes the wavenumber in rad/m and λ is the wavelength in meters. Note, the Green's function for the Laplace potential is recovered when $k = 0$. Sums of the form (1) often arise in electromagnetics and acoustics [18], in which $\Phi(\mathbf{r})$ must be evaluated at each source point $\mathbf{r}_m, m = 1, \ldots, N_s$, implying a cost of $\mathcal{O}(N_s^2)$. The multilevel fast multipole algorithm (MLFMA) [19], [20], or H-FMM as referred to in this paper, allows approximating these quantities in $\mathcal{O}(N_s \log N_s)$ for surface distributions or $\mathcal{O}(N_s)$ volumetric distributions to arbitrary precision.

To set the stage for the scope of the problems that we aim to address, and to provide some context, we will assume that the distribution of these N_s sources is related to the wavelength λ. We are *not* interested in pathological distributions, for instance along a line as in a wire or dilute as in light scattering from clouds/dust, etc. In a number of these cases, there are fast algorithms that outperform H-FMM [21], [22]. Likewise, in dilute distribution, it is possible to envision a scenario wherein a direct summation is faster. The motivating application for the presented work is acceleration of dense matrix-vector products arising from discretization of integral equations, specifically surface integral equations. In general, for this class of problems, the number of degrees of freedom $N_s \propto 1/\lambda^2 \propto f^2$ where f is the frequency. By and large, this rule governs the growth of the number of degrees of freedom. Of course, there are a number of situations wherein geometric features increase the number of degrees of freedom locally. The increase in density of distribution locally implies that one needs a hybrid/wideband algorithm: one that behaves like L-FMM in some regions and transitions to H-FMM elsewhere. There exists a sizable body of research, including parallel algorithms, that address this problem [15], [23], [24], [25], [26], [27], [28], [29]. In this paper, we do not consider a wideband algorithm. Our focus is purely on optimizing operations in the H-FMM algorithm presented in detail in [15], [27] that ensures error controllable operations.

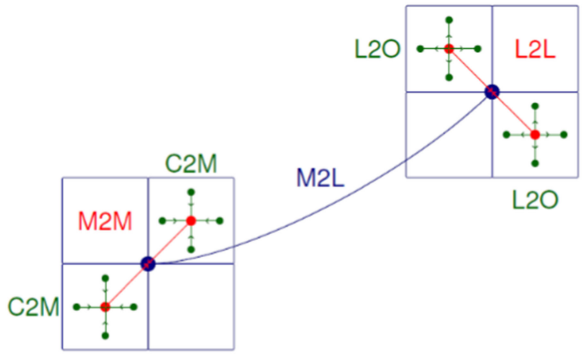

Fig. 1. Top-down view of the lowest two levels of an FMM tree, illustrating the key computational kernels. Particles (green dots) are mapped onto leaf nodes (small squares) in an octree partitioning.

Fast Multipole Methods. Both L-FMM and H-FMM follow the same algorithmic rubric. First, the computational domain is recursively subdivided into cubes (boxes) until a predetermined box dimension or a maximum number of particles per box is reached, and each particle is mapped to the box in which it resides. The hierarchical structure resulting from this recursive subdivision procedure can be represented as a tree specifically an *octree* in which each subdivision results in the creation of (up to) eight boxes of half the diameter in 3D.

Assuming an octree with L levels, let the root node reside at level $l = 1$, and the leaf boxes reside at level $l = L$. At all levels, boxes are classified as being either in the near- or far-field of each other if they are sufficiently close. Two boxes are in each other's near-field if their domains share a face, edge or node. At any given level, two boxes are designated to be in each other's far-field if (a) they are not in the near field of each other, and (b) their parents are in each other's near-field. This permits a hierarchical partitioning of the computational domain in terms of near- and far-field interactions. Near-field calculations take place *only* at the lowest level; at all other levels, interactions between boxes is performed via far-field through a five stage process shown in Fig. 1 and listed below (note, specific mathematical details are omitted in this discussion, and can be found in [23], [30] and elsewhere [5], [6]):

1) Compute charge to multipole information for each leaf node based on the particles it encloses (C2M),
2) compute the multiple expansions for each node in the tree by interpolating from the multipole information of all of its children (M2M),
3) calculate interactions between far-field pairs by translating multipole expansions of sources to the observers' locations (M2L),
4) starting at the highest level nodes and going all the way down to the leaves, shift and then downsample (anterpolate [1], [31]) multipole expansions aggregated at non-leaf observer boxes as a result of M2L (these expansions are then referred to as *local expansions*) from parent to child boxes (L2L),

5) convert the resulting local expansions at each leaf box to particles enclosed therein (L2O).

Laplace versus Helmholtz FMM and Ramifications on Parallelization. As alluded to earlier, highly efficient parallel implemetations of L-FMM exist [12], [13] in contrast to those for H-FMM [17], [27], [32], [33]. To understand why, one needs to examine the underlying mathematics.

- In L-FMM, irrespective of the tree level, the amount of multipole data needed to be stored at each node in the tree is identical because the magnitude of a Laplace potential monotonically decreases with distance.
- Due to the constant information content in L-FMM nodes, the per-level cost of far-field interactions falls off exponentially, and the bulk (80-90% or more) of the work in L-FMM is associated with the leaf nodes. Thus, existing L-FMM efforts have focused on optimizing the computation and masking communication costs primarily at the leaf level [34], [35], [36], [37].
- In contrast, in H-FMM the amount of multipole data for a node depends on the level due to the oscillatory nature of the potential. More precisely, the multipole data for a parent must be at least four times larger than that of its children to ensure that the desired level of accuracy can be obtained. This fundamental difference leads to major scalability and memory bottlenecks in H-FMM implementations [1], [15].
- Due to the quadrupling of the information content as one ascends an H-FMM tree, computational and memory costs in H-FMM stay constant for each tree level for surface geometries and halves for volumetric problems. For this reason, advances in parallel L-FMM do not translate readily to parallel H-FMM despite significant and sustained research [32], [38], [39], [40], [41], [42]. The commonly-used local essential tree (LET) [43] paradigm from the L-FMM literature fails for H-FMM because the size of the "ghost" region representing sources residing on other processes rapidly exceeds available memory.

Addressing these performance and scaling challenges, while maintaining high accuracies in potential evaluations, constitutes the main motivation for the present work.

Related Work. The aforementioned work profile of the H-FMM octree suggests that any efficient parallelization must strike a balance between distributing the many lightweight boxes at lower levels and distributing the work of the few heavyweight boxes at higher levels across processes. Furthermore, the choice of interpolation/anterpolation method significantly influences the design of the parallel algorithm. The existing algorithms address these different scenarios with different trade-offs. For the purposes of the ensuing discussion, we note that multipole and local expansions for any box at level l may be viewed as two-dimensional $N_\theta(l) \times N_\phi(l)$ arrays of sampled function values evaluated at angular points $(\theta_i, \phi_j),\ i = 1, \ldots, N_\theta(l),\ j = 1, \ldots, N_\phi(l)$ on the unit sphere $S^2 \doteq \{(\theta, \phi) \mid 0 \leq \theta \leq \pi,\ 0 \leq \phi < 2\pi\}$.

At scale, multipole and local expansions of octree boxes at the uppermost tree levels must be distributed across processes to achieve load balance [44]. To reduce the costs of communication in M2M and L2L for these distributed nodes, several authors have employed *local* interpolation techniques [33], [45], [46], in which only a small "halo" of nearby samples are required to compute each new entry of the multipole or local expansions. However, despite a slightly lower M2M and L2L asymptotic complexity of $\mathcal{O}(N_s \log N_s)$, when compared to exact ("global") interpolation's complexity of $\mathcal{O}(N_s \log^2 N_s)$ [47], [48], [49], this approach increases memory and computational costs due to the need to significantly oversample multipole and local expansions, in addition to reducing the numerical accuracy of the H-FMM. Alternatively, we propose the use of a parallel version of the *global* interpolation method, which has typically remained restricted to the serial M2M/L2L operations at lower levels of the tree. Though the global algorithm obviously has higher communication costs, it does not introduce additional numerical errors, and it facilitates optimal (minimum) sampling rates for multipole/local expansions [27], [50].

Local interpolation based hierarchical partitioning (HiP) approach distributes expansions hierarchically at the uppermost levels in block columns, or strips [46]. However, the M2M and L2L communication costs scale as $\mathcal{O}(\sqrt{N_s})$ per process, hampering the scalability of the H-FMM evaluation [32]. The blockwise HiP (B-HiP) strategy [33], [51] alleviates this bottleneck by distributing expansions in blocks, whose much lower surface-to-volume ratio results in $\mathcal{O}(1)$ communication costs per process, hence improving scalability [52]. In both methods, M2L operations are carried out in parallel by collecting on each process samples of the remote multipole expansions required to compute the local expansions it owns. It should also be noted that the increased sampling required by local interpolation hampers the scalability of the M2L phase, as collecting remote multipole expansions requires a significantly higher communication bandwidth compared to a global scheme with optimal sampling rates.

Building on the HiP approach, Yang *et al.* [17] transition from hierarchical partitioning to plane-wave partitioning (PWP) [44] for the highest levels of the tree, using a binary tree decomposition of the MPI communicator to flexibly load balance the computation. The PWP approach achieves zero communication overhead for M2L operations by distributing expansions at the uppermost levels of the tree by assigning each process a fixed window of samples for *all* expansions at a given level. However, the transition from HiP to PWP requires expensive communications in M2M and L2L phases to rearrange the expansions, though this cost may be justified by recognizing that each node interacts with at most 8 other nodes to perform the M2M/L2L operations, while the maximum number of nodes for M2L operations is 189 (with a volumetric problem).

As previously stated, local interpolation methods are convenient for parallelization but result in an H-FMM that is *not* strictly error-controllable. The principal challenge to a scalable H-FMM with error control is the communication cost of distributed global (exact) interpolation. In [15], Melapudi *et. al.* describe an error-controllable H-FMM based on global interpolation using a bottom-up partitioning which gives great flexibility for load balanced partitioning of the tree. This line of research in H-FMM with global interpolation has continued with the work reported by Hughey *et. al.* [27]. The scalability of the algorithm described by Hughey *et. al.* is hampered by its coarse-grained parallelization where all tree nodes are handled by individual processes.

While this scheme avoids most inefficiencies of the LET technique, it still fails to leverage the high degrees of parallelism available within the high level tree nodes. As will be discussed in more detail below, a fine-grained parallelization of samples within large tree nodes among multiple processes during M2L and complete elimination of redundant calculations in M2M/L2L phases are the main improvements of this work over that reported by Hughey *et. al.* [27].

Contribution. In this paper, we build upon our earlier efforts [15], [27] toward a scalable, error-controllable H-FMM based on global interpolation. We address several challenges regarding parallelization and communication, and we demonstrate an efficient and scalable method for evaluation of the Helmholtz potential. In particular, our contributions can be summarized as follows:

1) Development of a fine-grain parallel algorithm with bottom-up partitioning that enables scalable evaluation of deep uniform MLFMA trees,
2) maintain the high level of controllable accuracy shown in previous global interpolation implementations,
3) a detailed analytical model to characterize the complexity and memory use of the parallel algorithm for far-field interactions, and
4) demonstration of the overall algorithm performance and validation of this performance against our analytical model for different test scenarios.

3 Parallel Algorithms and Implementations

In what follows, we describe details of each stage of the algorithm. For completeness, we replicate some of the concepts introduced in [15], [27], before delving into details of our specific contributions. Mathematical formulae and operators used in this algorithm can be found in [15], [27].

3.1 Tree Construction and Setup

Let N_p denote the number of processes used in the computation. We initially distribute the N_s particles evenly across all processes and determine the diameter D_0 of the cube bounding the entire computational domain. Given the finest box diameter $D(L)$, the number of levels in the tree is calculated as the smallest integer L such that $L \geq \log_2(D_0/D(L)) + 1$. Once the number of levels and therefore the position of the leaf nodes are known, every particle is assigned a key based on the Morton-Z order traversal of the tree [53]. A parallel bucket sort on the keys is then used to roughly equally distribute particles across processes at the granularity of leaves. This is done by selecting $N_p - 1$ keys, or "splitters", which divide the Morton Z-curve into N_p contiguous segments. Whole leaves are uniquely assigned to processes using these splitters. Given a contiguous segment of leaf nodes, each process determines all ancestor keys of its leaves up to the root. The leaf through ancestor nodes are used to construct the *local subtree*. A simple method of storing the local subtree is as a post-order traversal array. To quickly access any node, we use an indexer into this local subtree array.

Plural Nodes. Despite the non-overlapping partitioning of leaf nodes, overlaps among different processes at the higher level nodes are inevitable (and in fact, are desirable) as shown in Fig. 2. Details and associated proofs on such partitioning

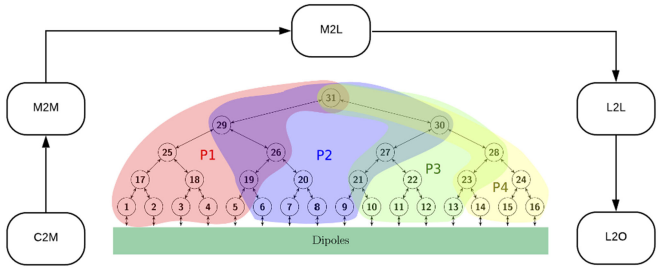

Fig. 2. Parallel farfield MLFMA.

can be found in [15]. We call such nodes shared by multiple processes *plural nodes*. While there are no limitations to the number of processes that can share a plural node, we designate a particular process, i.e., the right-most process sharing the plural node in the MLFMA tree, as its *resident process*. We refer to the resident process's copy of a plural node as a *shared node* and copies residing on other processes as *duplicate nodes*. We call the set of processes that own these duplicate nodes *users* of the shared node, denoted by $U(s)$, where s is the shared node.

One notable advantage provided by plural nodes is that storage of the node is split between multiple processes. In this case, the indexer additionally stores which *slice* of a tree node the current process actually stores in its local subtree array. As the information content for a node is not available to any single process in its entirety, a fine grain parallelization is necessary to perform computations associated with plural nodes. We note that a process can have at most two plural nodes per level in its local tree (essentially one to the "left", and another to the "right"); for proof see [15].

3.2 Parallel Evaluation

3.2.1 C2M

C2M is unchanged from our previous works [15], [27]. As each process is assigned a contiguous segment of whole leaf nodes, each process handles the C2M phase for its assigned leaf nodes in parallel independently.

3.2.2 M2M

In a nutshell, M2M creates multipole expansions of non-leaf boxes from the multipole expansions of their children. This first requires multipole expansions of all children to be *interpolated* to the size of the parent box using fast Fourier transform (FFT) interpolation. Next, each interpolated child expansion is *shifted* from the center of the child box to the center of the parent box. Finally, multipole contributions of every shifted child box are *aggregated* to form the multipole expansion for the parent box.

M2M computations start at the leaf level and proceed upwards in the tree following a post-order traversal of the local tree. Our parallelization of *M2M* depends on the level of the node and is described in Algorithm 1. The approach is as follows: i) Non-plural tree nodes (typically found at lower levels of the tree) are handled by their owner processes in parallel independently, ii) for plural nodes without any plural children, *interpolation* and *shift* steps for child nodes are performed sequentially, and the aggregation step is performed as a reduce-scatter operation among processes sharing the plural node, iii) plural nodes with plural children

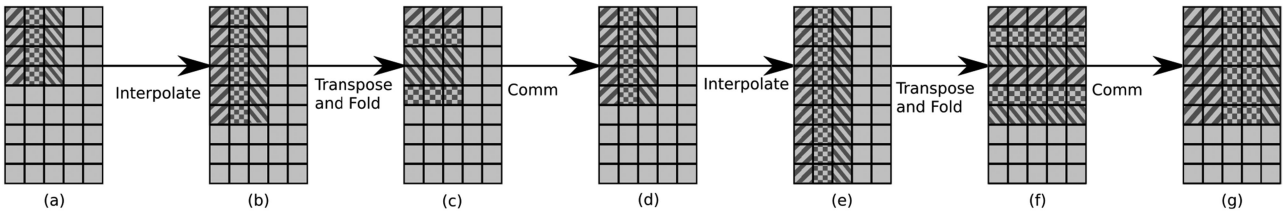

Fig. 3. Graphical illustration of the transposition and folding operations during fine grained parallel interpolation of multipole data of child node c to parent node p for $N_\theta = 3, N_\phi = 4$ (for c) and $M_\theta = 5, M_\phi = 6$ (for p) using 3 processes each of which owns a column of the initial multipole data as indicated by the hashmarks. Multipole data from (a) is interpolated along the ϕ dimension locally, leading to the multipole expansions in (b). The folding operation acting on the interpolated data is shown by the repositioning of the data as in (c). The hash marks show how the folded data is stored on the wrong processes, and must be communicated to the correct process as shown in (d). With the entire multipole data columns in the θ direction being available on each process, another set of interpolations are performed locally (e), which is then transposed and folded to yield the final multipole expansions (f). A final communication step is needed to send each θ vector to their owner processes (g) which can then be shifted to the center of the parent box and added to the parent's multipole expansions in accordance with the spherical symmetry condition.

(which typically are located at the higher levels of the tree and incur significant computational and storage costs) are processed using a fine-grained parallel algorithm that we discuss in more detail below.

Algorithm 1. Multipole-to-Multipole Interpolation

Require: $p.center$ coordinates of the parent box center
Ensure: pmp is parent's multipole representation
for each box p in post-order traversal **do**
  for each child box c **do**
    if c is plural **then**
      $mp[c] \leftarrow$ parallel_interpolation(c)
    else
      $mp[c] \leftarrow$ serial_interpolation(c)
    end if
    $smp[c] \leftarrow$ shift(mp, $p.center$)
  end for
  if p is plural **then**
    reduce_scatter(smp, $users(p)$)
  else
    for each child box c **do**
      aggregate($pmp, smp[c]$)
    end for
  end if
end for

Fine-Grained Parallel Interpolation. For plural nodes that necessitate fine-grained parallelization of M2M, the multipole data of the child nodes needed for FFTs are themselves split among multiple processes as indicated in Algorithm 1. Prior to elucidating our parallel algorithm, we note that our M2M implementation uses a Fast Fourier Transform (FFT)-based interpolation over the uniformly spaced multipole expansions of the child nodes [48]. FFT-based interpolation on the Fourier sphere requires equispaced samples along θ (vertical) and ϕ (horizontal) directions. Due to the interdependencies of the FFT algorithm, there is no way to partition the data so as to avoid communication.

Our approach is as follows: First, each process is assigned a (roughly) equal number of contiguous columns of multipole data (which correspond to groups of samples along the ϕ direction). The operation begins with a set of independent FFTs along these columns for interpolation in the ϕ direction, performed the same as in [48]. Then, the interpolated columns are shifted into rows (see Fig. 3), transposing and folding the samples in the θ direction into individual columns. The next step with serial processing would be FFT interpolation in the θ direction, but this data is split between processes sharing the plural node. Therefore, each process is communicated the θ samples they need to complete their assigned columns using an `MPI_Alltoallv` collective call. Now that each process is storing full columns of θ samples, these multipole data can be interpolated. The fully-interpolated multipole data is transposed and folded back to its original form (ϕ samples along columns, θ samples along rows). The data are again communicated back to the processes that own the corresponding multipole samples via another `MPI_Alltoallv`. These major steps are illustrated in Fig. 3.

Shifting of Interpolated Data. Shifting of multipole data is simply the translation of the interpolated samples from the child node's center to the parent node's center. Translation of each multipole data is independent of others and trivially parallelizable even in the case of fine-grained parallel M2M.

Aggregation. Aggregation requires adding all corresponding samples from each interpolated and shifted child node together to form the multipole expansion of a parent node (step (g) in Fig. 3). When children are owned by separate processes (as is the case for plural nodes), reduction communications are required. Note that in fine-grained parallel M2M for a plural node, each process owns only a portion of the parent node's multipole data. A reduce/scatter operation (for instance using `MPI_Reduce_scatter`) could perform both the aggregation and distribution of the appropriate portions of the aggregated multipole data to the processes sharing a plural node. One complication here is that the reduce/scatter operation would require memory to be allocated to the full-size of the parent node by each user process through padding the parts not owned by a process with 0s. Clearly, this would lead to significant memory and computational overheads, especially at the highest levels of the H-FMM tree (due to the large sizes of the plural nodes there). Therefore, we opt for a custom point-to-point aggregation scheme where the interpolated and shifted multipole samples from child nodes are communicated directly (via `MPI_Send` and `MPI_Recvs`) to the process that owns the corresponding samples of the parent node. If the source and destination are the same process, obviously no communication is performed. Each process sharing the parent node then adds up the corresponding multipole samples it receives from each child node, local or communicated. This method reduces both the amount of temporary memory necessary for aggregation and the overall communication volume.

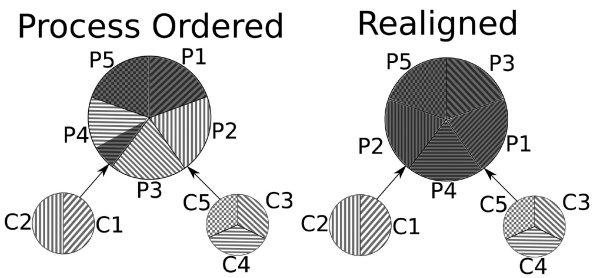

Fig. 4. This image shows how multipole samples, ordered clockwise starting at 12 o'clock, are assigned to processes owning the children nodes (C1-C5) overlap with processes owning the parent node (P1-P5) when assigned in process rank order on the left, and with our realignment scheme on the right. The darker portions on the parent samples show the regions where the parent node data overlap with the child node data, and are essentially local data that do not require communication.

Process Alignment. In fine-grained parallel M2M, performance impact of how the multipole data of child and parent plural nodes are mapped to processes sharing those nodes also needs to be considered. From the description of our custom point-to-point aggregation scheme above, it is evident that increasing the overlap of the multipole sample regions owned by a process in the child and parent nodes is critical for reducing the communication volume. As an extreme example, if a process owns no samples in the parent node that correspond with any of the samples it owns in the child node, all its interpolated and shifted child node samples would have to be communicated to another process for aggregation. In fact, this extreme example is not uncommon when multipole data of a node is simply partitioned into blocks and mapped to user processes according to their process ranks. This situation is illustrated on the left side of Fig. 4; some processes own samples of a parent node that has no overlap with the samples they own in the child node. Specifically, while process 1 owns overlapping samples in the child and parent nodes, process 2 and 3 own no overlapping samples.

As a heuristic to minimize the communication volume, we order processes within a parent node such that the parent node samples are assigned by following the priorities below to ensure maximal overlap with their child node samples:

1) Index of the lowest sample they own in the child nodes (lower comes first),
2) number of samples they own in the child nodes (fewer comes first),
3) process rank.

In the example given in Fig. 4, both process 1 and process 3 own samples with index 0 in the child nodes, but process 3 has a smaller number of samples so it is assigned the first portion of the parent node samples with process 1 being assigned the second portion. Following processes 3 and 1, process 4 owns the multipole sample with the lowest index in the child nodes, followed by process 2, and then process 5. As such, remaining portions of the parent node samples are assigned in this order. As can be seen in the figure, all samples each process owns in the parent node fully overlap with samples that they own in the child nodes, despite the non-uniform layout. With the proposed process alignment scheme, process 3 will still need to communicate some samples to process 1 for aggregation, but over half of the child samples interpolated by process 3 remains local. Note that with the straight-forward ordering of processes by their ranks, the entire child data interpolated by process 3 would have to be communicated to processes 1 and 2. In the new scheme, all processes use interpolated samples from their part of a child node without having to communicate. While this scheme would work best with a perfectly balanced tree, this approach will still be effective in reducing the communication volume during aggregation with any tree structure.

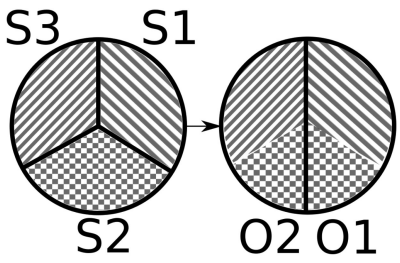

Fig. 5. A translation operation between two plural nodes shared by different numbers of processes.

3.2.3 M2L

The M2M step builds the multipole expansions of all tree nodes owned/shared by a process, starting from leaves all the way up to the highest level of computation. During M2L each observer node loops through all source nodes in its far-field and translates the source multipole data to its locale, aggregating the effects from all its far-field interactions in the process. When the source-observer node pair is on the same process, this interaction is handled purely locally. However, when the source node data is on another process (or a set of processes), one needs a load balanced algorithm that is communication efficient.

To understand the scope of the problem, consider Fig. 5. Here, the source node is a plural node shared by three processes (S1, S2 and S3); the observer node is shared by two processes (O1 and O2). Multipole samples for both are shared starting at the top of each circle and increasing clockwise (consistent with the process alignment scheme utilized during M2M). Process S1 and O1 both own multipole samples with the lowest indices, with S1 having less samples than O1. For this far-field interaction, S1 would need to send all samples that it owns to O1. S3 and O2 both own samples with the highest indices; here S3 would need to send all of its samples to O2. Finally, S2 owns samples that are needed by both O1 and O2, therefore S2 must send half of its samples to O1 and the other half to O2. Since each node in the H-FMM tree interacts with several others ($\approx$27 for surfaces and up to 189 for volumes) each of which may be shared by a varying number of different processes, it is evident that coordination of all communications that must take place during an H-FMM evaluation is non-trivial.

In an initialization step before the actual M2L operations, all processes discover the owner process(es) of the tree nodes (i.e., observers) which will need the multipole data for the source nodes they own, given the partitioning of leaf nodes (for load balancing purposes) and process alignments for plural nodes. This pre-calculated list is formed to carry out the actual communications that will take place during the ensuing H-FMM potential evaluations. If a source node or the corresponding observer node is plural, then the pre-

calculated communication list will include only the intersection of the multipole data owned by both the source and target processes. When a source node on a process is needed by multiple observers on another process, it is sufficient to include the source node in the communication list to that other process only once. Also, multipole data for multiple source nodes residing on one process that are needed by another can be combined into a single message in this communication list, even if the source nodes are at different levels. Note that the tree structure in most H-FMM applications are fixed. As such, the overheads associated with such an initialization stage is minimal.

Algorithm 2. M2L Translation

```
1: Determine the intersection of data owned by both source
   and target.
2: procedure M2L
3:   for each source box do
4:     for each farfield interaction target do
5:       if Target box is not local then
6:         for each process sharing target box do
7:           Add source multipole data in union of data owned
             by both source and target to buffer.
8:         end for
9:       end if
10:    end for
11:  end for
12:  Communicate buffers between processes.
13:  for each target box do
14:    for each farfield interaction source do
15:      if Source box is local then
16:        Load source multipole data from local memory.
17:        Translate source multipole data to target and add to
           existing translated data for target node
18:      else
19:        for each process sharing source box do
20:          Read source multipole data from communication.
21:          Translate source multipole data to target and add to
             existing translated data for target node
22:        end for
23:      end if
24:    end for
25:  end for
26: end procedure
```

While far field interactions between plural nodes (which constitute the most expensive communications in an H-FMM evaluation) could be carried out using all-to-all communications that involve only the users of the two corresponding plural nodes, due to the excessive number of M2L interactions present in large-scale computations (and hence the large number of different communicators that must be created), we choose to perform these communications using non-blocking point-to-point send/recv operations (i.e., `MPI_Isends` and `MPI_Irecvs`) in the default global communicator. Another reason for opting for a point-to-point scheme is that there are significant differences in the amount of data that must be sent to one process compared to another. As part of the initialization step then, each process allocates a message buffer for every other process that it will communicate with. The size of the send buffer is limited to avoid excessive memory use and maximize communication overlap.

To perform communications during M2L, every process first fills their send buffers for each process based on the precalculated communication list and initiates the message transmission using `MPI_Isends`. Immediately after initiating the sends, each process starts waiting for their expected messages using `MPI_Irecvs`. The status of these communications are checked periodically. Computations are overlapped with M2L communications in two ways. First, blocks of translations that are entirely local (which is actually common at the lower tree levels) are processed while the nonblocking send/recvs are taking place. Second, translation data that is detected as received during the periodic checks are applied immediately, overlapping the corresponding computational task with communications underway. Due to the limit we impose on the message buffer sizes, communications with processes that involve a large amount of data transfer need to be performed in multiple phases. Hence, upon reception/delivery of a message from another process, if there is more data to be transferred, a new non-blocking recv/send operation is initiated.

Translation Operators. Source node data is translated to the target node by multiplying it with a translation operator. The translation operators can be pre-calculated to reduce computational costs. As these can potentially take significant memory, we limit such memory use by each process by having them store the pre-calculated operators only for translation of the local and remote samples that they will actually need. This information can be determined from the pre-calculated M2L communication list. In case the memory available is not sufficient to store the needed operators, we use techniques outlined in [27] to sample and interpolate translation operators.

3.2.4 L2L

To anterpolate and distribute the translated local expansions down to the child nodes, L2L applies the operations in M2M in reverse order. First, local expansions at the parent node are shifted to the center of each child node, they are then anterpolated and percolated down the tree. Finally, the anterpolated data is aggregated with local expansions previously translated to the child node during the M2L stage.

Much like M2M, L2L is parallelized in three different ways: i) Non-plural parent tree nodes at the lower tree levels are processed independently in parallel by their owner processes, ii) for a plural node with a non-plural child, shifts involve communications, but the ensuing anterpolation and aggregation (with translated local expansions) are performed sequentially by the owner of the child node, iii) plural nodes with plural children require fine-grained parallelization.

Shift. In a parallel shift operation, parent node samples corresponding to those of the child node must be communicated by the processes sharing the parent node to the process(es) owning the child node. This is most easily done before the data has been shifted, as the parent will not need to know the position of the child node. In case of plural parent and non-plural child, this communication would essentially be a *gather*, and in case of plural parent with a plural child, it would be an *all-to-all*, in both cases involving all processes sharing the parent node. However, only a subset of the processes sharing the parent node will actually share

the child node. To avoid non-trivial issues that would arise from having to coordinate several collective calls among different subsets of processes, we again resort to point-to-point communications instead. Consequently, messages are only sent from processes owning a piece of the parent node to a process owning the corresponding piece of the child node. Once a process gathers all of necessary samples of the parent node, it applies the shift operation to all its multipole samples independently.

Anterpolation. Anterpolation would have to be performed in parallel only if the child node is a plural node. The procedure for parallel anterpolation is exactly the same as that of the parallel interpolation, except that the number of multipole samples is reduced (rather than increased).

Aggregation. Aggregating the shifted and anterpolated parent data with translated local expansions is trivial. Even in the case of plural child nodes, all required data is already available locally.

3.2.5 L2O

As in C2M, each process handles the L2O computations of its assigned leaf nodes in parallel independently.

4 Computational and Communication Complexity Analysis

In this section, we analyze the asymptotic computational and communication complexity of the parallel H-FMM algorithm described above. To simplify the analysis, we focus on two extreme cases, a 2D surface represented by points on a regularly spaced planar grid (dimension $d = 2$) and a 3D volumetric structure represented by points on a regularly spaced cubic grid ($d = 3$). These represent extreme cases, and hence are ideal for asymptotic analysis.

Denote the multipole truncation number at level l as $K(l) \propto kD_l$, where k is the wavenumber in the simulation medium and D_l is the diameter of a box at level l. Observe that the total number of samples in (θ, ϕ)-space is of order $\mathcal{O}(K(l)^2)$. All FMM algorithms are constructed such that the only operators that depend on particles are C2M and L2O, the operations of M2M, M2L and L2L only depend on the existence of the leaf node [54], [55], [56], [57]. As such we assume that each leaf node contains $\mathcal{O}(1)$ samples. It follows that the number of leaf nodes is $\propto N_s$, the number of source points. For simplicity and with no loss in generality, we assume that the constant of proportionality is 1. Next, we denote the number of nodes at level l by $G(l)$. The total number of levels is given by N_L. As one moves up the octree, we observe that the number of groups per level is reduced by roughly 4 times for the 2D surface and 8 times for the 3D volume. Leveraging the relation between the dimensionality of a structure and the rate of decrease in the number of nodes per level, one can write $G(l)$ as follows:

$$G(l) = \frac{N_s}{(2^d)^{(l-1)}}. \tag{3}$$

Since $K(l)$ doubles at each level, given $K(1) = C_k$, it follows that

$$K(l) = 2^{(l-1)} C_k. \tag{4}$$

Finally, we define P as the number of processes, P_L as the level where (almost) all nodes in a level start becoming plural and $P_N(l)$ as the average number of processes sharing a plural node at level l. Equivalently, P_L is the level when $G(P_L) < P$ for the first time, and this remains true from hereon to the root. Given P, N_s and d,

$$P_N(l) = \frac{P}{G(l)} = \frac{P(2^d)^{(l-1)}}{N_s}. \tag{5}$$

4.1 Interpolation (M2M)

Computational Complexity. M2M is performed for each node, starting from the leaf level up to the highest level N_L. The dominant component in the computational complexity for $M2M$ is FFT-based interpolation. Shifting and aggregation are $O(K(l)^2)$ operations each, while interpolation for a given node costs $O(K(l)^2 \log^2(K(l)))$. This gives a total computational complexity of

$$C \propto \sum_{l=1}^{N_L} G(l) K(l)^2 \log^2(K(l)). \tag{6}$$

Plugging in the Equations (3) and (4) and simplifying the summation, we obtain the computational complexity for a surface to be

$$C \propto O(N_s \log^2 N_s), \tag{7}$$

and for a volume to be:

$$C \propto O(N_s). \tag{8}$$

Number of Messages (Latency). Communication in $M2M$ happens during aggregations for both coarse-grained and fine-grained parallel $M2M$s, as well as the FFTs of the fine-grained parallel $M2M$s. As described in Section 3.2.5, we perform aggregations (which are effectively reduce-scatter operations) using point-to-point communications. In an ideal tree, every source and observer node will be divided among the same number of nodes. This means the portion of a source node owned by any process will only be owned by a single process in the observer node, limiting the communication for each source node process to one process in each observer node. Since this is done for each group at each level, the total message count for aggregations becomes

$$M_{Ag} \propto \sum_{P_L}^{N_L} G(l) P_N(l+1). \tag{9}$$

Using expressions for $P_N(l+1)$ and $G(l)$, yields the number of messages for aggregation

$$M_{Ag} = O(P \log(N_s) 2^d). \tag{10}$$

Here, we ignore aggregations that would be needed for plural nodes (located at process boundaries) below level P_L. Note that there may only be two such plural nodes per level for each process and these aggregations will involve only two processes. As such, their contribution to the number of messages during aggregations is of a lower order term.

Next, consider the parallel FFTs in fine-grained parallel $M2M$s. In this case, an all-to-all communication is performed after each of the two fold and transpose operations. As we implement these all-to-all communications using point-to-point calls, the message count for FFTs is then

$$M_{FFT} = \sum_{l=P_L}^{N_L} G(l)P_N(l)^2 \propto O(P^2). \tag{11}$$

Consequently, the total number of messages for $M2M$ is

$$M_{M2M} = O(P^2 + P\log(N_s)). \tag{12}$$

Note, the N_s portion of the equation above is only going to matter when P_L is greater than the number of levels in the tree. In all other cases, increasing the height of the tree does not increase the number of levels with plural nodes. Given that it is practically useless to have more processes than the number of leaf nodes (which is the condition required for P_L to be more than the tree height), the message count can be simplified to $M_{M2M} = O(P^2)$.

Communication Volume (Bandwidth). Bandwidth during interpolation is due to all to all communications during interpolation, and a reduce scatter during the aggregation. Each of these operation communicates up to the entire node, resulting in a bandwidth that can be written as

$$B \propto \sum_{l=1}^{N_L} G(l)K(l)^2. \tag{13}$$

Applying the previous definitions for $G(l)$ and $K(l)$ yields a communication bandwidth of

$$B \propto N_s\log N_s, \tag{14}$$

for the surface geometry, and

$$B \propto N_s, \tag{15}$$

for the volume geometry.

4.2 Translation (M2L)

Computational Complexity. The complexity for the translation operation at a given level is directly proportional to the number of multipole samples for nodes, the average number of interactions per node (denoted by $I(l)$ for level l), and the number of nodes at that level. Summing these costs across all levels, we obtain

$$C \propto \sum_{l=1}^{N_L} K(l)^2 I(l)G(l). \tag{16}$$

While the number of interactions for a node changes based on its exact position in the geometry (for instance, corner or edge nodes), the upper limit is the constant $6^d - 3^d$. Using the equations for $K(l)$ and $G(l)$, computational complexity of the translation step can be simplified to

$$O(N_s\log N_s), \tag{17}$$

for the surface structure, and to

$$O(N_s), \tag{18}$$

for the volume structure.

Number of Messages (Latency). At level P_L or above, a process can have multipole samples for only one node. Since a process owns at most part of a single node, each of its interactions will require a separate communication because the nodes in its far-field will all reside on different processes. Assuming an ideal tree partitioning where the source and target nodes are shared among the same number of processes, the kth process for the target node will only need the source node data from the kth process of the source node. As we limit the size of each translation message, the number of messages will then be proportional to the communication volume between a pair of processes divided by the message buffer size M_S. At levels below P_L, a process can own multiple nodes. Here groups of nodes can be communicated to the same process, if all source nodes reside on one process and all observer nodes reside on another. In this case, the interaction count is going to be based on the total amount of data communicated between the two interacting processes, divided by the message buffer size, summed up for all interacting processes.

Considering contributions at/above P_L and below P_L gives a total message count of

$$M \propto \sum_{l=P_L}^{N_L} PI(l)\frac{K(l)^2}{P_N M_S} + PI(l)\left\lceil \frac{\sum_{l=1}^{P_L-1} K(l)^2 \frac{G(l)}{P}}{M_S} \right\rceil, \tag{19}$$

where M_S is the size of message buffers. For the surface geometry, this can be simplified to $M \propto O(N_s\log N_s) + O(N_s)$ (where the first term is for levels $\geq P_L$ and the second term is for levels $< P_L$), and for the volume geometry it can be simplified to $M \propto O(N_s)$ (with both below and above P_L having the same impact).

Communication Volume (Bandwidth). Communication volume can be analyzed in two parts as well. At and above P_L, all multipole data for every source node must essentially be communicated to every target node as no process contains any multipole data other than its own. Even if the number of processes increases, still the same amount of data needs to be transmitted, just among an increased number of nodes. Therefore, for level at or above P_L, the communication volume is independent of the number of processes

$$B \propto \sum_{l=P_L}^{N_L} K(l)^2 G(l)I(l). \tag{20}$$

This expression simplifies to $O(N_s\log N_s)$ for the surface geometry and to $O(N_s)$ for the volume geometry.

Below P_L, each process will own more than one node, nodes will be interacting with nodes on the same process, or multiple nodes owned by a neighboring process. In fact, only nodes within two nodes off the edge of process boundaries will require communications with other processes. Total communication bandwidth can then be expressed as

$$B \propto \sum_{l=1}^{P_L-1} PK(l)^2(S_N), \tag{21}$$

where S_N is the number of nodes that have nodes in its far-field from at least one (out of the 8 possible neighboring processes for the surface and 26 for the volume) other processes

touching them and can be represented as $S_N = \frac{G(l)^{\frac{d-1}{d}}}{P} = \frac{1}{P}(\frac{N_s}{(2^d)^{(l-1)}})^{\frac{d-1}{d}}$. With this definition of S_N, the total communication volume for M2L below P_L becomes

$$B \propto O(N_s), \tag{22}$$

for the surface, and

$$B \propto O(N_s^{\frac{2}{3}}), \tag{23}$$

for the volume, due to the lower portion of the tree dominating.

4.3 Anterpolation (L2L)

As mentioned before, L2L is the reverse operation for M2M. Similar to M2M, anterpolation dominates the computational complexity for L2L. Computational complexity for anterpolations is the same as that of interpolations, so L2L's computation complexity is the same as M2M's. Likewise, communications performed are the same but in reverse order. Therefore, the latency and bandwidth costs of L2L are the same as those of M2M.

5 Performance Evaluation

In this section, we evaluate the performance of the parallel H-FMM algorithm described. All results were obtained on the Cori-Haswell supercomputer at National Energy Research Scientific Computing Center (NERSC). Each node on this system contains two sockets with Intel Xeon E5-2698 v3 (Haswell) processors clocked at 2.3 GHz. Each node has 32 cores and 128 GB 2133MHz DDR4 RAM. The Haswell system uses the Cray Aries with Dragonfly topology interconnect network. The code is implemented in Fortran 90 using only MPI parallelization and was compiled with the Intel compiler, version 19.0.3.199. The Cray-FFTW library, version 3.3.8.4, is used for all FFT operations.

The runs here focus on the timing of the M2M, M2L and L2L phases of the tree traversal. As discussed in Section 4, operations in these phases only depend on the existence and location of the leaf nodes. Thus, we simply populate each leaf node only with a single unknown, effectively bypassing the near-field, C2M and L2O processing steps which are not prone to scaling bottlenecks anyway, allowing us to focus on the high frequency portions of the tree where nodes become too large to efficiently manage on a single process. For work that includes the lower frequency regions of the tree, see [27]. We note that this actually makes the number of unknowns being processed much smaller than what could be processed by M2M, M2L and L2L in the same amount of time when analyzing actual physics as in [27], and it makes the overall parallel scaling of our implementation look lower than what it would be in practice. In a typical 0.25λ leaf box (as we use in the runs below) with a $0.1\ \lambda$ discretization rate, one could potentially have anywhere between 100-180 particle per box. Our largest tree being processed is 14 levels with 42 million points for one point per leaf box. If the leaf nodes were fully populated, this tree would be equivalent to processing a tree with 4.2 to 7.5 billion points. Populating leaf nodes would increase the time of the C2M and L2O steps, but have no impact on the execution times of the M2M, M2L and L2L phases. Additionally, all runs use the same θ and ϕ discretization as set by the 0.25λ leaf box size and an oversampling rate of $\chi = 1.1$, see [40], [50].

Numerical results from all runs were checked against results in our previous work [27] to verify the only differences were due to floating point precision, and the previous work showed the results to be error controllable with respect to the analytical solution. We note that this previous work of ours [27] addressed the load balancing issues in H-FMM for highly non-uniform geometries. Here, we mostly focus on improvements to parallel processing of higher level nodes and the complexity analysis of the algorithm. Thus, we perform our numerical experiments on uniform geometries to verify that the proposed techniques are performing optimally with respect to the expected complexity.

For all runs testing an increasing number of processes, the first run is performed with the lowest number of nodes that can execute the geometry without an out of memory exception using 32 processes per node (1 process per core). Processes are assigned to cores using srun, with –cpus-per-task set to 1 and –nodes set to the number of processes divided by 32. The number of processes are increased by increasing the number of nodes used, while maintaining 32 processes per node.

5.1 Load Balance With the Fine Grain Parallel Algorithm

The intent of the fine-grain parallel algorithm is to provide improved balance at the upper tree levels where a small number of large nodes reside. First, we look at the performance of a planar grid of particles (in the $z = 0$ plane) of dimensions $512\lambda \times 512\lambda$ with a grid spacing of $\lambda/4$ and 4,194,304 particles in total. The box size is chosen to be 0.25λ, resulting in a 12-level tree with 10 levels of computation. As seen in the upper subfigure of Fig. 6, the resulting execution profile is very balanced across process ranks. Execution time of the fastest to the slowest process varies by only 1.43%. Balance of total time can be a little misleading as M2L cannot progress until all processes that a given process interacts with completes their M2M stage. However, M2M execution times are also very balanced, varying from slowest to fastest process by 5.23%.

Next, we look at the performance for a sphere of diameter 384λ discretized using 4,542,208 dipoles on the surface with a leaf box size of $d_0 = 0.25\lambda$, yielding an 11-level tree. This geometry is less balanced than the grid (see the lower subfigure of Fig. 6) as high level nodes can range from having no children due to no particles being in that part of the geometry at the leaf level, up to having a completely filled quad tree from the leaf level up to a high level node. This results in notable imbalance in M2M, which as discussed before, results in delays in M2L execution. Note that while there are no explicit barriers between phases, there is an implicit barrier at the beginning of M2L processing, as an M2L interaction communication cannot proceed until both interacting processes have completed their M2M phases (though the faster process can perform local translations while waiting). Another (less significant) implicit barrier occurs at the beginning of L2L where nodes that are fully owned by a process must have all of the data from the translated parent node to

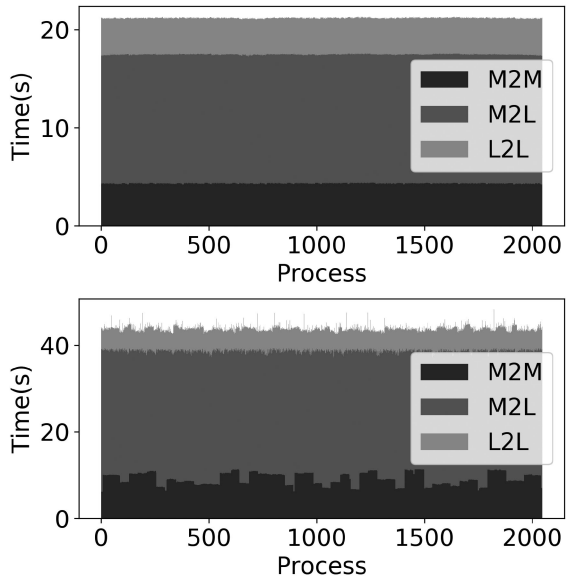

Fig. 6. Process execution times for a grid and a sphere geometry.

perform anterpolation on this parent node. The M2M execution range from the fastest to the slowest processes varies by as much as 84.5%. Despite this imbalance, long execution times are not clustered among a small group of processes as one would see if each of the highest level tree nodes were to be handled by a single process without fine-grain parallelization. Even in this unbalanced geometry, the fine-grain parallel algorithm is helping to maintain a good load balance across processes.

5.2 Scalability

Next, we investigate the strong scaling efficiency of our parallel Helmholtz FMM algorithm, first on a surface and then on a volumetric structure. In this section we focus on very uniform geometries to simplify comparison of estimated complexities with measured performance. For previous analysis of non-uniform geometries, see [1].

For the 2D surface structure, we use the same $512\lambda \times 512\lambda$ planar grid as above. As our base case for strong scaling efficiency, we use the performance on 128 cores because this is the smallest number of cores that this problem can be executed on due to its memory requirements. As seen in Table 1, both the interpolation and anterpolation phases (M2M and L2L) perform very well with the increasing process counts, while M2L's performance falls off rapidly (down to 25% efficiency on 2,048 cores). There are a couple of factors that contribute to this difference we observe in scaling characteristics. First factor is that M2M and L2L incur significant communications only at the highest level nodes, while M2L communications occur at every level where the source and observer nodes are on separate processes, which may essentially happen all the way down to the leaf nodes. Second, and more importantly, M2M and L2L computations involve relatively computation-heavy FFTs in between its communication steps. When the number of nodes in a level exceeds the number of processes, no process can own both a source and observer node of any translation, so all node data must be communicated. As the number of processes approaches the number of leaf nodes, the M2L communication bandwidth asymptotically approaches the worst case estimate. This means the increase in M2L bandwidth exceeds the worst case estimate increase as the number of processes approaches the number of leaf nodes. Despite M2L not scaling very well, the fine grained parallel algorithm presented still provides good speedups, nearly an 8x speedup when going from 128 to 2,048 processes without showing any performance stagnation.

Next, we examine strong scaling on a $32\lambda \times 32\lambda \times 32\lambda$ volumetric structure (Table 2). From 128 to 512 processes, we observe very good scaling (80% overall efficiency), but then parallel efficiency drops off quickly (down to 50% at 2,048 cores). In a volumetric problem, each tree node has a large number of nodes in its far-field (up to 189). Therefore the overall execution time is largely dominated by the M2L stage which does not manifest good scaling. The ideal scenario for our algorithm is when the nodes of a given level are distributed evenly among processes, i.e., when the number of processes divides evenly into the number of nodes in a level or vice versa. This does not occur at 1,024 or 2,048 processes for this particular problem. Nevertheless, the overall speedup remains at around 8x when going from 128 to 2,048 processes.

Finally, we look at scaling on the 384λ sphere (Table 3). As seen in the load balance analysis of the previous subsection, load imbalances result in the faster processes having to wait for slower processes. This results in a noticeable drop in scaling efficiency of the M2M phase, where the imbalance has the greatest impact, as well as the M2L phase, where some processes that are already in their M2L phase have to wait for others that are still in their M2M phase. This also has an impact on the overall speedup. While increasing the number of processes continues to improve execution times, the speedup when going from 128 to 2048 processes is just under 5x in the sphere case.

5.3 Complexity Analysis

To help validate the complexity analysis presented in Section 4, the software was instrumented to report the computational cost, the number of messages sent and the size of

TABLE 1
Performance of the MLFMA Algorithm on the 512λ Grid Geometry

	Grid (s)				Speedup	Par Eff. (%)			
N_p	M2M	M2L	L2L	Tot	Tot	M2M	M2L	L2L	Tot
128	5.80	5.30	5.30	18.55	1.00	1.00	100	100	100
256	3.06	4.13	2.66	11.21	1.65	0.95	64	99	83
512	1.52	2.76	1.31	6.42	2.89	0.95	48	101	72
1024	0.81	2.12	0.69	4.05	4.58	0.89	31	95	57
2048	0.43	1.31	0.37	2.38	7.78	0.84	25	89	49

TABLE 2
Performance of the MLFMA Algorithm on the 32λ Volumetric Geometry

	Volume (s)				Speedup	Par Eff. (%)			
N_p	M2M	M2L	L2L	Tot	Tot	M2M	M2L	L2L	Tot
128	0.527	2.15	0.526	3.26	1.00	1.00	100	100	100
256	0.266	1.10	0.263	1.68	1.94	0.99	97	99	97
512	0.14	0.679	0.135	0.99	3.27	0.93	79	97	82
1024	0.079	0.380	0.084	0.574	5.68	0.83	71	78	71
2048	0.051	0.271	0.058	0.406	8.03	0.65	50	57	50

TABLE 3
Performance of the MLFMA Algorithm on the 384λ Diameter Sphere Geometry

	Sphere (s)				Speedup	Par Eff. (%)			
N_p	M2M	M2L	L2L	Tot	Tot	M2M	M2L	L2L	Tot
128	6.71	13.54	5.29	26.76	1.00	1.00	100	100	100
256	3.86	10.05	2.7	18.00	1.49	0.87	67	97	74
512	2.34	6.20	1.46	11.04	2.42	0.72	55	91	61
1024	1.23	4.32	0.72	7.70	3.47	0.68	39	92	43
2048	0.92	3.19	0.416	5.58	4.79	0.46	26	79	30

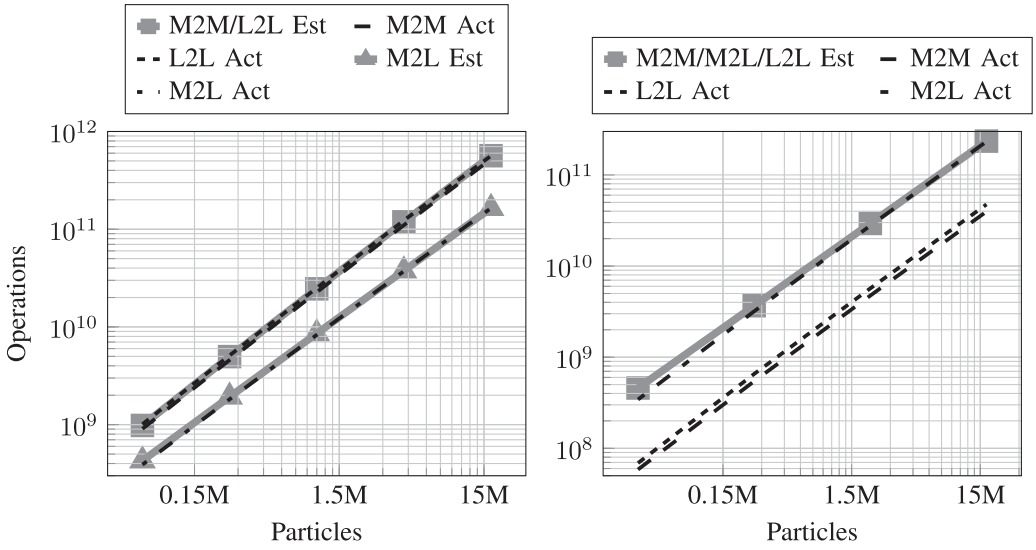

Fig. 7. Actual versus estimated computational complexity. The left subfigure shows results for the surface geometry, while the right subfigure is for the volume geometry.

these messages. In accordance with the geometries analyzed in Section 4, data was collected on the grid geometries ranging from 64λ to 1024λ and volume geometries ranging from 16λ to 16λ to as these geometries produce perfect quadtrees of heights ranging from 9 to 13 levels and octrees of heights ranging from 7 to 11 levels, respectively. As complexity estimates are asymptotic, they are scaled by least-squares fit to help visualize how well the estimates match the actual measurements.

Fig. 7 shows the actual versus the estimated overall computational complexities for the surface and volume geometries. Note that the computational complexity for the M2M and L2L phases for a surface was given in Eq. (7), and for a volume in Eq. (8). Similarly, the computational complexity for the M2L phase for a surface was estimated in Eq. (17), and for a volume in Eq. (18). The actual complexities match the estimates very closely. This indicates that the implementation of this algorithm does not have any unnecessary overhead costs in computation as computation is near to the ideal for Helmholtz FMM.

Fig. 8 shows the actual versus the estimated communication volumes for each phase separately. Estimated communication volume for the M2M and L2L tree traversal phases for a surface geometry is shown in Eq. (14), and for a volume geometry in Eq. (15). Estimated communication volume for the M2L tree traversal phase for a surface geometry is shown in Eq. (22), and for a volume geometry in Eq. (23). Of note is how the measured communication volume drops off relative to the estimate. We believe this is due to the number of samples producing a tree with more nodes at lower levels than the number of processes. Hence, many nodes are fully owned by a single process and require no

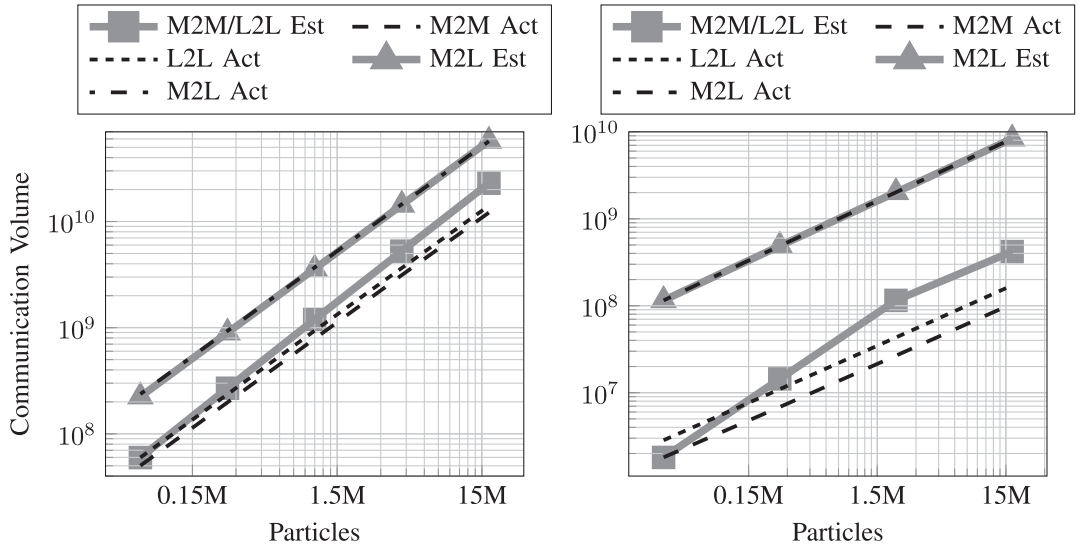

Fig. 8. Actual versus estimated communication volume. The left subfigure shows results for the surface geometry, while the right subfigure is for the volume geometry.

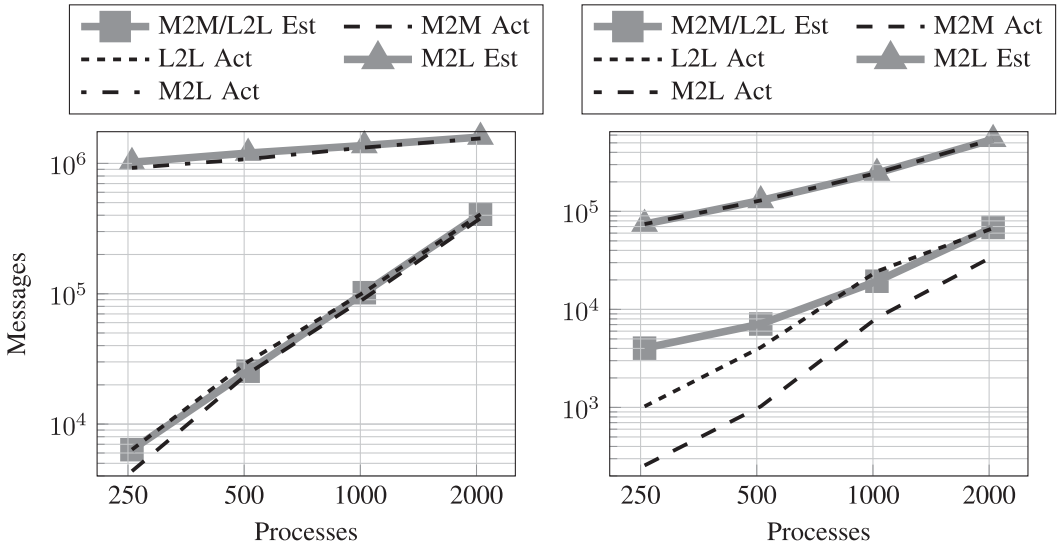

Fig. 9. Message counts versus expected Big-O message counts. The left diagram shows analysis of a surface geometry, while the right diagram shows analysis of a volume geometry.

communication during M2M or L2L. Increasing the number of processes would lead to more levels with plural nodes, bringing the communication volume closer to our estimates. M2L does not show the same communication volume falloff as M2M and L2L compared to the estimated volume because fully owned nodes still require data from the source nodes to be communicated to the process owning the observer node. Such communications will be required all the way down to the leaf nodes.

Fig. 9 shows the measured worst case messages versus the estimated worst case messages for M2M and L2L. Estimated message count for the M2M and L2L tree traversal phases is shown in Eq. (12). Estimated message count for the M2L tree traversal phase for a surface geometry is shown in Eq. (19). M2M and L2L Message counts are dominated by the P^2 complexity of the all to all communications and the actual message count reflects this. The M2L prediction simplifies a very complex process that results in the number of M2L messages communicated.

5.4 Process Alignment

In Table 4, we compare the number of packets sent between Rank Ordered and Process Aligned schemes during M2M and L2L phases for the 512λ grid geometry. There is a notable reduction in the number of messages exchanged, and hence the overall bandwidth, for lower process counts and continued reduction at higher process counts as expected. This reduction is likely to be effective in the relatively good scaling characteristics of M2M and L2L phases.

5.5 Memory Utilization

Tables 5 and 6 shows the total program memory utilization of the three data structures with largest memory needs with increasing process counts. As expected, the memory used for tree storage (Tree Mem) does not increase with process count, despite some fluctuations due to different partitionings of the leaf nodes. This shows that the tree data structure is being nicely partitioned across processes. Size of the translation operators (Trans Ops) increase slowly with process

TABLE 4
Comparison of the Number of Packets Sent Between Rank Ordered and Process Aligned Schemes for the 512λ Grid Geometry in Millions of Packets Sent

N_P		128	256	512	1024	2048
Rank Ordered	M2M Bandwidth	1714	1872	3009	3126	4246
	L2L Bandwidth	1206	1166	2429	2382	3649
	Combined Bandwidth	2920	3038	5438	5508	7895
Process Aligned	M2M Bandwidth	1468	1662	2711	3272	4412
	L2L Bandwidth	960	955	2131	2104	3338
	Combined Bandwidth	2428	2617	4842	5376	7750
Delta		-492	-421	-596	-132	-145

count, slightly more than doubling going from 128 to 2,048 processes. This is due to the spatial distribution of the tree nodes; multiple source observer pairs with the same translation in the tree may belong to different processes. Particularly, as the process count increases and the number of nodes in a process decreases. This results in some processes storing some of the same translation operators as the other processes. The greatest memory increase is in the message buffers (S/R Buffs). The translation send and receive buffers (S/R Buffs) are used to communicate the data for source nodes that interact with nodes in another process. Single node communications for each source and observer pair would eliminate the need for this buffer, but would result in drastically more translation messages which would degrade performance. So the translation message buffers are maximized to use any remaining memory to limit the number of translation messages that must be sent.

On the other end of what can be performed with H-FMM is the volume geometry. Here the number of nodes per level is significantly increased due to the underlying full oct-tree structure (as opposed to a quad tree for a surface geometry), but the maximum height of a tree that can be computed is reduced. Most memory is reduced due to the shorter height of the tree, which reduces the size of the nodes at the top of the tree. However, the translation message buffers still use up as much memory as possible to improve translation communication performance.

5.6 Performance Comparison With BEMFMM

Finally, we compare our approach against another H-FMM implementation. We note that open-source H-FMM codes are almost non-existent, with the BEMFMM [58] being a recent exception. BEMFMM is an FMM-accelerated Boundary Element Method (BEM) parallel solver framework for wave scattering problems. It uses the exaFMM backbone [59] for its matrix-vector multiplication kernel and surrounds it with a GMRES solver for solving the boundary integral equations. To access the exaFMM kernel for comparison of mat-vec times, we have taken a number of steps. These are as follows: (a) The point cloud used in our code is derived from data that is an input into the exaFMM tree structure. (b) The leaf box size is identical as is the tree. For instance, for the spherical distribution, we use the same 1.44 million point clould (generated from 240,000 panels). This results in an 8-level tree with 65471 leaf boxes in *both codes*. Using this distribution, we have comapred *only* mat-vec timings between exaFMM kernel in BEMFMM and H-FMM kernel as seen in Table 7. As is evident from Table 7, our fine grain parallel Helmholtz FMM algorithm shows significantly better performance than exaFMM (in BEMFMM).

There are two root causes for the significant performance differences observed: First, our M2M/L2L implementations are based on FFT-based global interpolation methods that are known to be asymptotically optimal in terms of computational costs, whereas BEMFMM uses dense rotation based ("point and shoot") M2M/L2L operations as in the Laplace FMM codes that have $\mathcal{O}(K(l)^3)$ computational complexity at level l, compared with our $\mathcal{O}(K(l)^2 \log K(l))$. The second important cause is that our implementation uses fine-grained parallelization of H-FMM operations across the tree as explained throughout this paper, but the BEMFMM code uses a locally essential tree (LET) [60], which does not parallelize the processing of higher level nodes as effectively as our approach. While the first cause explains the speedup we obtain over BEMFMM at two processes, the impact of fine-grained parallelization is seen in the increasing speedups we obtain as the number of processes are increased.

TABLE 5
Total Memory Utilization (in GBs) by the Three Largest Data Structures for the 512λ Grid Geometry

N_p	S/R Buffs	Trans Ops	Tree Mem
128	1.7	107.5	52.7
256	4.5	141.0	62.0
512	102.7	162.2	52.8
1024	257.0	199.0	62.0
2048	431.8	221.4	52.8

TABLE 6
Total Memory Utilization (in GBs) by the Three Largest Data Structures for the 32λ Volume Geometry

N_p	S/R Buffs	Trans Ops	Tree Mem
128	3.1	6.6	5.0
256	8.4	10.1	5.2
512	22.3	16.3	5.5
1024	31.9	21.4	5.1
2048	51.7	30.4	5.4

TABLE 7
Comparison of BEMFMM versus Our Parallel MLFMA Implementation (Referred to as "this work")

Processes	2	4	8	16	32	64	128	256	512
BEMFMM (s)	108.92	100.85	148.69	91.03	60.45	35.72	29.42	26.72	24.72
This work (s)	8.29	4.28	2.37	1.26	0.68	0.39	0.23	0.17	0.15
Speedup	13.1	23.6	62.7	72.2	88.9	91.6	127.9	157.1	164.8

6 Conclusion and Future Work

We have demonstrated a novel method for parallel computation of large, upper level tree nodes in large-scale Helmholtz FMM which helps alleviate a key performance bottleneck associated with node dependency. The complexity of this method has been characterized. The results presented support the provided characterization and show the balance provided by this method. The performance of the algorithm has been shown to compare favorably with an existing Helmholtz FMM implementation.

Beyond the improvements presented, further work can be performed to improve memory usage, as well as the M2L communication overhead. One possible method is a hybrid approach of MPI parallel with thread parallel. Using thread parallel within a given node provides the opportunity to exploit shared memory parallelism. All cores within a node can be assigned to shared memory threads, rather than MPI processes, eliminating the need to communicate between these threads, and reducing duplicate memory allocation.

References

[1] S. Hughey, H. Aktulga, V. Melapudi, B. Shanker, M. Lu, and E. Michielssen, "Parallel non-uniform MLFMA for multiscale electromagnetic simulation," in *Proc. IEEE Int. Symp. Antennas Propag. USNC/URSI Nat. Radio Sci. Meeting*, 2018, pp. 1837–1838.

[2] B. Dembart and E. Yip, "The accuracy of fast multipole methods for maxwell's equations," *IEEE Comput. Sci. Eng.*, vol. 5, no. 3, pp. 48–56, Third Quarter 1998.

[3] L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations," *J. Comput. Phys.*, vol. 20, pp. 63–71, 1987.

[4] B. Shanker and H. Huang, "Accelerated cartesian expansions - A fast method for computing of potentials of the form r^{-ν} for all real ν," *J. Comput. Phys.*, vol. 226, pp. 732–753, 2007.

[5] M. Vikram and B. Shanker, "An incomplete review of fast multipole methods from static to wideband as applied to problems in computational electromagnetics," *Appl. Comput. Electromagn. Soc. J.*, vol. 27, 2009, Art. no. 79.

[6] N. Nishimura, "Fast multipole accelerated boundary integral equation methods," *Appl. Mechanics Rev.*, vol. 55, no. 4, pp. 299–324, 2002.

[7] A. Appel, "An efficient program for many-body simulations," *SIAM J. Sci. Comput.*, vol. 6, pp. 85–103, 1985.

[8] J. Barnes and P. Hut, "A hierarchical $((n \log n)$ force calculation algorithm," *Nature*, vol. 324, pp. 446–449, 1986.

[9] L. Greengard, *The Rapid Evaluation of Potential Fields in Particle Systems*. Cambridge, MA, USA: MIT Press, 1988.

[10] L. Ying and D. Zorin, "A simple manifold-based construction of surfaces of arbitrary smoothness," *ACM Trans. Graph.*, vol. 23, pp. 271–275, Aug. 2004. [Online]. Available: http://doi.acm.org/10.1145/1015706.1015714

[11] T. Hamada, T. Narumi, R. Yokota, K. Yasuoka, K. Nitadori, and M. Taiji, "42 TFlops hierarchical n-body simulations on GPUs with applications in both astrophysics and turbulence," in *Proc. Conf. High Perform. Comput. Netw. Storage Anal.*, 2009, pp. 62:1–62:12. [Online]. Available: http://doi.acm.org/10.1145/1654059.1654123

[12] T. Ishiyama, K. Nitadori, and J. Makino, "4.45 pflops astrophysical n-body simulation on k computer: The gravitational trillion-body problem," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2012, pp. 5:1–5:10. [Online]. Available: http://dl.acm.org/citation.cfm?id=2388996.2389003

[13] A. Rahimian *et al.*, "Petascale direct numerical simulation of blood flow on 200k cores and heterogeneous architectures," in *Proc. ACM/IEEE Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2010, pp. 1–11. [Online]. Available: http://dx.doi.org/10.1109/SC.2010.42

[14] O. Ergul, "Parallel implementation of MLFMA for homogeneous objects with various material properties," *Prog. Electromagn. Res.*, vol. 121, pp. 505–520, 2011.

[15] V. Melapudi, B. Shanker, S. Seal, and S. Aluru, "A scalable parallel wideband MLFMA for efficient electromagnetic simulations on large scale clusters," *IEEE Trans. Antennas Propag.*, vol. 59, no. 7, pp. 2565–2577, Jul. 2011.

[16] B. Michiels, J. Fostier, I. Bogaert, and D. De Zutter, "Performing large full-wave simulations by means of a parallel MLFMA implementation," in *Proc. IEEE Antennas Propag. Soc. Int. Symp.*, 2013, pp. 1880–1881.

[17] M.-L. Yang, B.-Y. Wu, H.-W. Gao, and X.-Q. Sheng, "A ternary parallelization approach of mlfma for solving electromagnetic scattering problems with over 10 billion unknowns," *IEEE Trans. Antennas Propag.*, vol. 67, no. 11, pp. 6965–6978, Nov. 2019.

[18] W. Chew, E. Michielssen, J. M. Song, and J. M. Jin Eds., *Fast and Efficient Algorithms in Computational Electromagnetics*. Norwood, MA, USA: Artech House, Inc., 2001.

[19] B. Dembart and E. Yip, "A 3D fast multipole method for electromagnetics with multiple levels," in *Proc. 11th Annu. Conf. Appl. Comput. Electromagn.*, 1995, pp. 621–628.

[20] J. M. Song, C. C. Lu, and W. C. Chew, "MLFMA for electromagnetic scattering by large complex objects," *IEEE Trans. Antennas Propag.*, vol. 45, no. 10, pp. 1488–1493, Oct. 1997.

[21] E. Michielssen, A. Boag, and W. C. Chew, "Scattering from elongated objects: Direct solution in o (n log2n) operations," *IEE Proc.-Microw. Antennas Propag.*, vol. 143, no. 4, pp. 277–283, 1996.

[22] V. Jandhyala, E. Michielssen, B. Shanker, and W. C. Chew, "A combined steepest descent fast multipole algorithm for the fast analysis of three-dimensional scattering by rough surfaces," *IEEE Trans. Geosci. Remote Sens.*, vol. 36, no. 3, pp. 738–749, May 1998.

[23] L. Greengard, J. Huang, V. Rokhlin, and S. Wandzura, "Accelerating fast multipole methods for the helmholtz equation at low frequencies," *IEEE Comput. Sci. Eng.*, vol. 5, no. 3, pp. 32–38, jul.–sep. 1998.

[24] J.-S. Zhao and W. C. Chew, "Integral equation solution of maxwells equations from zero frequency to microwave frequencies," *IEEE Trans. Antennas Propag.*, vol. 48, no. 10, pp. 1635–1645, Oct. 2000.

[25] M. Vikram, H. Huang, B. Shanker, and T. Van, "A novel wideband FMM for fast integral equation solution of multiscale problems in electromagnetics," *IEEE Trans. Antennas Propag.*, vol. 57, no. 7, pp. 2094–2104, Jul. 2009.

[26] M. Vikram, B. Shanker, and S. Aluru, "Provably scalable parallel FMM algorithm for multiscale electromagnetic simulations," in *Proc. IEEE Antennas Propag. Soc. Int. Symp.*, 2009, pp. 1–4.

[27] S. Hughey, H. M. Aktulga, M. Vikram, M. Lu, B. Shanker, and E. Michielssen, "Parallel wideband MLFMA for analysis of electrically large, nonuniform, multiscale structures," *IEEE Trans. Antennas Propag.*, vol. 67, no. 2, pp. 1094–1107, Feb. 2019.

[28] N. A. Gumerov and R. Duraiswami, *Fast Multipole Methods for the Helmholtz Equation in Three Dimensions*. Amsterdam, Netherlands: Elsevier, 2005.

[29] N. A. Gumerov and R. Duraiswami, "A broadband fast multipole accelerated boundary element method for the three dimensional helmholtz equation," *J. Acoust. Soc. Amer.*, vol. 125, no. 1, pp. 191–205, 2009.

[30] S. Wandzuraz, "The fast multipole method for the wave equation: A pedestrian prescription," *IEEE Antennas Propag. Mag.*, vol. 35, no. 3, pp. 7–12, Jun. 1993.

[31] J. M. Song and W. C. Chew, "Multilevel fast-multipole algorithm for solving combined field integral equations of electromagnetic scattering," *Microw. Opt. Technol. Lett.*, vol. 10, no. 1, pp. 14–19, 1995.

[32] O. Ergül and L. Gürel, "Fast and accurate analysis of large-scale composite structures with the parallel multilevel fast multipole algorithm," *JOSA A*, vol. 30, no. 3, pp. 509–517, 2013.
[33] B. Michiels, J. Fostier, I. Bogaert, and D. De Zutter, "Weak scalability analysis of the distributed-memory parallel MLFMA," *IEEE Trans. Antennas Propag.*, vol. 61, no. 11, pp. 5567–5574, Nov. 2013.
[34] L. Ying, G. Biros, and D. Zorin, "A kernel-independent adaptive fast multipole algorithm in two and three dimensions," *J. Comput. Phys.*, vol. 196, no. 2, pp. 591–626, 2004.
[35] H. Sundar, R. S. Sampath, and G. Biros, "Bottom-up construction and 2: 1 balance refinement of linear octrees in parallel," *SIAM J. Sci. Comput.*, vol. 30, no. 5, pp. 2675–2708, 2008.
[36] E. Agullo, B. Bramas, O. Coulaud, E. Darve, M. Messner, and T. Takahashi, "Task-based FMM for multicore architectures," *SIAM J. Sci. Comput.*, vol. 36, no. 1, pp. C66–C93, 2014.
[37] R. Wang, C. Chen, J. Lee, and E. Darve, "PBBFMM3D: A parallel black-box fast multipole method for non-oscillatory kernels," 2019, *arXiv:1903.02153*.
[38] O. Ergül, "Solutions of large-scale electromagnetics problems involving dielectric objects with the parallel multilevel fast multipole algorithm," *JOSA A*, vol. 28, no. 11, pp. 2261–2268, 2011.
[39] O. Ergul and L. Gurel, "Accurate solutions of extremely large integral-equation problems in computational electromagnetics," *Proc. IEEE*, vol. 101, no. 2, pp. 342–349, Feb. 2013.
[40] M. Vikram, C. Knowles, B. Shanker, and L. Kempel, "An ultra-wideband FMM for multi-scale electromagnetic simulations," in *Proc. 27th Annu. Rev. Prog. Appl. Comput. Electromagn.*, 2011, pp. 564–569.
[41] C. Waltz, K. Sertel, M. Carr, B. C. Usner, J. L. Volakis, and Others, "Massively parallel fast multipole method solutions of large electromagnetic scattering problems," *IEEE Trans. Antennas Propag.*, vol. 55, no. 6, pp. 1810–1816, Jun. 2007.
[42] J. M. Taboada, M. G. Araujo, F. O. Basteiro, J. L. Rodríguez, and L. Landesa, "MLFMA-FFT parallel algorithm for the solution of extremely large problems in electromagnetics," *Proc. IEEE*, vol. 101, no. 2, pp. 350–363, Feb. 2013.
[43] J. K. Salmon, "Parallel hierarchical n-body methods," PhD dissertation, Division Phys. Math. Astron. Caltech, California Inst. Technol., Pasadena, CAs, 1991.
[44] S. Velamparambil, J. Song, and W. C. Chew, "On the parallelization of electrodynamic multilevel fast multipole method on distributed memory computers," in *Proc. Int. Workshop Innov. Archit. Future Gener. High-Perform. Processors Syst.*, 2000, pp. 3–11.
[45] W. C. Chew, E. Michielssen, J. Song, and J.-M. Jin, *Fast and Efficient Algorithms in Computational Electromagnetics*. Norwood, MA, USA: Artech House, Inc., 2001.
[46] Ö. Ergül and L. Gürel, "Hierarchical parallelisation strategy for multilevel fast multipole algorithm in computational electromagnetics," *Electron. Lett.*, vol. 44, no. 1, pp. 3–5, 2008.
[47] R. J. Chien and B. K. Alpert, "A fast spherical filter with uniform resolution," *J. Comput. Phys.*, vol. 136, pp. 580–584, 1997.
[48] J. Sarvas, "Performing interpolation and anterpolation by the fast fourier transform in the 3D multilevel fast multipole algorithm," *SIAM J. Numer. Anal.*, vol. 41, pp. 2180–2196, 2003.
[49] C. Cecka and E. Darve, "Fourier-based fast multipole method for the helmholtz equation," *SIAM J. Sci. Comput.*, vol. 35, no. 1, pp. A79–A103, 2013.
[50] M. P. Lingg, S. M. Hughey, and H. M. Aktulga, "Optimization of the spherical harmonics transform based tree traversals in the helmholtz FMM algorithm," in *Proc. 47th Int. Conf. Parallel Process.*, 2018, pp. 1–11.
[51] B. Michiels, J. Fostier, I. Bogaert, P. Demeester, and D. De Zutter, "Towards a scalable parallel MLFMA in three dimensions," in *Proc. Comput. Electromagn. Int. Workshop*, 2011, pp. 132–135.
[52] B. Michiels, J. Fostier, I. Bogaert, and D. De Zutter, "Full-wave simulations of electromagnetic scattering problems with billions of unknowns," *IEEE Trans. Antennas Propag.*, vol. 63, no. 2, pp. 796–799, 2015.
[53] M. Warren and J. Salmon, "A parallel hashed Oct-Tree N-body algorithm," in *Proc. ACM/IEEE Conf. Supercomputing*, 1993, pp. 12–21.
[54] L. Greengard, J. Huang, V. Rokhlin, and S. Wandzura, "Accelerating fast multipole methods for the helmholtz equation at low frequencies," *IEEE Comput. Sci. Eng.*, vol. 5, no. 3, pp. 32–38, Third Quarter 1998.
[55] R. Coifman, V. Rokhlin, and S. Wandzura, "The fast multipole method for the wave equation: A pedestrian prescription," *IEEE Antennas Propag. Mag.*, vol. 35, no. 3, pp. 7–12, Jun. 1993.
[56] A. A. Ergin, B. Shanker, and E. Michielssen, "Fast evaluation of three-dimensional transient wave fields using diagonal translation operators," *J. Comput. Phys.*, vol. 146, no. 1, pp. 157–180, 1998.
[57] A. A. Ergin, B. Shanker, and E. Michielssen, "The plane-wave time-domain algorithm for the fast analysis of transient wave phenomena," *IEEE Antennas Propag. Mag.*, vol. 41, no. 4, pp. 39–52, Aug. 1999.
[58] M. Abduljabbar *et al.*, "Extreme scale FMM-accelerated boundary integral equation solver for wave scattering," *SIAM J. Sci. Comput.*, vol. 41, no. 3, pp. C245–C268, 2019.
[59] R. Yokota and L. A. Barba, "A tuned and scalable fast multipole method as a preeminent algorithm for exascale systems," *Int. J. High Perform. Comput. Appl.*, vol. 26, no. 4, pp. 337–346, 2012.
[60] M. S. Warren and J. K. Salmon, "Astrophysical N-body simulations using hierarchical tree data structures," in *Proc. ACM/IEEE Conf. Supercomputing*, pp. 570–576, vol. 92, 1992.