PVFMM: A parallel kernel independent FMM for particle and volume potentials

Dhairya Malhotra* and George Biros

The University of Texas at Austin, Austin, TX 78712

Abstract. We describe our implementation of a parallel fast multipole method for evaluating potentials for discrete and continuous source distributions. The first requires summation over the source points and the second requiring integration over a continuous source density. Both problems require $\mathcal{O}(N^2)$ complexity when computed directly; however, can be accelerated to $\mathcal{O}(N)$ time using FMM. In our PVFMM software library, we use kernel independent FMM and this allows us to compute potentials for a wide range of elliptic kernels. Our method is high order, adaptive and scalable. In this paper, we discuss several algorithmic improvements and performance optimizations including cache locality, vectorization, shared memory parallelism and use of coprocessors. Our distributed memory implementation uses space-filling curve for partitioning data and a hypercube communication scheme. We present convergence results for Laplace, Stokes and Helmholtz (low wavenumber) kernels for both particle and volume FMM. We measure efficiency of our method in terms of CPU cycles per unknown for different accuracies and different kernels. We also demonstrate scalability of our implementation up to several thousand processor cores on the Stampede platform at the Texas Advanced Computing Center.

AMS subject classifications: 31-04, 35J05, 65Y05, 65Y20

Key words: fast multipole method, N-body problems, potential theory.

Program Summary

Program title: Parallel Volume Fast Multipole Method (PVFMM)

Nature of problem: solving constant coefficient elliptic PDEs and N-body problems

Software licence: GPL 3.0

CiCP scientific software URL:

Distribution format: .gz

Programming language(s): C/C++

*Corresponding author. *Email addresses:* dhairya.malhotra@gmail.com, gbiros@acm.org

Computer platform: Any

Operating system: Unix-like operating systems

Compilers: icpc, g++

RAM: at least 1GB

External routines/libraries: MPI, OpenMP, BLAS, LAPACK, FFTW3

Running time:

Restrictions:

Supplementary material and references: Software homepage, http://pvfmm.org

Additional Comments:

1 Introduction

Many problems in physics and engineering require computing potentials for discrete or continuous source distributions. In this paper, we focus on potentials related to constant coefficient elliptic PDEs. Applications include computing gravitational interactions in astrophysics [12, 18, 24], fluid flows [7, 12, 17], electro-magnetic and acoustic scattering [3,22] and many others [6,10,11]. Particle N-body problems also arise from discretization of boundary integral representation [21, 28]. Volume potentials are required for solution of elliptic PDEs. Several applications discretize volume potential problems in to particle N-body problems with mollification of the kernel function [17]. However, these methods are low order. Our implementation uses a high order adaptive method of [15], which is designed for computing volume potentials. The method uses piecewise Chebyshev polynomials on an adaptive octree to represent the continuous source distribution and also the final potential. Our particle and volume FMM is based on the kernel independent formulation of [26] and this allows us to solve problems for a wide range of kernels corresponding to constant coefficient elliptic PDEs. In this paper, we present results for Laplace, Stokes and low-frequency Helmholtz (wavenumber < 20). We support free-space and periodic boundary conditions. Our method can also be used to solve variable coefficient elliptic PDEs by solving the integral formulation using iterative linear solvers. The details of that work can be found in [20].

Features We now summarize the most important features of our method:

- Our method is kernel independent and supports kernel functions for several different constant coefficient elliptic PDEs.
- We modified the original Kernel Independent FMM (KIFMM) formulation to use backward stable pseudoinverse and this allows us to achieve better accuracy.
- The volume FMM scheme is adaptive and high order accurate.

- The piecewise polynomial representation of potential, for volume FMM, allows us to evaluate the potential at arbitrary points and also compute derivatives of the potential.
- We support computation in both single and double precision.
- Our implementation is highly optimized and uses cache optimized algorithms for far-field computations and vectorized kernel functions for near interactions.
- We support shared memory parallelism using OpenMP and distributed parallelism using MPI. Our volume FMM implementation can also utilize NVIDIA GPU or Phi coprocessor on heterogeneous architectures.

Related work The original particle FMM was first introduced in [9] for Laplace kernel. The method has since been optimized [2] and extended to other kernel functions in [6], [7] and [11]. The earliest distributed memory algorithms for particles were presented in [24] and [25]. They introduced the concepts of local essential tree and space-filling curves which are now used in many implementations including ours. The original KIFMM was presented in [26] and parallel implementations were presented in [27] and [16]. Other algorithms for general kernel functions include [8] and black-box FMM of [5]. There has been extensive work on optimizing the FMM. Our distributed memory FMM follows closely on our previous work on particle FMM [16]. Other scalable implementations of particle N-body codes include [28], [12], [14] and [13]. The importance of blocking to explore locality was also discussed in [1]. An efficient implementation for far-field interactions in black-box FMM was discussed in [23]; however, this approach worked well only for uniform particle distributions, using single precision computation on GPUs.

The volume FMM was first presented in [4] for 2D and then in [15] for 3D. We extended this work in [19], by including new algorithmic optimizations, vectorization, support for coprocessors and distributed memory parallelism.

Organization of the paper In §2, we explain the algorithms for our particle and volume FMM. We also discuss modifications to the original algorithm, by using backward stable pseudoinverse to improve the accuracy and convergence of the method. Then, in §3, we briefly discuss several performance optimizations and distributed memory parallelism. In §4, we explain the steps for compiling our software and running the code. In §5, we present convergence and scalability results for our method. Finally, in §6 we conclude with a summary of our contributions.

2 Methodology and Algorithms

In this section, we first introduce the particle N-body problem and briefly describe the basic idea behind the fast multipole method in this context. We discuss the kernel independent variant [26] for particle N-body problems. Finally, we describe the modifications

K	kernel function
y, q	source: coordinates, density
x, u	target: coordinates, potential
N_{pt}	number of source, target points
N	number of unknowns in target potential
$\mathcal{T}$	octree
$\mathcal{B}$	tree node (octant)
N_{leaf}	number of leaf octants
$\mathcal{N}(\mathcal{B})$	near region of $\mathcal{B}$
$\mathcal{F}(\mathcal{B})$	far region of $\mathcal{B}$: $\Omega \backslash \mathcal{N}(\mathcal{B})$
m	multipole order: number of points on an edge of equivalent surface
q	maximum degree of Chebyshev polynomials
f	source density function
ϵ_{tree}	tolerance for adaptive refinement

Table 1: Notation used in this paper for N-body problems, tree data structure, particle and volume FMM.

to the particle FMM, for evaluating volume potentials. In Table 1, we summarize notation used this section. We will introduce and explain the notation in detail as we discuss these concepts in the following section.

N-body problem We are given a set of N_{pt} source and target points. For each source point, we have its coordinates $y_j \in \mathbf{R}^3$ and its source density q_j. For each target point, we have its coordinates x_i and the unknown potential u_i. The potential u_i at target points x_i is given by the sum:

$$u_i = \sum_{j=1}^{N_{pt}} K(x_i, y_j) q_j, \quad \forall i = 1, \cdots, N_{pt}$$

where, K is called the kernel function. Computing this sum directly requires $\mathcal{O}(N_{pt}^2)$ time. With FMM, we can evaluate this sum in $\mathcal{O}(N_{pt})$ time.
To solve the N-body problem with FMM, we first partition the domain using a tree data structure $\mathcal{T}$ (Figure 1). For each target point $x_i \in \mathcal{B}$, we compute interactions from all other tree nodes in $\mathcal{T}$. In FMM, we split these interactions into two parts, near interactions and far interactions:

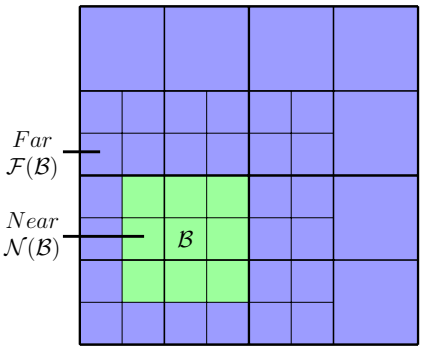

Figure 1: Near $\mathcal{N}(\mathcal{B})$ and far $\mathcal{F}(\mathcal{B})$ interaction nodes for a target node $\mathcal{B}$.

$$u_i = \sum_{y_j \in \mathcal{N}(\mathcal{B})} K(x_i, y_j) q_j + \sum_{y_j \in \mathcal{F}(\mathcal{B})} K(x_i, y_j) q_j$$

The near interactions are computed exactly through direct summation, by adding contributions from each source point $y_j \in \mathcal{N}(\mathcal{B})$. The tree nodes further away from $\mathcal{B}$ are

called *well-separated* from $\mathcal{B}$.

Interactions from source points in a well-separated tree node to target points in $\mathcal{B}$, are low rank and can be approximated. Furthermore, instead of computing interactions at the leaf level, far-field interactions in FMM are computed hierarchically at the coarsest possible length scale. In Figure 2, we show far-field interactions for a target node $\mathcal{B}$. At the finest level, we compute interactions to $\mathcal{B}$ from other source nodes $\mathcal{B}_s \in \mathcal{V}(\mathcal{B})$ which are not well-separated from $\mathcal{P}(\mathcal{B})$ (parent of $\mathcal{B}$). Similarly, at the next coarser level we compute interactions to $\mathcal{P}(\mathcal{B})$ from source nodes $\mathcal{B}_s \in \mathcal{V}(\mathcal{P}(\mathcal{B}))$ and so on for all ancestors of $\mathcal{B}$. Finally, we combine the contributions from all ancestors of $\mathcal{B}$ to obtain the far-field potential at target points in $\mathcal{B}$.

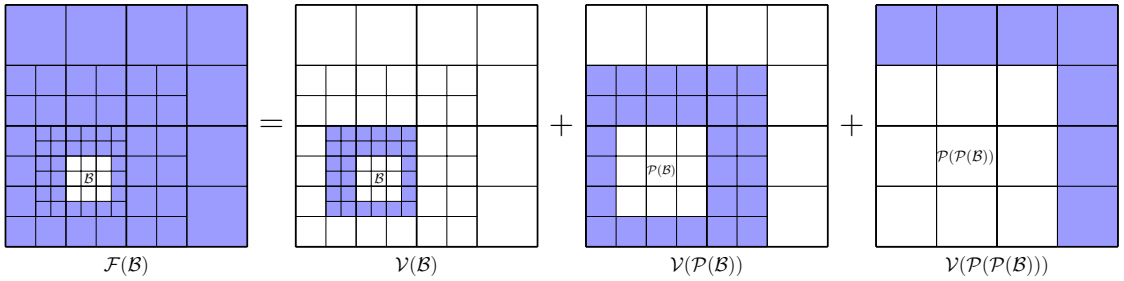

Figure 2: Far interactions broken into parts evaluated hierarchically at different levels in the tree.

2.1 Kernel Independent FMM

Here we summarize the important features of the Kernel Independent FMM. In particular, we discuss the far-field translation operator, which is the defining feature of KIFMM. We first discuss the two basic building blocks for the FMM in the context of KIFMM: the multipole expansion and the local expansion. The far-field interactions will then be defined by translation operators between these expansions.

Multipole Expansion For a tree node $\mathcal{B}$, the multipole expansion (Figure 3) approximates the far-field potential due to the source points within $\mathcal{B}$. In KIFMM, we evaluate the potential $u^{u,\mathcal{B}}$ from these source points at points $x^{u,\mathcal{B}}$ on a check surface. We then compute a set of densities $q^{u,\mathcal{B}}$ for points $y^{u,\mathcal{B}}$ on an equivalent surface by solving a linear system.

$$u_i^{u,\mathcal{B}} = \sum_{y_j \in \mathcal{B}} K(x_i^{u,\mathcal{B}}, y_j) q_j, \quad \forall i$$

Then, the potential at a point well-separated from $\mathcal{B}$ can be evaluated by computing the potential due to these equivalent sources. In 3D, the points $y^{u,\mathcal{B}}$ and $x^{u,\mathcal{B}}$ are arranged in regular grids on cubic surfaces (equivalent and check surface) centered on $\mathcal{B}$. There are $m \times m$ points on each face of the cube, where m is the multipole order and it determines

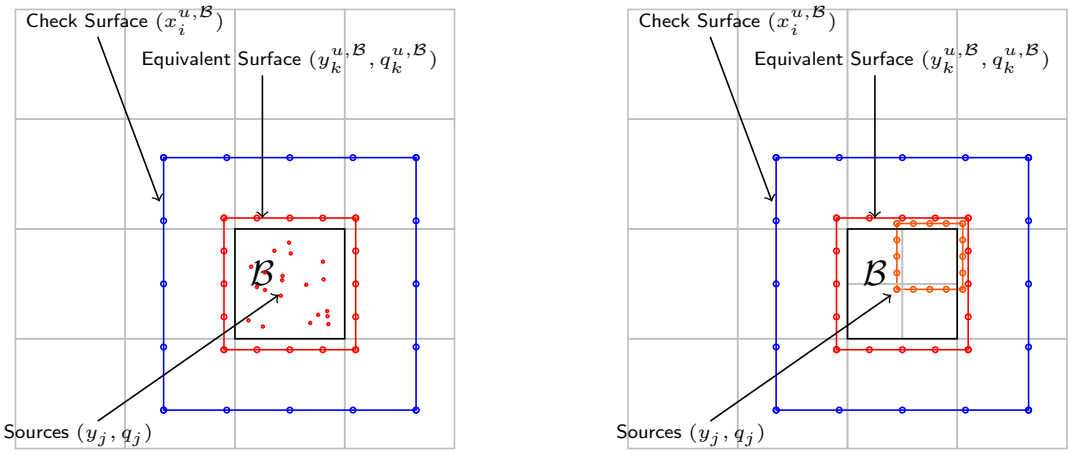

Figure 3: Left: Multipole expansion of a leaf octant computed directly from source points. Right: Multipole expansion of a non-leaf octant computed from the upward-equivalent density of its children.

the accuracy of the multipole expansion. The edge length of the equivalent surface (s_e) and check surface (s_c) must satisfy the relation $s < s_e < s_c < 3s$, where, s is the edge length of $\mathcal{B}$. The ratios s_e/s and s_c/s are determined empirically and are set to 1.05 and 2.95 respectively in our implementation.

Local Expansion For a tree node $\mathcal{B}$, the local expansion (Figure 4) approximates the potential at points in the interior of $\mathcal{B}$ due to the source points well-separated from it. We evaluate the potential $u^{d,\mathcal{B}}$ from well-separated source points at points $x^{d,\mathcal{B}}$ on a check surface. We then compute a set of densities $q^{d,\mathcal{B}}$ for points $y^{d,\mathcal{B}}$ on an equivalent surface by solving a linear system.

$$u_i^{d,\mathcal{B}} = \sum_{y_j \in \mathcal{B}} K(x_i^{d,\mathcal{B}}, y_j) q_j, \quad \forall i$$

Then, the potential at a point in the interior of $\mathcal{B}$ can be evaluated by computing the potential due to these equivalent sources. As for multipole expansion, the points $y^{d,\mathcal{B}}$ and $x^{d,\mathcal{B}}$ are arranged in regular grids on cubic surfaces (equivalent and check surface) centered on $\mathcal{B}$, with $m \times m$ points on each face. The edge length of the equivalent surface (s_e) and check surface (s_c) must satisfy the relation $s < s_c < s_e < 3s$, where, s is the edge length of $\mathcal{B}$. The ratios s_c/s and s_e/s are determined empirically and are set to 1.05 and 2.95 respectively in our implementation.

We now define five translation operators for computing far-field interactions using multipole and local expansions.

- *S2M translation.* For a leaf node, we compute the check potential directly from its

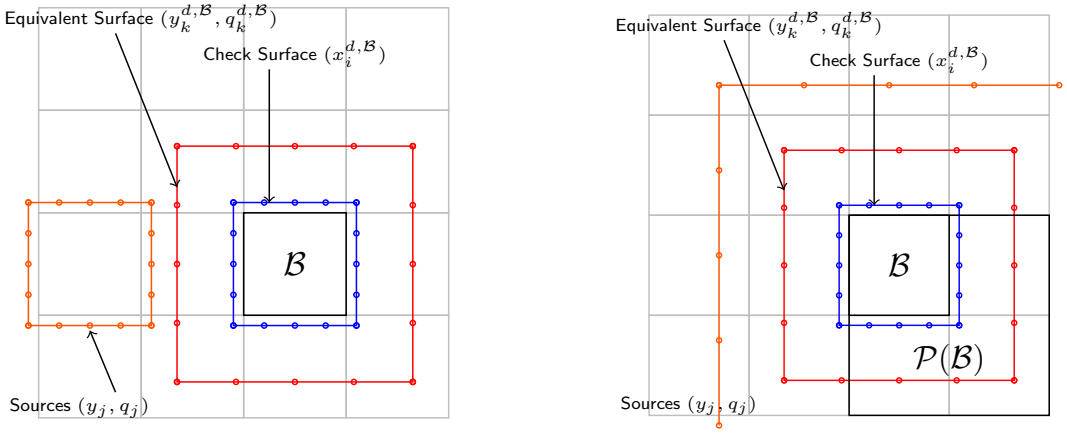

Figure 4: Left: Local expansion from upward-equivalent source distribution of a well-separated octant. Right: Local expansion from downward-equivalent source distribution of the parent octant.

source points and then compute its multipole expansion by solving a linear system as discussed above (Figure 3: left).

- *M2M translation.* For a non-leaf node, we compute the check potential by evaluating the multipole expansion of each of its children and summing the result. We then compute its multipole as discussed earlier (Figure 3: right).
- *M2L translation.* For a target node $\mathcal{B}_t$ we compute contributions from a well-separated source node $\mathcal{B}_s$, by evaluating the multipole expansion of $\mathcal{B}_s$ at the check surface for $\mathcal{B}_t$ and computing its local expansion by solving a linear system (Figure 4: left).
- *L2L translation.* For a node $\mathcal{B}$, we add contribution from the local expansion of its parent $\mathcal{P}(\mathcal{B})$ by evaluating the local expansion of $\mathcal{P}(\mathcal{B})$ at the check surface of $\mathcal{B}$ and then computing the local expansion of $\mathcal{B}$ as discussed before (Figure 4: right).
- *L2T translation.* For a leaf-node $\mathcal{B}$, we evaluate far-field component of the potential at its target points by evaluating its local expansion.

An outline of the steps in the FMM algorithm is presented below. We refer to steps 2,3 and 4 as the downward pass. Figure 5 shows the interactions in upward and downward passes for a quadtree.

1. *Upward pass.* For all leaf nodes compute S2M translation. Then, for all non-leaf nodes compute M2M translations in a post-order traversal of the tree.
2. *Far interactions.* For all nodes, compute M2L translations from well separated source nodes.

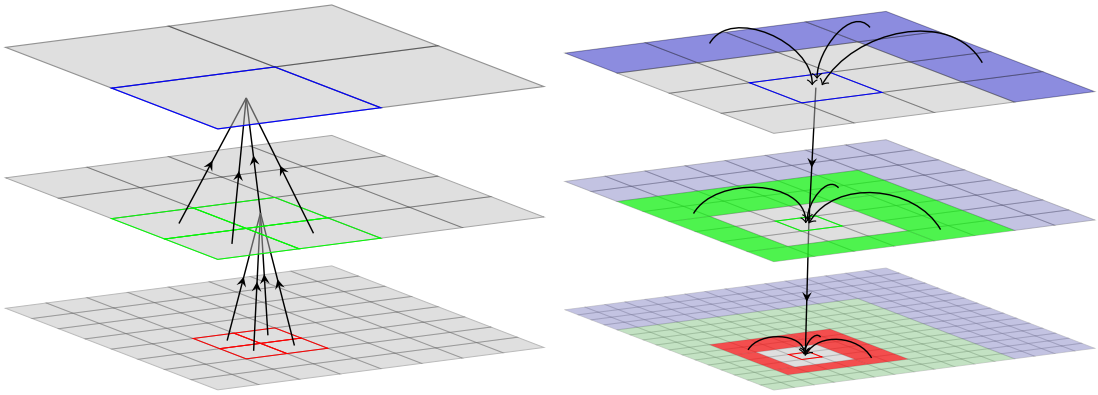

Figure 5: Upward Pass: constructing multipole expansions and Downward Pass: constructing local expansions, evaluating near interactions.

3. *L2L and L2T interactions.* For all nodes compute L2L translations in pre-order traversal of the tree. Then, for all leaf nodes, evaluate the local expansion to get the target potential.

4. *Near interactions.* For each leaf node, compute direct interactions from other leaf nodes which are not well-separated from it and add the result to the target potential.

FFT Acceleration of M2L interaction Each M2L translation involves computing the downward-check potential for a node $\mathcal{B}$ from the multipole expansion of a node well-separated from it. Each surface has $\mathcal{O}(m^2)$ points and the translation requires $\mathcal{O}(m^4)$ complexity. Since the points are on a regular grid, this operation can be treated as a convolution in three dimensions. The convolution converts into a complex Hadamard product in Fourier space. This reduces the computational cost to $\mathcal{O}(m^3 \log m)$ when Fourier transform and its inverse are computed using FFT and IFFT. The FFT and IFFT need to be computed only once for each tree node. The details can be found in [26].

Backward Stable Pseudo-inverse In the original KIFMM, when computing the multipole expansion (or the local expansion) from the check potential, the linear system was solved by computing a pseudo-inverse i.e. by computing a singular value decomposition (SVD), inverting the diagonal matrix with appropriate regularization and then multiplying the factors together. However, it is an ill-conditioned linear system and therefore, with finite precision arithmetic, we lose precision when computing the equivalent density. Consequently, for the original FMM, we could only achieve about 9-digits of accuracy in double precision.

There are two source of errors when computing the equivalent density. The first is from the round-off error (due to finite precision) in the check potential which gets amplified by a factor equal to the condition number of the matrix. The second error source

is from multiplying the factors together to form the pseudo-inverse, since the diagonal matrix has very large and very small numbers. It turns out that the error from check potential, which is amplified by the condition number of the matrix, is also damped by by a similar factor when computing the far-field potential from the equivalent density. Therefore, this component of error does not affect the final accuracy of FMM. The error introduced when forming the pseudo-inverse can be avoided by storing the inverse as its factors. The diagonal matrix can be multiplied with one of the two orthonormal matrices and therefore, we only need to store two factors. This is the basic idea behind the backward stable pseudo-inverse. With this modification, we can now achieve about 14-digits of accuracy with KIFMM in double precision.

2.2 Volume FMM

In this section we describe how the kernel independent FMM can be adapted to evaluate potential due a to continuous source distribution f defined on a cubic domain Ω. Instead of summation, as for particle FMM, we now need to evaluate integrals over the computational domain Ω. Similar to particle FMM, we partition the domain using an octree and evaluate interactions over each leaf node in the tree. As before, we distinguish between near and far (well-separated) octants:

$$u(x) \quad = \int_{\mathcal{N}(x)} K(x-y)f(y) + \int_{\mathcal{F}(x)} K(x-y)f(y)$$

The far-field interactions (over $\mathcal{F}(x)$) are approximated using multipole and local expansions and near interactions (over $\mathcal{N}(x)$) are evaluated exactly. Since the kernel function has a singularity at the origin, this leads to singular and near-singular integrals which are costly to evaluate.

Next, we will discuss the octree data structure used to represent the source density function f. We will also describe efficient computation of singular and near-singular integrals on this data structure. A detailed discussion of the volume FMM can be found in [15] and [19].

Tree construction We discretize the continuous source distribution f, using a piecewise polynomial representation. We discretize the domain using an octree and use Chebyshev polynomials at each leaf node. We evaluate the given density function f at the Chebyshev nodes in each leaf node and then use a tensor product rule to compute Chebyshev coefficients and retain coefficients up to degree q. At each leaf node, the function f can now be evaluated at a point (x_1, x_2, x_3) as follows,

$$\hat{f}(x_1, x_2, x_3) = \sum_{i,j,k \geq 0}^{i+j+k \leq q} \alpha_{i,j,k} T_i(x_1) T_j(x_2) T_k(x_3) \tag{2.1}$$

where, $\alpha_{i,j,k}$ are the Chebyshev coefficients, $T_i(x)$ is the Chebyshev polynomial of degree i in x. For adaptive mesh refinement, we start with a uniform octree, and compute the absolute sum of the highest order coefficients for each tree node. This gives an estimate of the polynomial truncation error for each node. For tree nodes with a truncation error larger than a specified tolerance ϵ_{tree}, we further subdivide the nodes until we achieve the desired tolerance or reach the maximum allowed depth for the tree. For volume FMM, the final potential u is also represented by piecewise polynomials on this octree data-structure, by first evaluating the potential at the Chebyshev points of each leaf node and then computing the Chebyshev coefficients.

Singular Integration With a polynomial representation of f at a leaf node $\mathcal{B}$, we can compute the potential from $\mathcal{B}$ at any point in the domain Ω. This lets us compute the multipole expansion of a leaf node, by computing the potential at points on the check surface, and also compute near interactions by evaluating the potential at the Chebyshev node points of adjacent tree nodes. However, since the kernel function K is singular, computing potential at points close to or within $\mathcal{B}$ can be very expensive. To compute interactions efficiently, we precompute integrals I_i^x over Chebyshev basis functions T_i for each evaluation point x relative to the tree node. Then, the potential at a point x from a tree node with Chebyshev coefficients α_i is given by the sum, $u(x) = \sum_i \alpha_i^{\mathcal{B}} I_i^x$. Therefore, we can form matrices and compute S2M translations and near interactions between leaf nodes through matrix-vector products.

To limit the memory needed for these precomputed quadratures, we set a maximum allowed depth for the tree and also enforce a 2:1 balance constraint on the tree nodes i.e. adjacent leaf nodes can not differ by more than one level. To further reduce memory overhead, we note that kernel functions are translation invariant and therefore these quadratures need to be computed for only one octant at each level in the octree. For many problems, kernel function is also scale invariant (e.g. Laplace and Stokes), and for these we need to precompute for only one level in the tree. The kernel functions also have directional symmetry and this further reduces the memory requirement.

3 Performance Optimizations

In this section we mention several important optimizations and discuss parallel implementation of our method. For a detailed discussion of these concepts, we refer the interested readers to [19]. We first discuss intra-node optimizations in §3.1 and then discuss distributed memory parallelism in §3.2.

3.1 Intra-Node Optimizations

Modern computing architecture is extremely complex, with multiple levels of cache, multi-core CPUs, non-uniform memory access, vectorization and coprocessors. Extracting maximum performance requires a considerable amount of programming effort and

often requires redesigning of the algorithms. In this section, we discuss these intra-node optimizations which we have included in our implementation.

Near Interactions In particle FMM, computing near interactions requires evaluating the kernel functions. This is one of the most expensive parts of the FMM algorithm. In our code, we have developed highly optimized implementations for Laplace, Stokes and Helmholtz kernels. We offer single and double precision implementation with both SSE and AVX vectorized versions for these kernels. For Laplace and Stokes kernels, evaluating inverse square root operation is the most expensive part of the computation. This operation has high latency and low throughput on current architectures. In our implementation, we use the fast approximate inverse square root instruction to compute $x \approx a^{-1/2}$. Then, for higher accuracy, we perform a Newton iteration $x = 0.5x(3 - ax^2)$. The scaling by 0.5 is deferred until after computing the sum over all source points. With these optimizations we achieved about $3.8\times$ speedup for double precision with Laplace kernel. For the Helmholtz kernel we use Intel SVML library, when available, for evaluating sin and cos functions.

For the volume FMM, near interactions require computing matrix-vector products. The interaction matrices depend only on the relative position of the source and target tree nodes. Therefore, we group several interaction in the same direction and compute all interactions with a matrix-matrix product. In addition, several interaction matrices are related due to directional symmetry of the kernel function. Therefore, we also group interactions in different directions together. This approach has better data locality and gives higher performance. To compute these matrix-matrix products, we use BLAS libraries which are vectorized and highly tuned. Therefore, we can achieve close to the peak theoretical performance.

Far-field Interactions Each M2L interaction requires computing a Hadamard product and this has low arithmetic intensity when computed naively. The unoptimized implementation was memory bound and achieved just 5% of the peak floating point performance. In our implementation, instead of computing interaction between two octants at a time, we compute all 8×8 interactions between two groups of sibling octants. We interleave sibling data and this allows us to vectorize the computation. We also use spatial locality of interactions to further improve data locality and arithmetic intensity. Our implementation has been carefully tuned and vectorized using AVX and SSE vector intrinsics. These algorithmic improvements in addition to using shared memory parallelism with OpenMP have improved performance by over an order of magnitude. On Intel Sandy Bridge architecture, we achieve over 50% of the peak theoretical performance for far-field computation, for both particle and volume FMM.

Asynchronous Execution on Coprocessors In our volume FMM, we have added support for NVIDIA GPUs (using CUDA) and Intel Phi coprocessors. We use task parallelism

to compute near interactions on coprocessors and far interactions on the CPU. All computations and memory transfers between the coprocessor and the host are asynchronous and overlapped with computation on CPU.

3.2 Distributed Memory Parallelism

The PVFMM library supports distributed memory parallelism using MPI. In this section, we discuss parallel tree data structures for particle and volume FMM and also discuss the communication step in parallel FMM.

Parallel Tree Construction We use space-filling curve to partition data across processors. For particle FMM, we start with a set of coordinates for source and target points and compute the Morton id for each particle coordinate. We sort the particle coordinates and the associated source density by Morton id and partition the data uniformly across processors. Each processor constructs its local octree and then we remove duplicate leaf octants at processor boundaries. This gives a distributed octree with leaf octants sorted by their Morton id.

For volume FMM, we start with a uniform octree partitioned between processors as described above. We refine this octree adaptively. If, at any stage in the adaptive refinement, we observe a significant load imbalance, then we repartition this list of Morton id sorted leaf nodes and continue adaptive refinement. We have also developed a very efficient parallel algorithm for 2:1 balance refinement. Our original algorithm did not load balance during the refinement process and redistributed load as a post processing step. For highly adaptive trees, this resulted in very large load imbalance and this affected scalability. Our new algorithm is more robust to such load imbalance.

FMM Communication In the parallel FMM algorithm, at the end of the upward pass, we need to communicate ghost octants so that the source densities and multipole expansions of these ghost octants are available to the user processes. Once this communication completes, the resulting tree is called a local essential tree and has all the necessary data for downward pass of the FMM algorithm. To communicate these ghost octants, we use the hypercube communication scheme of [16]. This scheme requires only point-to-point communication of octree node data and has been shown to be scalable to several thousand processors. For an uncongested network, and a uniform octree, the communication cost is expected to scale as $\mathcal{O}((N/p)^{2/3}\log p)$ for p processors and N unknowns.

4 Software Installation and Numerical Examples

The latest version of this library can be obtained from the project webpage (`http://pvfmm.org`). After downloading and extracting the source code, the configure script must be executed to configure the package for the target system. In most cases, the configure

Error	maximum relative error in particle potential
$\|\|e_f\|\|_\infty$, $\|\|e_u\|\|_\infty$	relative error in source density and potential
T_{Tree}	tree construction time
T_{Setup}	setup time
T_{FMM}	FMM evaluation time
T_{All}	total solve time $\approx T_{\text{Tree}} + T_{\text{Setup}} + T_{\text{FMM}}$
$cycles/N$	number of CPU cycles per unknown

Table 2: Notation for convergence and performance results in §5

script will detect the compiler and the libraries needed to build the library. If it is unable to find a library, it will display options with which the user must specify the location of the library. Our code requires at least an MPI C++ compiler, with OpenMP, FFTW3, BLAS and LAPACK libraries. For GPU support, nvcc compiler and cuBLAS library are needed. Additional compiler options or library paths can be specified to the configure script. These details can be found in the INSTALL file. Once the configure script completes, the library can be compiled by running make.

Several examples are provided in the examples directory. These can be compiled by running 'make all-examples' on the command line. Executing these examples lists the arguments needed by them. In the scripts directory, we provide scripts to generate convergence and scalability results for particle and volume FMM. We present these results in the following section.

5 Numerical Results and Performance Analysis

We present numerical results to demonstrate convergence of our method in §5.1. We also demonstrate single node (shared-memory) performance for our code. Then, in §5.2 we demonstrate scalability of our method. In Table 2, we summarize the notation used in this section.

Hardware We use Stampede and Maverick platforms at Texas Advanced Computing Center (TACC) for our experiments.

Stampede is a Linux cluster with 6400 compute nodes connected by a Mellanox FDR InfiniBand network in a fat-tree topology. Each node has two 8-core Intel Xeon E5-2680 running at 2.7GHz and has 32GB of memory. In addition, each node also has one Intel Xeon Phi SE10P coprocessor. The peak theoretical double-precision performance for each node is about 1.4TFLOPS.[†] There are also 16 large memory nodes with 32 cores and 1TB of memory per node.

Maverick is a 132 node Linux cluster with a Mellanox FDR InfiniBand interconnect. Each node has two 10-core Intel Xeon E5-2680 v2 running at 2.8GHz, 256GB of memory

[†] 345.5GFLOPS for the CPU and 1.1TFLOPS for the coprocessor.

and an NVIDIA Tesla K40 GPU.

Software We use the Intel compiler version 14.0.1.106, with Intel MPI library version 4.1.3.049 to compile our code. We compile our code with -O3 optimization level and with -mavx option for AVX vectorization. We also use Intel SVML to vectorize the Helmholtz kernel. In addition, we have used FFTW3/3.3.2 and Intel MKL libraries. For the GPU runs, we used the NVIDIA compiler version 6.0 to compile the CUDA code and cuBLAS library for GEMM on GPU.

5.1 Convergence Results and Single Node performance

All results in this section were obtained on a single node of Stampede, using one MPI process and 16 OpenMP threads. For most results, we have used a regular (16-core, 32GB) node. However, for some results we needed to use the large memory node, while using only 16-cores and with KMP_AFFINITY=compact. In the results, we indicate wherever we have used a large memory node.

N_{pt}	N_{leaf}	m	Error	T_{All}	$cycles/N_{pt}$	T_{Tree}	T_{Setup}	T_{FMM}(GFLOPS)
1E+6	1.8E+4	4	5E-04	0.51	2.2E+4	0.19	0.14	0.15 (232)
	1.5E+4	6	5E-06	0.57	2.5E+4	0.18	0.11	0.25 (246)
	4.1E+3	10	7E-09	1.26	5.4E+4	0.16	0.06	1.01 (174)
	2.0E+3	16	2E-13	2.77	1.2E+5	0.15	0.09	2.49 (161)
8E+6	1.5E+5	4	5E-04	5.09	2.7E+4	2.20	1.24	1.24 (235)
	1.2E+5	6	4E-06	5.52	3.0E+4	2.13	0.92	2.08 (247)
	3.3E+4	10	6E-09	11.66	6.3E+4	2.15	0.34	8.71 (173)
	1.5E+4	16	1E-13	24.80	1.3E+5	2.09	0.24	22.02 (158)

Table 3: Convergence with increasing multipole order m for Laplace kernel with uniform particle distribution. The timing results are for 16 cores on a single node of Stampede.

We first show convergence for the particle FMM. As mentioned earlier, for particle FMM, the near interactions are computed exactly and the accuracy of the far-field interactions is determined by the multipole order m. In our results, we report the maximum relative error (estimated by computing direct interactions for a subset of the target particles) as we increase the multipole order. For each case the source and the target particles coincide and the source densities are generated from a uniform random distribution in the interval (-0.5, 0.5). We use two kinds of particle distributions: *uniform distribution* with particles distributed uniformly in a cube and *highly non-uniform distribution* with particles on the surface of an ellipsoid, distributed uniformly over the azimuthal and polar angles. The ellipsoid has a pole-to-pole distance of 0.9 and the equatorial cross-section is a circle of diameter 0.225.

In Table 3 and Table 4, we report results for Laplace kernel with uniform and non-uniform particle distributions respectively. In each table we report two sets of results, first

N_{pt}	N_{leaf}	m	Error	T_{All}	$cycles/N_{pt}$	T_{Tree}	T_{Setup}	T_{FMM}(GFLOPS)
1E+6	3.1E+4	4	1E-04	0.55	2.4E+4	0.23	0.14	0.15 (267)
	2.1E+4	6	2E-06	0.57	2.5E+4	0.21	0.10	0.24 (285)
	1.8E+4	10	8E-10	1.32	5.7E+4	0.22	0.11	0.96 (156)
	6.2E+3	16	2E-13	2.94	1.3E+5	0.19	0.12	2.61 (143)
8E+6	2.1E+5	4	2E-04	4.91	2.7E+4	2.31	1.06	1.13 (280)
	1.7E+5	6	1E-05	5.31	2.9E+4	2.21	0.84	1.83 (284)
	1.5E+5	10	6E-10	12.13	6.6E+4	2.58	0.92	8.11 (154)
	4.3E+4	16	4E-14	25.00	1.4E+5	2.30	0.37	21.82 (140)

Table 4: Convergence results for Laplace kernel with a highly non-uniform particle distribution. For $m = 4$, the octree is refined to 12 levels for 1E+6 particles and 14 levels for 8E+6 particles.

N_{pt}	N_{leaf}	m	Error	T_{All}	$cycles/N_{pt}$	T_{Tree}	T_{Setup}	T_{FMM}(GFLOPS)
1E+6	3.1E+4	4	6E-04	0.87	3.8E+4	0.23	0.15	0.44 (268)
	2.1E+4	6	4E-06	1.19	5.1E+4	0.21	0.11	0.81 (274)
	1.3E+4	12	6E-10	7.48	3.2E+5	0.25	0.30	6.84 (137)
	6.2E+3	16	2E-12	12.01	5.2E+5	0.25	0.73	10.96 (127)
8E+6	2.1E+5	4	3E-03	7.70	4.2E+4	2.51	1.09	3.39 (273)
	1.7E+5	6	8E-05	10.76	5.8E+4	2.47	0.89	6.70 (272)
	9.4E+4	12	1E-09	61.18	3.3E+5	2.79	1.02	56.44 (136)
	4.3E+4	16	6E-12	94.99	5.1E+5	2.64	1.14	90.30 (125)

Table 5: Convergence results for Stokes kernel with a highly non-uniform particle distribution.

with 1E+6 particles and in the second with 8E+6 particles. In each case as we increase the multipole order m, we observe spectral convergence in maximum relative error, almost up to machine precision. We observe similar results for Stokes and Helmholtz kernels in Table 5 and Table 6 respectively. The Helmholtz problem corresponds to a wavenumber of ten. To get meaningful results, the distance between points on the check surfaces at the coarsest scale in the tree must resolve the oscillatory part of the Helmholtz kernel and therefore, we start with a larger multipole order. Ideally, we should use a larger multipole order only at the coarsest levels of the octree; however, this feature is currently not supported in our implementation.

In all results, we also report the total wall-time to obtain the solution T_{All}, and the number of CPU cycles per particle. The cycles per particle is computed as $T_{\text{All}} \times 2.7\text{GHz} \times 16 cores/N_{pt}$ and this provides an estimate of the efficiency of the algorithm for a given accuracy, independent of the problem size and CPU frequency. For a particular kernel and fixed multipole order, the cycles per particle is relatively independent of the problem size. Comparing Table 3 and Table 4, we observe that the cycles per particle is also

N_{pt}	N_{leaf}	m	Error	T_{All}	$cycles/N_{pt}$	T_{Tree}	T_{Setup}	T_{FMM}(GFLOPS)
1E+6	2.7E+4	8	6E-05	2.50	1.1E+5	0.21	0.13	2.11 (122)
	1.9E+4	10	2E-06	3.10	1.3E+5	0.19	0.15	2.72 (137)
	1.4E+4	12	3E-09	9.67	4.2E+5	0.22	0.50	8.89 (63)
	6.2E+3	16	8E-14	22.46	9.7E+5	0.30	2.17	19.92 (44)
8E+6	2.0E+5	8	3E-04	20.37	1.1E+5	2.44	1.06	16.35 (124)
	1.6E+5	10	4E-05	25.80	1.4E+5	2.37	0.97	21.90 (147)
	1.0E+5	12	7E-09	78.14	4.2E+5	3.04	1.33	72.94 (63)
	4.3E+4	16	2E-13	172.11	9.3E+5	3.49	3.28	164.39 (44)

Table 6: Convergence results for Helmholtz kernel with wavenumber 10 and a highly non-uniform particle distribution. The results for the shaded rows are computed on a large memory node of Stampede using 16 processor cores.

relatively independent of particle distribution.

We also report detailed breakdown of the time spent in different stages of the algorithm. We report the tree construction time T_{Tree}. This includes the time to sort the source particles and the associated source densities by their Morton id and then construct the octree with a prescribed maximum number of particles per octant. This stage is dominated by the particle sort time and depends on the number of particles and the size of the associated density data. Across tables, we note that for 1E+6 particles, T_{Tree} is about $0.2s$ and for 8E+6 particles, it is about $2s$. We also report the setup time T_{Setup}, which involves pre-allocating memory buffers and determining interaction lists for the FMM algorithm. Finally, we report the time (T_{FMM}) spent in the FMM algorithm and the FLOP rate for this stage. At the end of the FMM algorithm, we redistribute the target potentials to the original ordering of the particles and this stage is less than 5% of the total time and is not reported in these results; however, it is included in the total time T_{All}. In all cases, we choose the number of particles per octant to achieve the minimum T_{FMM}. Results with accuracy less than 1E-7 were computed in single precision. We observe a significant drop in the FLOP rate as we switch from single to double precision, due to the smaller SIMD vector length for double precision in the kernel implementation. In our implementation, we count each inverse square root operation as two floating-point operations, with four additional operations for each Newton iteration. We use one Newton iteration for single precision and two for double precision. Therefore, each evaluation of the Laplace kernel is counted as 16FLOP in single precision and 20FLOP in double precision. For the Helmholtz kernel, we use Intel SVML to vectorize sin and cos operations. We count each evaluation of sin and cos as one FLOP. Since evaluating these functions has lower throughput on current architectures, we achieve low FLOP rates for the Helmholtz problem. Since the Helmholtz kernel is not scale-invariant, it requires much more memory than Laplace and Stokes kernels. Therefore, for the high accuracy experiments (shaded rows in Table 6), we needed more than 32GB of memory and we had to use the large memory node on Stampede.

$\|\|e_f\|\|_\infty$	m	q	N_{leaf}	$\|\|e_u\|\|_\infty$	T_{All}	$cycles/N$	T_{Tree}	T_{Setup}	T_{FMM}(GFLOPS)
4E-05	4	8	1.8E+2	1E-04	0.16	2.4E+5	0.02	0.14	0.01 (56)
5E-07	6	10	5.1E+2	5E-06	0.25	7.4E+4	0.04	0.19	0.02 (150)
2E-09	10	13	8.5E+2	2E-08	0.51	4.7E+4	0.09	0.30	0.12 (218)
3E-12	14	15	1.2E+3	9E-12	0.96	4.3E+4	0.16	0.38	0.42 (220)
2E-14	18	17	1.9E+3	7E-14	2.35	4.8E+4	0.25	0.52	1.57 (201)

Table 7: Convergence with multipole order m for Poisson problem.

Now, we present convergence results for volume potential problems. In Table 7, we solve the Poisson problem with free-space boundary condition and $f(x) = -(4\alpha^2|x|^2 - 6\alpha)e^{-\alpha|x|^2}$ where, $\alpha = 160$, $x \in (-0.5,0.5)^3$. The solution $u(x)$ is given by $e^{-\alpha|x|^2}$. As we increase m, we also use a smaller tolerance (ϵ_{tree}) for the adaptive refinement of the Chebyshev octree and therefore, the maximum relative error ($||e_f||_\infty$) in approximating f converges with ϵ_{tree}. We choose the Chebyshev degree q to approximately match the cost of near and far-field interactions. We report the maximum relative error in the output $||e_u||_\infty$, and this shows spectral convergence with m. We also report the total time to solution T_{All} and the number of CPU cycles per unknown. We give a breakdown of the time spent in the different stages of the algorithm: the tree construction time T_{Tree}, the setup time T_{Setup} and the FMM evaluation time T_{FMM}. We also report the FLOP rates for the evaluation phase. In general, the number of CPU cycles per unknown should be smaller for the low order cases; however, at low orders, the setup time dominates and therefore the cycles per unknown is larger. It is sometimes possible to amortize the setup cost when multiple FMM evaluations are required on the same octree, such as when solving a linear system iteratively when the matrix-vector product is implemented with FMM.

$\|\|e_f\|\|_\infty$	m	q	N_{leaf}	$\|\|e_u\|\|_\infty$	T_{All}	$cycles/N$	T_{Tree}	T_{Setup}	T_{FMM}(GFLOPS)
2E-04	4	8	1.8E+2	1E-03	0.27	1.3E+5	0.02	0.22	0.02 (121)
2E-06	6	10	5.1E+2	7E-05	0.45	4.4E+4	0.05	0.27	0.13 (220)
3E-07	10	13	2.9E+2	3E-07	1.02	9.1E+4	0.08	0.59	0.35 (196)
2E-11	14	15	1.4E+3	5E-11	4.83	6.3E+4	0.18	1.17	3.48 (241)
1E-12	18	17	1.1E+3	1E-12	11.23	1.3E+5	0.25	3.32	7.66 (192)

Table 8: Convergence with multipole order m for Stokes problem.

In Table 8 we show convergence for the following Stokes problem, where we solve for the velocity field $u(x)$, with free space boundary condition in the domain $(-0.5,0.5)^3$

$$\begin{aligned} -\mu\Delta u + \nabla p &= 4L^2(5-2L|x|^2)e^{-L|x|^2}(x_3e_2 - x_2e_3), \quad \text{div}\, u = 0 \\ u(x) &= \frac{2L}{\mu}e^{-L|x|^2}(x_3e_2 - x_2e_3) \end{aligned}$$

where e_2, e_3 are unit vectors along Y and Z axes respectively, $L = 125$, and the viscosity

$\mu = 1$. The analytical solution for the velocity field $u(x)$ is used to compute the output relative error.

$\|\|e_f\|\|_\infty$	m	q	N_{leaf}	$\|\|e_u\|\|_\infty$	T_{All}	$cycles/N$	T_{Tree}	T_{Setup}	T_{FMM}(GFLOPS)
4E-04	8	6	1.8E+2	3E-03	0.11	1.7E+5	0.01	0.04	0.06 (38)
1E-08	12	10	9.6E+2	4E-08	0.99	7.8E+4	0.06	0.43	0.51 (143)
5E-13	16	13	3.6E+3	1E-12	8.86	9.5E+4	0.19	1.39	7.28 (132)
1E-13	18	14	3.6E+3	7E-14	13.57	1.2E+5	0.31	2.14	11.12 (131)

Table 9: Convergence with multipole order m for Helmholtz problem with wavenumber 10. The results for the shaded row are computed on the large memory node of Stampede using 16 processor cores.

In Table 9, we demonstrate the case of an oscillatory kernel by solving the Helmholtz equation with wavenumber 10 and free space boundary condition in the domain $(-0.5,0.5)^3$

$$\begin{aligned} \Delta u + \mu^2 u &= (4\alpha^2|x|^2 - 6\alpha + \mu^2)e^{-\alpha|x|^2} \\ u(x) &= e^{-\alpha|x|^2} \end{aligned}$$

where, $\alpha = 160$, $\mu = 20\pi$. The analytical solution $u(x)$ is used to compute the output error. As before, for particle FMM, we need to use a multipole order of 8 or larger.

5.2 Scalability Results

In Figure 6 and Figure 7 we show fixed-size or strong scalability results for Laplace and Helmholtz problems respectively with 1E+8 particles. For the Laplace problem we use particles on the surface of an ellipsoid, distributed uniformly over the space of polar and azimuthal angles. This leads to a very high density of particles at the poles of the ellipsoid and results in an octree with 18 levels of refinement. For the Helmholtz problem, we uniformly distribute particles over the surface of a sphere and the resulting octree has 9 levels of refinement. In each case we report the total processor time (computed as: wall-time $\times$ #of-cores). We also report breakdown of the total time in to the time for tree construction (T_{Tree}), the setup time (T_{Setup}), the FMM communication time ($T_{\text{FMM}_{\text{COMM}}}$), the FMM computation time ($T_{\text{FMM}_{\text{COMP}}}$) and the time to scatter the target potentials to original ordering of the particles (T_{Scatter}).

For the Laplace problem, we get perfect scalability up to 256 cores. Beyond 256 cores the communication costs begin to grow. Notice that all stages, except FMM computation, require MPI communication. The FMM computation stage scales much better than the other stages. Overall, we achieve 95$\times$ speedup which corresponds to about 37% parallel efficiency.

We observe similar results for the Helmholtz problem, except that a much larger fraction of the total time is spent in the FMM computation stage since the Helmholtz kernel is

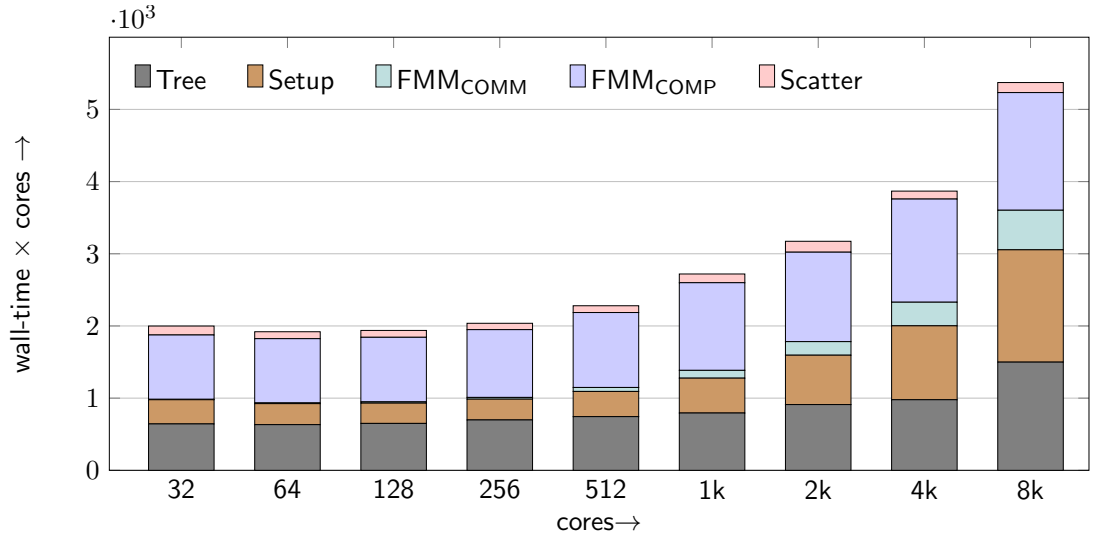

Figure 6: Strong scalability results for Laplace kernel with multipole order $m = 6$ and a highly non-uniform distribution of 1E+8 particles with 18 levels of octree refinement. We increase the number of processor cores from 32 to 8k and observe a 95× speedup and 37% parallel efficiency. We achieve 14TFLOPS of double precision performance on 8k CPU cores.

much more costly to evaluate and also because the multipole order is higher. We achieve a 60× speedup and 47% parallel efficiency.

Next, we present strong scaling results for the volume FMM in Figure 8 and Figure 9. We report the total CPU time, computed as wall-time × CPU cores, for the FMM evaluation phase (excludes tree construction and setup) and also present a breakdown of this time into the upward pass, the communication phase and the downward pass. For these results, we have also used the Phi coprocessor on each compute node. For the Laplace kernel (Figure 8), we achieve 439GFLOPS on a single compute node (16 CPU cores). This high FLOP rate is made possible due to the use of the Phi coprocessor. As we scale up to 2k CPU cores we achieve 43× speedup and 19TFLOPS of performance. We observe good scalability for the computational part of the algorithm (the downward pass); however, the communication costs become significant beyond 512 CPU cores.

In Figure 9, we present strong scaling results for the Helmholtz kernel as we scale from 64 to 4k CPU cores. We achieve 20× speedup with 31% parallel efficiency and 35TFLOPS of performance. Although we observe good performance on a small number of cores (1.7TFLOPS on 64 CPU cores), unlike for Laplace kernel, the computational phase does not scale well for the Helmholtz kernel. This is primarily due to the overhead of copying interaction matrices to the coprocessor. These precomputed interaction matrices can be several gigabytes in size and therefore cannot be stored on the coprocessor and must be copied from the host memory to the coprocessor for computing interactions at

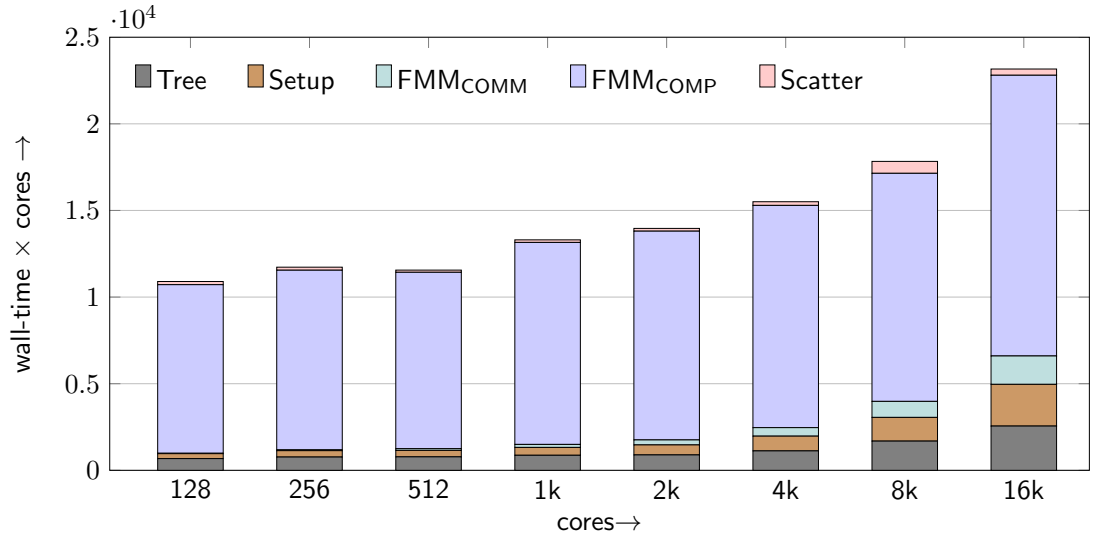

Figure 7: Strong scalability results for Helmholtz kernel (wavenumber=10) with multipole order $m = 10$ and a non-uniform distribution of 1E+8 particles with 9 levels of octree refinement. We increase the number of processor cores from 128 to 16k and observe a 60× speedup and 47% parallel efficiency. We achieve 37TFLOPS of performance on 16k CPU cores.

each level in the octree.

kernel	N	L_{max}	T_{tree}	T_{FMM}	TFLOPS
Laplace	1.8E+9	14	3.98	5.34	25.5
Laplace	7.1E+9	14	19.94	20.78	25.2
Stokes	1.8E+9	14	2.32	11.78	31.3
Stokes	6.6E+9	14	7.42	45.82	31.9
Helmholtz	5.1E+8	6	0.84	2.90	21.8
Helmholtz	4.1E+9	7	5.27	17.86	30.6

Table 10: Results for volume FMM on 32 compute nodes of Maverick using the K40 GPU on each node. We use uniform octrees for Helmholtz problem and non-uniform octrees for Laplace and Stokes. We use $m = 10$ and $q = 16$. The FLOP rates are for the FMM evaluation only.

In Table 10, we report results on Maverick using 32 compute nodes and K40 GPUs on each node. We observe high FLOP rates (∼800GFLOPS per compute node) due to the use of the GPU.

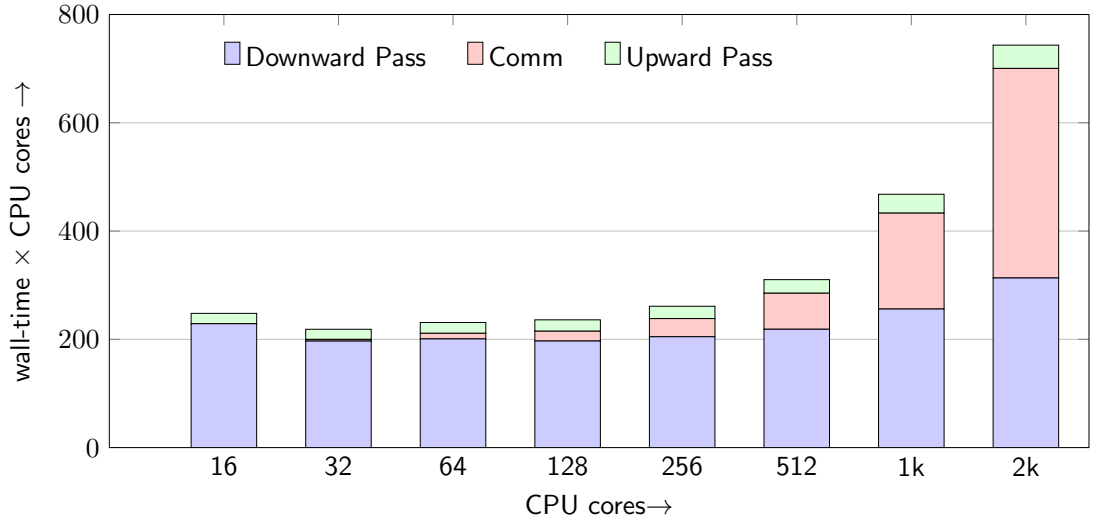

Figure 8: Strong scaling on Stampede while using Phi coprocessor for Laplace kernel with $m = 10$, $q = 13$, 2E+5 leaf octants and 16 levels of refinement for 1.13E+8 unknowns. We observe 43× speedup as we scale up to $2k$ CPU cores with 33% parallel efficiency. We achieve 19TFLOPS of performance with a wall-time of 0.36s on $2k$ cores.

6 Conclusions

We have discussed our implementation of the PVFMM software library. The method can efficiently solve N-body problems and also compute volume potentials for elliptic kernels such as Laplace, Stokes, Biot-Savart, Helmholtz (low frequency) and modified Laplace. We have discussed several performance improvements for near and far-field interactions. Our volume FMM is adaptive, high order accurate and is optimized to utilize GPU or Phi coprocessors. We have discussed the distributed memory implementation of the method. We have studied the singled node CPU performance of the method and presented results showing convergence with multipole order for both particle and volume FMM. With new modifications using backward stable pseudoinverse, we can achieve convergence up to 14-digits of accuracy. We have demonstrated scalability of our method to thousands processors on the Stampede platform at TACC.

Acknowledgments

We would like to thank Abtin Rahimian and Alex Barnett for informing us about backward stable pseudoinverse and its applicability to KIFMM for improving convergence. This material is based upon work supported by AFOSR grants FA9550-12-10484 and FA9550-11-10339; and NSF grants CCF-1337393, OCI-1029022, and OCI-1047980; and by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Com-

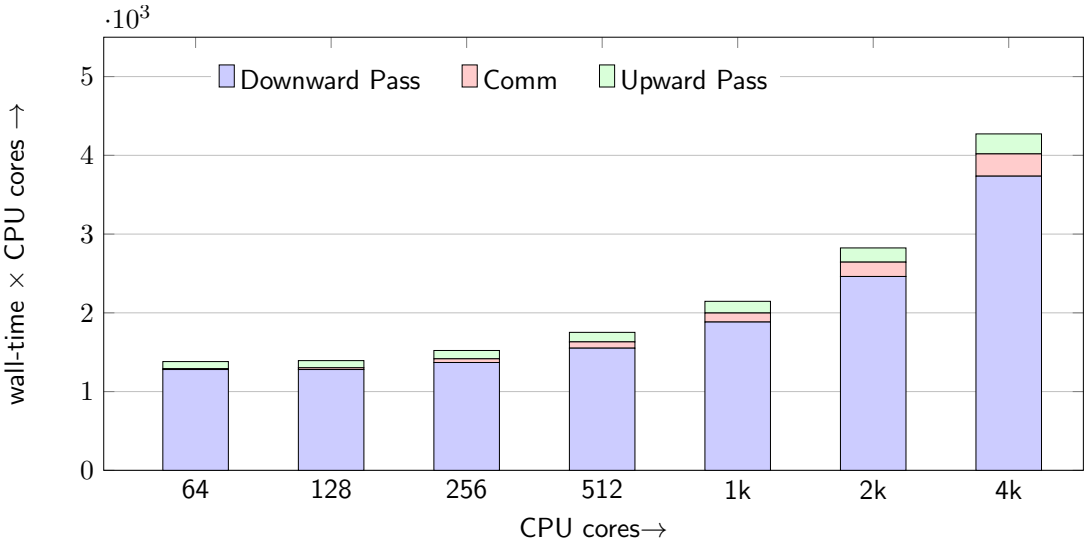

Figure 9: Strong scaling on Stampede for Helmholtz kernel with wavenumber=10, $m=10$, $q=14$, 2.6E+5 leaf octants and 6 levels of refinement for 3.6E+8 unknowns. We observe $20\times$ speedup as we scale up to $4k$ CPU cores with 31% parallel efficiency. We achieve 35TFLOPS of performance with a wall-time of 1.04s on $4k$ cores.

puting Research, Applied Mathematics program under Award Numbers DE-SC0010518, DE-SC0009286, and DE-FG02-08ER2585. Computing time on the Texas Advanced Computing Centers Stampede system was provided by an allocation from TACC and the NSF.

References

[1] Aparna Chandramowlishwaran, Kamesh Madduri, and Richard Vuduc. Diagnosis, tuning, and redesign for multicore performance: A case study of the fast multipole method. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12. IEEE Computer Society, 2010.

[2] H. Cheng, L. Greengard, and V. Rokhlin. A fast adaptive multipole algorithm in three dimensions. *Journal of Computational Physics*, 155(2):468 – 498, 1999.

[3] Björn Engquist, Lexing Ying, et al. A fast directional algorithm for high frequency acoustic scattering in two dimensions. *Communications in Mathematical Sciences*, 7(2):327–345, 2009.

[4] Frank Ethridge and Leslie Greengard. A new fast-multipole accelerated poisson solver in two dimensions. *SIAM Journal on Scientific Computing*, 23(3):741–760, 2001.

[5] William Fong and Eric Darve. The black-box fast multipole method. *Journal of Computational Physics*, 228(23):8712–8725, 2009.

[6] Yuhong Fu, Kenneth J. Klimkowski, Gregory J. Rodiny, Emery Berger, James C. Browne, James C, Jrgen K. Singer, Robert A. Van De Geijn, and Kumar S. Vemaganti. A fast solution method for three-dimensional many-particle problems of linear elasticity. *Int. J. Num. Meth. Engrg*, 42:1215–1229, 1998.

[7] Yuhong Fu and Gregory J Rodin. Fast solution method for three-dimensional stokesian many-particle problems. *Communications in Numerical Methods in Engineering*, 16(2):145–149, 2000.

[8] Zydrunas Gimbutas and Vladimir Rokhlin. A generalized fast multipole method for nonoscillatory kernels. *SIAM Journal on Scientific Computing*, 24(3):796–817, 2003.

[9] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *J. Comput. Phys.*, 73(2):325–348, December 1987.

[10] Leslie Greengard. Fast algorithms for classical physics. *Science*, 265(5174):909–914, 1994.

[11] Leslie F. Greengard and Jingfang Huang. A new version of the fast multipole method for screened coulomb interactions in three dimensions. *Journal of Computational Physics*, 180(2):642 – 658, 2002.

[12] 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 *Proceedings of SC09*, The SCxy Conference series, Portland, Oregon, November 2009. ACM/IEEE.

[13] Qi Hu, Nail A Gumerov, and Ramani Duraiswami. Scalable fast multipole methods on distributed heterogeneous architectures. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 36. ACM, 2011.

[14] Pritish Jetley, Lukasz Wesolowski, Filippo Gioachin, Laxmikant V Kalé, and Thomas R Quinn. Scaling hierarchical n-body simulations on gpu clusters. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–11. IEEE Computer Society, 2010.

[15] Harper Langston, Leslie Greengard, and Denis Zorin. A free-space adaptive fmm-based pde solver in three dimensions. *Communications in Applied Mathematics and Computational Science*, 6(1):79–122, 2011.

[16] Ilya Lashuk, Aparna Chandramowlishwaran, Harper Langston, Tuan-Anh Nguyen, Rahul Sampath, Aashay Shringarpure, Richard Vuduc, Lexing Ying, Denis Zorin, and George Biros. A massively parallel adaptive fast multipole method on heterogeneous architectures. *Communications of the ACM*, 55(5):101–109, May 2012.

[17] Keith Lindsay and Robert Krasny. A particle method and adaptive treecode for vortex sheet motion in three-dimensional flow. *Journal of Computational Physics*, 172(2):879–907, 2001.

[18] Junichiro Makino, Toshiyuki Fukushige, and Masaki Koga. A 1.349 tflops simulation of black holes in a galactic center on grape-6. In *Supercomputing, ACM/IEEE 2000 Conference*, pages 43–43. IEEE, 2000.

[19] Dhairya Malhotra and George Biros. `pvfmm`: A distributed memory fast multipole method for volume potentials, 2014. submitted.

[20] Dhairya Malhotra, Amir Gholami, and George Biros. A volume integral equation stokes solver for problems with variable coefficients. In *High Performance Computing, Networking, Storage and Analysis, SC14: International Conference for*, pages 92–102. IEEE, 2014.

[21] A. Rahimian, I. Lashuk, S.K. Veerapaneni, A. Chandramowlishwaran, D. Malhotra, L. Moon, R. Sampath, A. Shringarpure, J. Vetter, R. Vuduc, D. Zorin, and G. Biros. Petascale direct numerical simulation of blood flow on 200k cores and heterogeneous architectures. In *SC '10: Proceedings of the 2010 ACM/IEEE conference on Supercomputing*, pages 1–12, Piscataway, NJ, USA, 2010. IEEE Press.

[22] Jiming Song, Cai-Cheng Lu, and Weng Cho Chew. Multilevel fast multipole algorithm for electromagnetic scattering by large complex objects. *Antennas and Propagation, IEEE Transactions on*, 45(10):1488–1493, 1997.

[23] Toru Takahashi, Cris Cecka, William Fong, and Eric Darve. Optimizing the multipole-to-

local operator in the fast multipole method for graphical processing units. *International Journal for Numerical Methods in Engineering*, 89(1):105–133, 2012.

[24] Michael S Warren and John K Salmon. Astrophysical n-body simulations using hierarchical tree data structures. In *Proceedings of the 1992 ACM/IEEE Conference on Supercomputing*, pages 570–576. IEEE Computer Society Press, 1992.

[25] Michael S Warren and John K Salmon. A parallel hashed oct-tree n-body algorithm. In *Proceedings of the 1993 ACM/IEEE conference on Supercomputing*, pages 12–21. ACM, 1993.

[26] Lexing Ying, George Biros, and Denis Zorin. A kernel-independent adaptive fast multipole method in two and three dimensions. *Journal of Computational Physics*, 196(2):591–626, 2004.

[27] Lexing Ying, George Biros, Denis Zorin, and Harper Langston. A new parallel kernel-independent fast multipole method. In *Supercomputing, 2003 ACM/IEEE Conference*, pages 14–14. IEEE, 2003.

[28] R. Yokota, J.P. Bardhan, M.G. Knepley, LA Barba, and T. Hamada. Biomolecular electrostatics using a fast multipole bem on up to 512 gpus and a billion unknowns. *Computer Physics Communications*, 2011.