The Dissertation Committee for Dhairya Malhotra certifies that this is the approved version of the following dissertation:

Fast Integral Equation Solver for Variable Coefficient Elliptic PDEs in Complex Geometries

Committee:

George Biros, Supervisor

Bjorn Engquist

Robert A. van de Geijn

Thomas J. R. Hughes

Andreas Kloeckner

Fast Integral Equation Solver for Variable Coefficient Elliptic PDEs in Complex Geometries

by

Dhairya Malhotra

Dissertation

Presented to the Faculty of the Graduate School

of the University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

December 2017

Acknowledgments

I would like to thank my advisor Prof. George Biros for his guidance and many years of mentoring which has shaped me into the researcher that I am today. This work would not have been possible without him.

I would also like to thank my collaborators Hari Sundar, Amir Gholami, Abtin Rahimian and Denis Zorin. Special thanks to Hari, Abtin and Bryan Quaife for many insightful discussions. Many thanks to all the committee members for their helpful suggestions and comments. Thanks to all the faculty and staff at ICES for providing an amazing environment for learning and conducting research. Thanks to University of Texas at Austin and in particular ICES and TACC for providing the resources for this research.

Thanks to my brother Shaurya for teaching me to program, inspiring me to pursue science and engineering and for all the guidance over many years. Thanks to my aunt Manju for her care and support in helping me adjust to life in the US. Thanks to my parents for giving me every possible advantage to be successful and for teaching me what is important in life.

Fast Integral Equation Solver for Variable Coefficient Elliptic PDEs in Complex Geometries

by

Dhairya Malhotra, Ph.D.

The University of Texas at Austin, 2017

Supervisor: George Biros

This dissertation presents new numerical algorithms and related software for the numerical solution of elliptic boundary value problems with variable coefficients on certain classes of geometries. The target applications are problems in electrostatics, fluid mechanics, low-frequency electromagnetic and acoustic scattering. We present discretizations based on integral equation formulations which are founded in potential theory and Green's functions. Advantages of our methods include high-order discretization, optimal algorithmic complexity, mesh-independent convergence rate, high-performance and parallel scalability.

First, we present a parallel software framework based on kernel independent fast multipole method (FMM) for computing particle and volume potentials in 3D. Our software is applicable to a wide range of elliptic problems such as Poisson, Stokes and low-frequency Helmholtz. It includes new parallel algorithms and performance optimizations which make our volume FMM one of the fastest constant-coefficient elliptic PDE solver on cubic domains. We show that our method is orders of magnitude faster than other N-body codes and PDE solvers. We have scaled our method to half-trillion unknowns on 229K CPU cores.

Second, we develop a high-order, adaptive and scalable solver for volume integral equation (VIE) formulations of variable coefficient elliptic PDEs on cubic domains. We use our volume FMM to compute integrals and use GMRES to solve the discretized linear system. We apply our method to compute incompressible Stokes flow in porous media geometries using a penalty function to enforce no-slip boundary conditions on the solid walls. In our largest run, we achieved 0.66PFLOP/s on $2K$ compute nodes of the Stampede system (TACC).

Third, we develop novel VIE formulations for problems on geometries that can be smoothly mapped to a cube. We convert problems on non-regular geometries to variable coefficient problems on cubic domains which are then solved efficiently using our volume FMM and GMRES. We show that our solver converges quickly even for highly irregular geometries and that the convergence rates are independent of mesh refinement.

Fourth, we present a parallel boundary integral equation solver for simulating the flow of concentrated vesicle suspensions in 3D. Such simulations provide useful insights on the dynamics of blood flow and other complex fluids. We present new algorithmic improvements and performance optimizations which allow us to efficiently simulate highly concentrated vesicle suspensions in parallel.

Table of Contents

List of Tables

List of Figures

1 Introduction

In this dissertation, we present new numerical algorithms and related software that find applications in electrostatics, acoustic scattering, incompressible Stokes flow, porous media flow, particulate flows and complex fluids. Exemplary results from simulations obtained with software in this thesis are shown in Fig. 1.1. The mathematical models that describe these applications can be expressed as a set of partial differential equations (PDEs). Several methods based on direct discretization of the PDEs already exist; however, these discretizations are often ill-conditioned and the conditioning worsens with mesh refinement. One way to circumvent this problem is to use integral equation formulations based on potential theory.

Our work focuses on developing a new class of methods which are based on integral equation formulations. In such formulations, the resulting linear systems have bounded and mesh-independent condition numbers; they work well with high-order discretizations and are amenable to parallelization. However, integral equation formulations present a new set challenges: they require costly singular and near-singular integration and result in dense linear systems which are expensive to evaluate directly. Integral equation methods would not be practical without significant technological breakthroughs in the development of efficient quadratures and scalable numerical algorithms. In the following sections we discuss our contributions towards overcoming some of these challenges.

1.1 Overview of Methods

Here we summarize our contributions to numerical methods for elliptic PDEs, with a focus on Stokesian flows and their application to the simulations of complex fluids and porous media. The main contributions of this thesis can be briefly summarized as follows: (1) in Chapter 2, we developed a distributed-memory, high-order accurate solver for volume potentials – to our knowledge, the only one of its kind; we compared it with other state-of-the-art solvers for the Poisson problem; (2) in Chapter 3, we applied this solver to flows

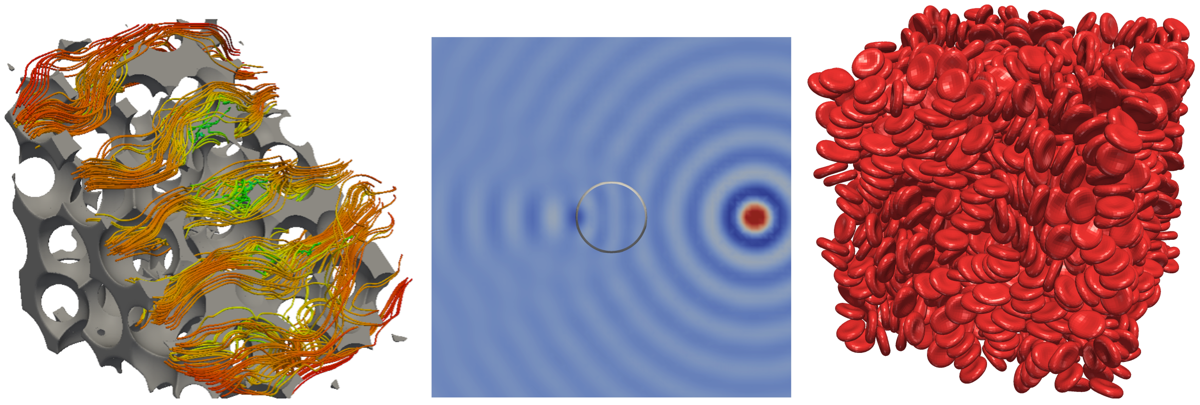

(a) *Stokes flow in complex geometry* **(b)** *Wave scattering* **(c)** *Cellular scale blood flow*

Figure 1.1 *(a) Simulation of Stokes flow in a porous media geometry. The geometry is resolved on an adaptive volumetric mesh by recursive refinement at the solid boundary (depicted in gray). We compute the solution using our volume integral equation (VIE) solver, which uses GMRES together with our parallel volume FMM (*PVFMM*) framework. (b) A cross section of the solution of variable coefficient Helmholtz problem in three dimensions with a Gaussian source near the right edge of the domain and a spherical scatterer at the center of the domain. We solve the Lipmann-Schwinger formulation using our fast VIE solver. (c) Simulation of blood flow with* 1500 *red blood cells and* 35% *hematocrit driven by a periodic Taylor-Green vortex flow. We discretize the surface of the cells using a* 16*-th order spherical harmonic basis. Our boundary integral solver uses novel singular- and near-singular quadrature schemes along with fast multipole acceleration.*

in porous media; (3) in Chapter 4, we developed a solver for problems on irregular geometries with Dirichlet and Neumann conditions; and (4) in Chapter 5, we developed a distributed-memory high-order accurate method for vesicle suspensions.

Constant-Coefficient Elliptic PDEs in the Unit Cube. For a general second order elliptic PDE, denoted by $\mathcal{L}u = f$, the solution u can be obtained by convolving f with the GreenâĂŹs function, G, of $\mathcal{L}$. The result of the convolution $u = G[f]$ is also called the volume potential induced by the density f through the kernel function G. GreenâĂŹs functions depend on boundary conditions and are rarely available. For constant-coefficient elliptic operators, the Green's function with free-space boundary conditions (also known as the fundamental solution) can be derived analytically. However, evaluating the volume potential $u = G[f]$ directly requires expensive quadratures and $\mathcal{O}\left(N^2\right)$ work for N unknowns. This has limited the applicability of integral equation based methods to very simple problems.

An efficient algorithm for computing volume potentials in three dimensions was proposed by [65]. In Chapter 2, we have extended this volume fast multipole method (FMM) by building a parallel, high-order and adaptive method for computing solutions to elliptic PDEs with free-space and periodic boundary conditions on cubic domains. We have made our method freely available in the form of the PVFMM software library[1]. Our library can compute potentials from both continuous as well as particle source density distributions. To our knowledge, this is the first and only highly parallel volume potential library currently available. Our library uses high-order piecewise polynomial discretization on adaptive octrees and is efficient up to 16-th order discretization. It uses precomputed quadratures to efficiently compute singular and near-singular integrals; has linear cost in the number of unknowns; and is applicable to several elliptic PDEs such as Poisson, Stokes and low-frequency Helmholtz.

To achieve high performance, we have developed a new cache-optimized algorithm for the multipole-to-local translation operator in kernel independent FMM. This increased the arithmetic intensity of the computation and improved performance by nearly an order of magnitude. We have also developed a new blocking algorithm for computing near interactions in volume FMM, which significantly improves performance, particularly on accelerators such as NVIDIA GPUs and Intel Xeon Phi co-processors. These novel algorithms together with vectorization and multi-threading allow us to achieve up to 60% of the peak theoretical performance on x86 architectures. Our distributed memory parallel algorithms use space-filling curves for partitioning data, a state-of-the-art parallel sorting algorithm [97, 96] and a scalable hypercube communication scheme. We present results to show that our scheme outperforms other particle fast multipole methods. Similarly, we show that our volume FMM is over an order of magnitude faster compared to a wavelet based approach. In [40], we compared our method to other parallel fast solvers for the constant-coefficient Poisson's equation. Our volume FMM was an order of magnitude faster than a high-order geometric multigrid (GMG) method for the same accuracy. In that work, we solved a Poisson problem with over half-trillion unknowns in 92s on $229K$ CPU cores.

Variable Coefficient Elliptic PDEs. In general, for variable coefficient elliptic PDEs, the Green's function is not known analytically. For such problems, we treat the variable coefficient as perturbations around a constant value and use the Green's function corresponding to this constant-coefficient value. The convolution of this Green's function with the PDE

[1]PVFMM (Parallel Volume Fast Multipole Method) software available at pvfmm.org

results in a volume integral equation (VIE). A classic example of such a formulation is the Lippmann-Schwinger equation. One property of such formulation is that, with appropriate discretization schemes, it results in linear systems with mesh-independent condition-number. However, the condition-number does depend on the magnitude of the variation in the variable coefficients of the PDE. We solve these VIEs using our volume FMM together with iterative linear solvers like GMRES. For mildly varying coefficients, accurate solutions can be obtained in only a few dozen GMRES iterations.

In Chapter 3, we use this approach to compute solutions for Stokes flow in porous media geometries (Fig. 1.1a). Such problems are difficult to tackle with boundary integral formulations due to the complexity of the solid-fluid interface and even harder with stencil-based methods due to the ill-conditioning of the underlying Stokes operator. We discretized this problem by generating a volume mesh, which was adaptively refined at the boundary between the solid phase and the fluid phase. To enforce no-slip at the solid boundary, we used a penalty formulation to force the fluid velocity to be zero in the solid phase. For the flow visualized in Fig. 1.1a, we solved a problem with 20-billion unknowns in 8 minutes on $2K$ compute nodes of the Stampede system at TACC and achieved nearly 0.66PFLOP/*s* of performance with 88% weak-scaling efficiency.

We have solved similar VIE formulations for Stokes flow with variable viscosity and for low frequency scattering problems using the Lippmann-Schwinger formulation for the Helmholtz equation as shown in Fig. 1.1b.

All the fast solvers described above are designed for elliptic PDEs defined on the unit box with either free-space or periodic boundary conditions. But what if we have more general geometries and boundary conditions? One possible solution that has appeared in the literature is to combine a volume integral formulation with a boundary integral formulation. This is a mathematically valid approach but can be computationally very expensive due to the online evaluation of nearly-singular integrals. As an alternative, in Chapter 4, we have developed a novel formulation for domains that are diffeomorphic to a cube. The methodology can be summarized as follows. We map the inhomogeneous (either constant or variable coefficient) elliptic boundary value problem (BVP) to a unit cube. This transforms the problem to a variable coefficient elliptic BVP problem on a cube. Dirichlet or Neumann boundary conditions on the cube are enforced using either method of images for scalar problems (like the Poisson equation) or boundary integrals for vector problems (like the Stokes equation). This formulation has the advantage that the quadratures for singular and near singular boundary integrals can be precomputed in the same way as we have done for volume integrals. In addition, the boundary solve can be significantly accelerated by using preconditioners constructed from precomputed factorization of boundary integral operator. We present convergence results for incompressible Stokes flow with Dirichlet boundary conditions. We show that high-order accurate solutions can be computed efficiently even for highly anisotropic geometries.

Vesicle Suspensions in Stokes Flows. Vesicles are inextensible membranes enclosing a small volume of a fluid. Studying the flow of vesicle suspensions is essential for understanding the dynamics of red blood cells (RBCs) and artificial vesicles used in drug delivery. We model such flows using a boundary integral formulation. In [86], we used this formulation to perform large scale simulations of low hematocrit (volume-fraction of RBCs) blood flow with up to 200 million RBCs and achieved a performance of 0.7PFLOP/*s* on $200K$ CPU cores.

In Chapter 5, we extend this work to allow long time-scale simulations of highly-concentrated vesicle suspensions. Such simulations require efficient algorithms for near-singular boundary integrals since the inter-vesicle distance can become arbitrarily small. To do this, we have developed an inexpensive parallel algorithm adapted from the work of [110]. Due to discretization errors, vesicles can sometimes intersect and therefore, robust methods for collision handling are required. We have developed a novel repulsion-based method to prevent vesicle collisions. In long time-scale simulations, errors can accumulate over time and this can change the area and volume of vesicles. We have developed an efficient algorithm to correct for this drift in each time step. In addition, we have also implemented an adaptive time-stepping scheme, made significant improvements to the surface re-meshing scheme, developed a faster algorithm for singular integration and used our optimized parallel kernel independent FMM implementation for computing far-field interactions. With these algorithmic improvements, we are able to simulate the flow of concentrated vesicle suspensions. Fig. 1.1c shows 1500 vesicles with 35% hematocrit in a Taylor-Green vortex flow with periodic boundary conditions. For a problem with 1.1E+5 vesicles, we achieved 43% parallel weak scaling efficiency on $16K$ x86 cores.

1.2 Related Work

Solutions to constant-coefficient elliptic BVPs can be represented as the potential induced by some distribution of source terms, either discrete particles (Dirac-delta functions) or continuous distributions. The discrete problem is solved by evaluating summations over the particles and can be computed efficiently using fast summation algorithms such as particle-mesh method [30], treecode [11] and fast multipole method (FMM) [52, 109]. The continuous problem requires evaluating integrals over the domain and can be computed efficiently using the volume FMM [33, 65]. Several parallel implementations of particle FMM have been developed and they achieve high performance on modern supercomputing architectures [60, 59, 112, 67]. Optimizations for the far-field translation operator in FMM have appeared in [27, 21, 22, 99, 98, 81]; however, they do not achieve efficiency comparable to our work. There is no other distributed memory volume FMM library similar to our PVFMM framework[75, 76]. Solutions to variable coefficient PDEs can be obtained by iteratively solving a second-kind Fredholm integral equations, where the kernel function in the integral corresponds to a constant-coefficient elliptic PDE [89]. Such formulations have been presented for Poisson, Stokes and Helmholtz problems [31, 77, 3].

Boundary integral methods have been used to solve homogeneous elliptic PDEs on complex domains [6, 58]. This requires evaluating singular and near-singular integrals using special quadratures [110]. Solutions to inhomogeneous problems can be obtained by using embedded boundary integral methods, where, the inhomogeneous equation is solved using FFT [110], volume FMM [66] or other fast solvers [80, 78, 13] on a regular domain. However, accurate representation of the density function near the domain boundary (on the cut-cells) can be problematic with regular grids. In 2D, this has been resolved by constructing a C^0 extension of the density function [5]. Our work on VIE solver for non-regular geometries remapped to a cubic domain is completely new.

Previous work on 3D vesicle simulation in unbounded domains includes [103, 104, 86, 105, 87]. In [113], boundary elements are used to model flow in confined geometries. A large scale simulation of blood flow in microfluidic devices is presented in [90]. Recent work on 2D vesicle simulations introduces collision detection, near-singular integration [83] and adaptive time-stepping [85]. Except for our own work, there is no other work on parallel boundary integral methods for simulating concentrated vesicle suspensions.

1.3 Contributions

Applicable Mathematics. We present volume integral equation (VIE) formulations for constant-coefficient elliptic PDEs with a range of boundary conditions on cubic domains. We also present VIE formulations for variable coefficient elliptic PDEs by treating the variable coefficients as perturbations around a constant value.

Our formulation for Stokes flow includes a penalty term which allows us to enforce no-slip boundary conditions without having to explicitly define the interface between the fluid and the solid walls. This is useful in situations where the interface boundary is too complex to construct boundary meshes.

We have developed new VIE formulations for Poisson and Stokes equations under coordinate transformations. It allows us to map problems on certain non-regular geometries to a cubic domain; allowing us to use fast solvers for cubic geometries. The resulting VIE has coefficients which depend on the Jacobian of the coordinate mapping. We discuss well-posedness of this formulation.

Numerical Analysis and Scientific Computation. We have developed a new high-order adaptive fast multipole code for computing particle and volume potentials. We have made important contributions towards performance optimizations and developed new parallel algorithms. Our PVFMM library is the only distributed-memory volume integral equation solver that we know of. It is scalable to over 200K CPU cores and achieves high performance on hybrid architectures. We have used this to build solvers for constant and variable coefficient elliptic PDEs.

We have developed efficient numerical algorithms for computing boundary integrals on closed surfaces that are homeomorphic to spheres. Our method uses special quadratures for singular and near-singular integrals and is parallelized using our particle FMM. We have used this to develop a scalable solver for moving boundaries embedded in a Stokesian fluid. For such simulations, we have also developed algorithms for adaptive times stepping, collision handling using repulsion and surface reparameterization.

Mathematical Modeling and Applications. Our solver for constant and variable coefficient elliptic PDEs has applications in electrostatics, fluid flows and electromagnetic and acoustic scattering. We have used our solver to model incompressible Stokes flows in porous media geometries.

We present mathematical models for the flow of vesicles in creeping flow. We simulate highly concentrated vesicle flows with periodic and free-space boundary conditions. Such flows are useful in studying the dynamics for blood flow and other complex fluids.

1.4 Organization of the Thesis

In Chapter 2, we describe our PVFMM software framework for computing particle and volume potentials. We discuss performance optimizations, scalability and comparison with other codes. In Chapter 3, we discuss our solver for volume integral equation formulations and apply it to incompressible Stokes flow in porous media geometries. In Chapter 4, we extend this to VIE formulations under coordinate transformations and use it to compute Stokes flow in non-regular geometries. In Chapter 5, we discuss our boundary integral equation solver for simulating concentrated vesicle flows. Finally, in Chapter 6, we present concluding remarks and discuss future work.

2 A Parallel Fast Multipole Algorithm for Particle and Volume Potentials

In this chapter we describe a parallel software library which we have designed to compute the potential from distributions of point sources and continuous source density distributions. Our method applies to a wide range of elliptic kernel functions. We discuss several performance optimizations which make our software efficient and scalable to large distributed memory machines. This library has been a key component in the development of other technologies discussed in later chapters.

2.1 Introduction

We consider the problem of rapidly evaluating the potential induced by a kernel function K, due to a source density distribution $f(y)$ defined at each point y on a domain $\Omega \subset \mathbf{R}^3$. The potential $u(x)$ at a target point $x \in \Omega$ is given by the integral,

$$u(x) = \int_\Omega K(x-y) f(y). \tag{2.1}$$

When the source distribution is defined by a set of N particles such that $f(y) = \sum_N f_i \delta(y - y_i)$, then the potential $u(x)$ can be computed by the sum,

$$u(x) = \sum_{j=1}^{N} K(x-y_j) f_j. \tag{2.2}$$

To evaluate the potential at N target points from a source density with N degrees-of-freedom after discretization, both problems require $\mathcal{O}\left(N^2\right)$ computation using direct methods. In addition, the kernel function (such as the fundamental solution of an elliptic partial differential equation) may be singular and therefore standard quadratures cannot be used to compute Eq. (2.1). In this chapter, we describe our PVFMM software framework which can efficiently compute such potentials for kernel functions corresponding to fundamental solutions of elliptic PDEs (such as Poisson, Stokes and low frequency Helmholtz).

This chapter is based on work that has been published in [75, 76].

In our implementation, we use the kernel independent fast multipole method (KIFMM) of [109] for computing particle potentials and we use the volume fast multipole method (FMM) of [65] for computing volume potentials. These methods reduce the computational cost of computing particle and volume potentials from $\mathcal{O}\left(N^2\right)$ to $\mathcal{O}\left(N\right)$ for a problem with N degrees-of-freedom and N unknowns. The use of KIFMM allows us to efficiently compute potentials for a wide range of elliptic kernels. The volume FMM is a high-order, adaptive method specially designed for computing volume potentials efficiently. The method uses high-order piecewise Chebyshev polynomials on an adaptive octree to represent the continuous source distribution and also the final potential. It uses precomputation to efficiently handle singular and near-singular volume integration. Our implementation support free-space and periodic boundary conditions on cubic computational domains. Our distributed memory implementation uses space-filling curves for partitioning data and a hypercube communication scheme. We also incorporate several performance optimizations including cache locality, vectorization, shared memory parallelism and use of coprocessors.

Motivation and Significance. Many problems in physics and engineering require computing potentials from discrete or continuous source distributions. Applications include computing gravitational interactions in astrophysics [106, 74, 56], fluid flows [38, 68, 56], electro-magnetic and acoustic scattering [93, 32] and many others [53, 39, 49]. Particle N-body problems also arise from discretization of boundary integral methods [86, 112]. The solution of a constant-coefficient elliptic partial PDEs can be computed by using the fundamental solution of the elliptic PDE as the kernel function in Eq. (2.1). We show an example of such an integral transform in Fig. 2.1.

Contributions. We build on previous work on the Kernel Independent FMM (KIFMM) [67, 109] for particle N-body problems and the volume FMM [65, 33] for volume potential problems. We discuss algorithmic modifications that significantly improve performance and scalability of the method.

- We present novel cache-optimized traversal schemes for the near and far interactions.
- We present integration of our method of volume potentials with coprocessors (Intel Xeon Phi and NVIDIA GPU).

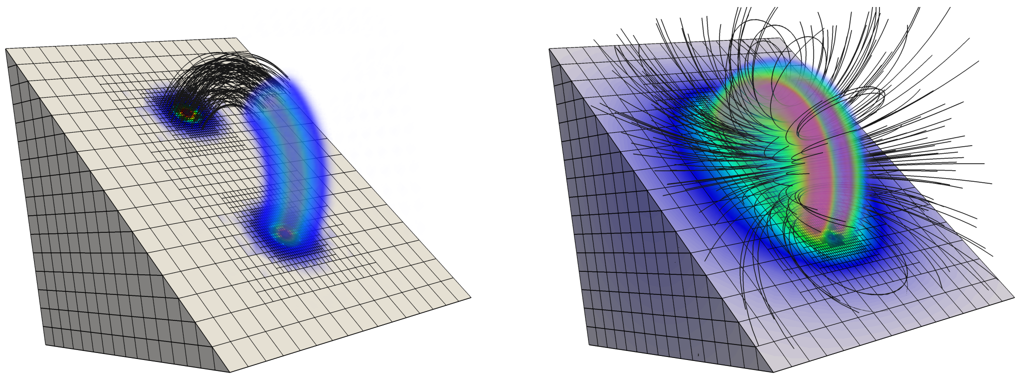


Figure 2.1 *Left: Vorticity field for a vortex-ring resolved on an adaptive Chebyshev octree is the input to volume FMM. Right: Output velocity field obtained from FMM by computing convolution of the vorticity field with the Biot-Savart kernel.*

- The single-node algorithmic refactoring and optimizations result in 7$\times$ speedup over an optimized, multithreaded implementation.
- We demonstrate the scalability of the method to several thousand cores for highly non-uniform distributions that use 25 levels of refinement.
- We modified the original Kernel Independent FMM (KIFMM) formulation to use backward stable pseudoinverse and this allows us to achieve better accuracy.

We have implemented our method in the PVFMM software library. The library is open source and can be downloaded from the library homepage (`http://pvfmm.org`). To our knowledge, this is among the fastest AMR constant-coefficient Poisson, Stokes and low-frequency Helmholtz solvers. It achieves four main algorithmic goals: high-order approximation, linear work, excellent single-node performance, and parallel scalability.

Related Work. The particle FMM was first introduced in [51] for Laplace kernel. The method has since been optimized [23] and extended to other kernel functions in [39, 38, 49]. The earliest distributed memory algorithms for particles were presented in [106, 107]. 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 [109] and parallel implementations were presented in [108, 67]. Other algorithms for general kernel functions include [44] and black-box FMM of [36]. There has been extensive work on optimizing the FMM. Our distributed memory FMM follows closely on the work of [67] on particle FMM. Other scalable implementations of particle N-body codes include [112, 56, 60, 59]. The importance of blocking to explore locality was also discussed in [21]. An efficient implementation of far-field interactions in black-box FMM was discussed in [98]; however, this approach worked well only for uniform particle distributions, using single precision computation on GPUs. None of the works on optimizing FMM performance discuss volume potentials.

A volume potential FMM was first proposed in [33] and a basic 3D shared-memory implementation (using OpenMP) was discussed in [65]. We extended this work in [76], by including new algorithmic optimizations, vectorization, support for coprocessors and distributed memory parallelism. To the best of our knowledge, there is no other work on parallel, high-order volume FMM.

There are many alternatives to using an integral equation formulation. Multigrid methods are also very effective and scalable [95] and are applicable to Stokes and low-frequency Helmholtz problems. Other scalable approaches include hybrid domain decomposition methods [73]. A very efficient Poisson solver is based on a non-iterative domain decomposition method [79] using a low-order approximation scheme. Another approach for integral equations is to use wavelet decomposition [57, 35].

Organization of the Chapter. We briefly review the kernel independent FMM in Section 2.2. We also discuss modifications to the original algorithm, by using backward stable pseudoinverse to improve the accuracy and convergence of the method. We review the volume FMM in Section 2.3. In Sections 2.4 and 2.5 we discuss our main contributions towards an optimized parallel implementation of the particle and volume FMM algorithms. Then, in Sections 2.6 to 2.8 we present results to show convergence, performance and scalability of our code. We provide comparisons with other software libraries in Section 2.9. In Table 2.1, we list some frequently used symbols for easy reference. In literature, many different abbreviations for the interaction (or translation) operators in the FMM algorithm have been used. Therefore, in Table 2.2 we provide the abbreviation used in this chapter and corresponding full descriptive name for each translation operator. Detailed definition of each translation operator can be found in [109, 65].

2.2 Kernel Independent FMM

N-body problem. We are given a set of N 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

Symbol	Definition	Symbol	Definition
K	Kernel function	$T_i(x)$	Chebyshev polynomial of degree i in x
$y,\ q$	Source: coordinates, density	f	Source density function
$x,\ u$	Target: coordinates, potential	ϵ_{tree}	Tolerance for adaptive tree refinement
N_{pt}	Number of source, target points	p	Number of processes
N	Number of unknowns in target potential	p_r	Rank of current process
$\mathcal{T}$	Octree	$\mathcal{T}_{p_r}$	Local tree of process p_r
$\mathcal{B}$	Tree node (octant)	$\mathcal{P}_u(\mathcal{B})$	User processes of $\mathcal{B}$
$\mathcal{P}(\mathcal{B})$	Parent of $\mathcal{B}$	$\mathrm{Send}(S, p_i)$	Send message S to process i
$\mathcal{N}(\mathcal{B})$	Near region of $\mathcal{B}$	$\mathrm{Recv}(R, p_i)$	Receive message R from process i
$\mathcal{F}(\mathcal{B})$	Far region of $\mathcal{B}$: $\Omega \setminus \mathcal{N}(\mathcal{B})$	t_w	Per-word transfer time
$\mathcal{L}(\mathcal{T})$	Leaf nodes in $\mathcal{T}$	t_s	Interconnect latency
N_{oct}	Number of octants	T_{Tree}	Time for tree construction
N_{leaf}	Number of leaf octants	T_{Setup}	Time for FMM setup
L_{max}	Maximum tree depth	T_{FMM}	Time for FMM evaluation
m	Order of multipole expansion	T_{All}	Total solve time $\approx T_{\text{Tree}} + T_{\text{Setup}} + T_{\text{FMM}}$
q	Degree of polynomial approximation		

Table 2.1 *Index of frequently used symbols.*

Interaction abbreviation	Description
S2M	Translation from source density to multipole expansion
S2L or X-list	Translation from source density to local expansion
S2T or U-list	Translation from source density to target potential
M2M	Translation from multipole expansion to multipole expansion
M2L or V-list	Translation from multipole expansion to local expansion
M2T or W-list	Translation from multipole expansion to target potential
L2L	Translation from local expansion to local expansion
L2T	Translation from local expansion to target potential

Table 2.2 *List of FMM translation operator abbreviations.*

is given by the sum:

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

where, K is called the kernel function. Computing this sum directly requires $\mathcal{O}\left(N^2\right)$ time. With FMM, we can evaluate this sum in $\mathcal{O}\left(N\right)$ time.

Near and Far Interactions. To solve the N-body problem with FMM, we first partition the domain using a tree data structure $\mathcal{T}$ (Fig. 2.2). For each target point x_i in the leaf node $\mathcal{B} \in \mathcal{T}$, we compute interactions from source points in every leaf nodes in $\mathcal{T}$. In FMM, we split these interactions into two parts, near interactions and far interactions:

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

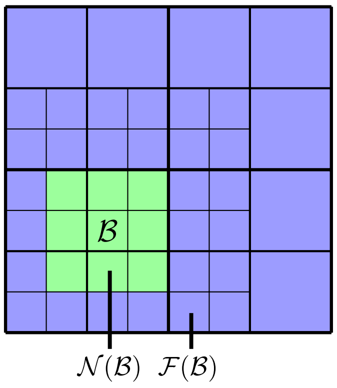


Figure 2.2 *Near $\mathcal{N}(\mathcal{B})$ and far $\mathcal{F}(\mathcal{B})$ interaction nodes for a target node $\mathcal{B}$.*

The near interactions are computed through direct summation over all source points $y_j \in \mathcal{N}(\mathcal{B})$. 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 the 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 Fig. 2.3, 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 $\mathcal{B}_s \in \mathcal{V}(\mathcal{B})$. These are the tree nodes that are well-separated from $\mathcal{B}$ but 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 $\mathcal{B}$.

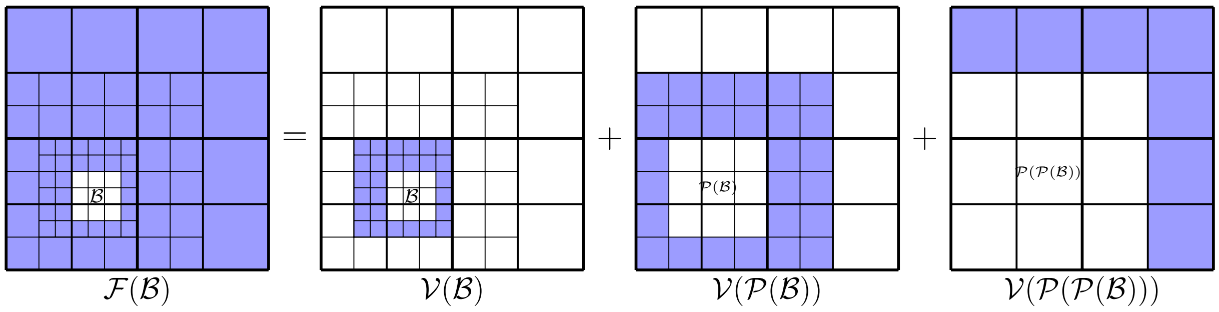
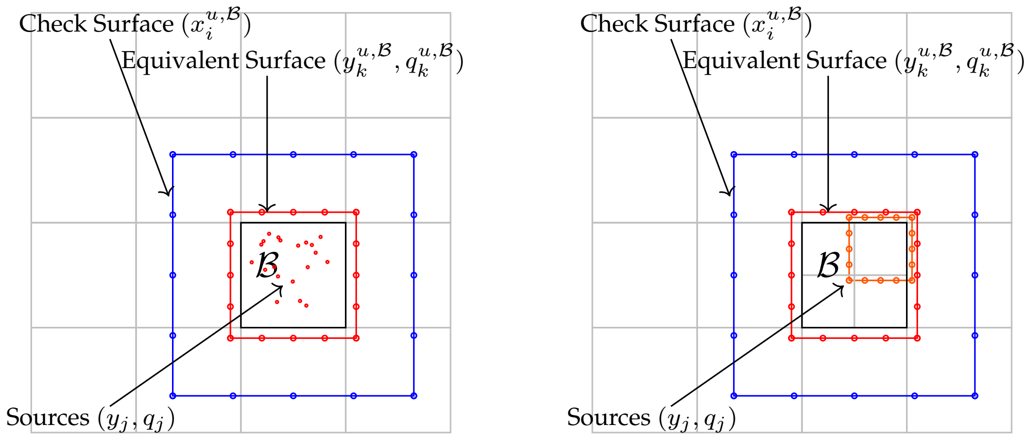

$\mathcal{B}$
$=$
$\mathcal{B}$
$+$
$\mathcal{P}(\mathcal{B})$
$+$
$\mathcal{P}(\mathcal{P}(\mathcal{B}))$
$\mathcal{F}(\mathcal{B})$
$\mathcal{V}(\mathcal{B})$
$\mathcal{V}(\mathcal{P}(\mathcal{B}))$
$\mathcal{V}(\mathcal{P}(\mathcal{P}(\mathcal{B})))$

Figure 2.3 *Far interactions broken into parts evaluated hierarchically at different levels in the tree.*

2.2.1 Far-field Interactions in KIFMM

We discuss the far-field translation operators, which is the defining feature of KIFMM. Additional details and an error analysis can be found in [109]. 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.



Check Surface ($x_i^{u,\mathcal{B}}$)
Equivalent Surface ($y_k^{u,\mathcal{B}}, q_k^{u,\mathcal{B}}$)
$\mathcal{B}$
Sources (y_j, q_j)
Check Surface ($x_i^{u,\mathcal{B}}$)
Equivalent Surface ($y_k^{u,\mathcal{B}}, q_k^{u,\mathcal{B}}$)
$\mathcal{B}$
Sources (y_j, q_j)

Figure 2.4 *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.*

Multipole Expansion. For a tree node $\mathcal{B}$, the multipole expansion (Fig. 2.4) 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 the following linear system,

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

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 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 $\mathcal{B}$.

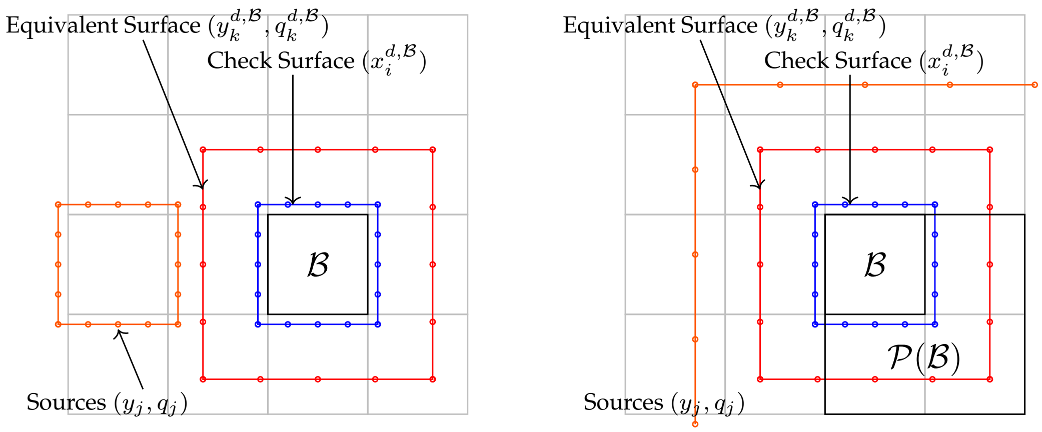

Equivalent Surface $(y_k^{d,\mathcal{B}}, q_k^{d,\mathcal{B}})$
Check Surface $(x_i^{d,\mathcal{B}})$
$\mathcal{B}$
Sources (y_j, q_j)
Equivalent Surface $(y_k^{d,\mathcal{B}}, q_k^{d,\mathcal{B}})$
Check Surface $(x_i^{d,\mathcal{B}})$
$\mathcal{B}$
$\mathcal{P}(\mathcal{B})$
Sources (y_j, q_j)

Figure 2.5 *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.*

Local Expansion. For a tree node $\mathcal{B}$, the local expansion (Fig. 2.5) 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 the following 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 $\mathcal{B}$.

Translation Operators. 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 source points and then compute its multipole expansion by solving a linear system as discussed above (Fig. 2.4: 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 (Fig. 2.4: 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 (Fig. 2.5: 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 (Fig. 2.5: 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.

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 $\mathcal{O}\left(m^2\right)$ points and the translation requires $\mathcal{O}\left(m^4\right)$ complexity. Since the points are on a regular grid, this operation can be treated as a convolution in three dimensions. In Fourier space, this convolution operation turns into a complex Hadamard product. This reduces the computational cost to $\mathcal{O}\left(m^3 \log m\right)$ 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 [109].

2.2.2 Outline of FMM

We only briefly discuss the case for uniform trees. The complete algorithm can be found in [51, 109].

An outline of the steps in the FMM algorithm is presented below. We refer to steps 2,3 and 4 as the downward pass. Fig. 2.6 shows the interactions in upward and downward passes for a quad tree.

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.

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.

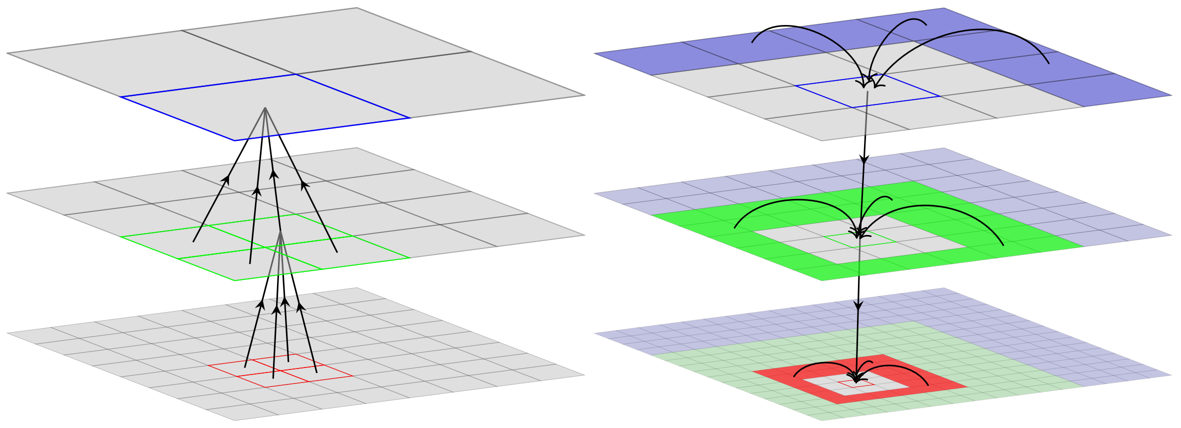


Figure 2.6 *Upward Pass: constructing multipole expansions and Downward Pass: constructing local expansions, evaluating near interactions.*

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

2.2.3 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, the original KIFMM could only achieve about 9-digits of accuracy in double precision.

The error arises from the multiplication of the factors together to form the pseudo-inverse, since the diagonal matrix has very large and very small numbers. This error can be avoided by storing the inverse in the factorized form. The diagonal matrix can be multiplied with one of the two orthonormal matrices and therefore, we only need to store two factors. For the M2M and L2L translation operators, the pseudo-inverse in multiplied with another matrix which computes the check potential. In this case, we can multiply all the matrices together to form a single matrix; however, we need to be careful of the order in which we compute the product. With this modification, we can now achieve about 14-digits of accuracy with KIFMM in double precision.

2.3 Volume FMM

The potential u at each point $x \in \Omega$ due to a continuous source distribution f defined on a cubic domain $\Omega = (0,1)^3$ is given by,

$$u(x) = \int_{\Omega} K(x-y)f(y) = \int\limits_{\mathcal{N}(x)} K(x-y)f(y) + \int\limits_{\mathcal{F}(x)} K(x-y)f(y) \tag{2.3}$$

In the volume FMM, we partition the domain using an adaptive octree. For each target evaluation point x, we have split the integral over Ω into the near interactions from $\mathcal{N}(x)$ (the set of octants containing or adjacent to x) and the far-field interactions from all the remaining octants $\mathcal{F}(x)$. The far-field interactions are approximated using multipole and local expansions. The near interactions are evaluated through direct numerical integration. Since the kernel function has a singularity at the origin, this leads to singular and near-singular integrals which are costly to evaluate. In the remainder of this section we briefly describe the volume fast multipole method. A more detailed discussion of the method can be found in [65].

2.3.1 Octree Construction

We partition the domain Ω using an octree $\mathcal{T}$ and approximate f at each leaf octant using Chebyshev polynomials of degree q. Then, at a leaf octant $\mathcal{B}$ (with coordinates mapped to $[-1,1]^3$), we have the following approximation for the density,

$$\hat{f}(x_1,x_2,x_3) = \sum_{i,j,k\geq 0}^{i+j+k\leq q} \alpha^{\mathcal{B}}_{i,j,k} T_i(x_1)T_j(x_2)T_k(x_3) \tag{2.4}$$

where, $T_i(x)$ is the Chebyshev polynomial of degree i in x. Notice that this is not a complete tensor order approximation since we truncate the expansion, so that $i+j+k \leq q$. For adaptive octrees, we specify an error tolerance ϵ_{tree} and a maximum octree depth L_{max}. We estimate the truncation error at each leaf octant by computing the absolute sum of the highest order coefficients in the Chebyshev approximation. We subdivide the leaf octants with truncation error larger than ϵ_{tree} and approximate f on each new octant. We refine recursively until the desired accuracy is achieved or we reach the maximum allowed depth L_{max}.

We also represent the volume potential solution u using piecewise Chebyshev polynomials. At each leaf octant $\mathcal{B}$, we compute the following representation for the potential,

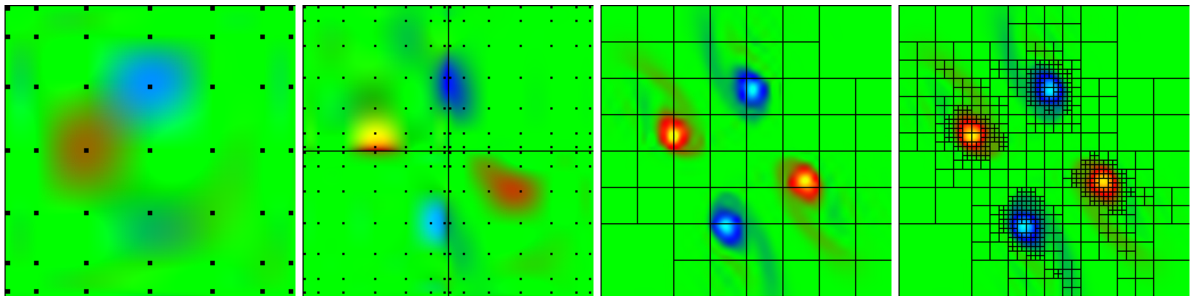


Figure 2.7 *Adaptive refinement of Chebyshev quadtree starting from the root node showing the Chebyshev node points where the input function f is sampled and refining adaptively up to six levels.*

$$\hat{u}(x_1, x_2, x_3) = \sum_{i,j,k \geq 0}^{i+j+k \leq q} \beta^{\mathcal{B}}_{i,j,k} T_i(x_1) T_j(x_2) T_k(x_3) \tag{2.5}$$

Since $u(x) \in H^1$, it is smoother than $f(x) \in L^2$. Based on this observation, we assume that the potential can be represented accurately using the same octree refinement that we used for the density.

2.3.2 Interaction Operators

To evaluate the potential in Section 2.3, we need to compute singular and near-singular integrals over the leaf octants. For a leaf octant $\mathcal{B}$ with Chebyshev approximation of the density $\hat{f}(y) = \sum_{i,j,k} \alpha^{\mathcal{B}}_{i,j,k} T_{i,j,k}(y)$, the potential at a point x is given by,

$$u(x) = \int_{y \in \mathcal{B}} K(x-y) \hat{f}(y) = \sum_{i,j,k} \alpha^{\mathcal{B}}_{i,j,k} \left[\int_{y \in \mathcal{B}} K(x-y) T_{i,j,k}(y) \right] \tag{2.6}$$

We precompute the integral term in square brackets using the method explained in Section 2.3.3. The potential from any leaf octant $\mathcal{B}$ at a point x (relative to $\mathcal{B}$) can then be evaluated using the sum $u(x) = \sum\limits_{i,j,k} \alpha^{\mathcal{B}}_{i,j,k} I^{x}_{i,j,k}$.

We precompute these quadratures (once for each level in the octree) so that source-to-multipole (S2L), source-to-target (U-list) and source-to-local (X-list) interactions in the FMM scheme can be represented as matrix-vector products. For S2M interactions, we precompute quadratures to evaluate the potential at each point on the upward-check surface of the leaf octant. For X-list interactions, for each possible direction of the target octant, we precompute quadratures to evaluate the potential at the downward-check surface of the target octant. Similarly, for U-list interactions, for each possible direction of the target octant, we precompute quadratures to evaluate the potential at the Chebyshev node points in the target octant. Then, we can construct the polynomial approximation of the potential from the values at the Chebyshev node points. We store the composition of these two operations (singular quadrature and polynomial approximation) so that the interactions are represented as translation from Chebyshev coefficients for density $\alpha^{\mathcal{B}_s}_{i,j,k}$ at the source octant $\mathcal{B}_s$, to the Chebyshev coefficients for potential $\beta^{\mathcal{B}_t}_{i,j,k}$ at the target octant $\mathcal{B}_t$.

2.3.3 Evaluating Singular Integrals

For direct interactions, we need to evaluate integrals of the following form.

$$u(r_0) = \int\limits_{\mathcal{B}} K(r - r_0)\, p(r) \tag{2.7}$$

where, $K(r - r_0)$ is the Green's function, $\mathcal{B}$ is the cubic domain of an octant and $p(r)$ is the polynomial approximation of the source density within the octant. This is a near-singular integral when r_0 is on the boundary of $\mathcal{B}$ and a singular integral when r_0 is inside $\mathcal{B}$. A simple tensor-product Gauss-quadrature rule will converge very slowly for values of r_0 within or close to $\mathcal{B}$. To evaluate such integrals, we make use of the Duffy transformation [29] followed by a tensor-product Gauss-quadrature rule.

Consider the following integral (with $K(r - r_0) = \frac{1}{|r - r_0|}$ for Poisson's equation) over a regular pyramid.

$$u = \int\limits_{0}^{1} \int\limits_{-x}^{x} \int\limits_{-x}^{x} \frac{1}{\sqrt{x^2 + y^2 + z^2}}\, p(x, y, z)\, \mathrm{d}z\, \mathrm{d}y\, \mathrm{d}x \tag{2.8}$$

We perform a change of variables ($y = u\,x,\ z = v\,x$) and transform the integration domain to a cuboid.

$$u = \int\limits_{0}^{1} \int\limits_{-1}^{1} \int\limits_{-1}^{1} \frac{x}{\sqrt{1^2 + u^2 + v^2}}\, p(x, u\,x, v\,x)\, \mathrm{d}v\, \mathrm{d}u\, \mathrm{d}x \tag{2.9}$$

This transformed equation does not have a singularity and can now be integrated using a tensor-product Gauss-quadrature rule. Moreover, the integral with respect to x can be computed exactly by choosing the order of the rule appropriately in the x-direction.

To evaluate the singular integral over a cubic domain (an octant), we partition the domain into six regular pyramidal regions with the apex of the pyramids at the singularity. The intersection of each pyramid with the volume of the cube can be represented as stacks of rectangular frustums and a smaller pyramid (Fig. 2.8). The integral over each of these

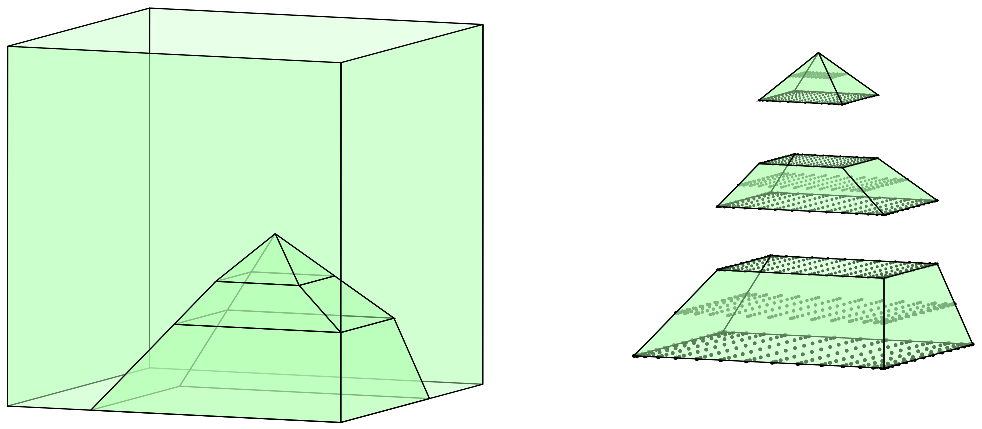


Figure 2.8 *Intersection of a regular pyramid (apex at r_0) with a cubic octant and decomposition into frustum stack and a smaller pyramid. We also show the node points for the Gauss-quadrature rule.*

components is individually evaluated using the technique described above using Duffy transformation.

2.3.4 2:1 Balance Constraint

For a general octree, the number of interaction operators that must be precomputed for the volume FMM can be very large. To limit the number of possible interaction directions, we constrain adjacent leaf octants to be within one level of each other. This is known as the 2:1 balance constraint.

We now explain the steps in our algorithm for 2:1 balance refinement (Algorithm 1). We loop from the finest to the coarsest level in the octree. In each iteration, we collect the set S of parents of all possible colleagues of non-leaf octants at that level. These are the non-leaf octants which must exist in the next coarser level for the 2:1 balance constraint to hold, and are therefore added to the set of non-leaf octants N in the next iteration. At each level, we add the set of children of the non-leaf octants N, to the final balanced octree $\widehat{\mathcal{T}}$. We use "std::set" to implement this algorithm. It allows addition, deletion and searching of octants in $O(logN)$ steps. The complexity of the overall algorithm is $T(N_{\text{oct}}) = O(N_{\text{oct}} log N_{\text{oct}})$ where, N_{oct} is the number of octants.

It is possible to improve this complexity estimate by using alternative data structures. Such as, for a pointer based tree which also maintains a list of colleagues for each octant; addition, deletion and searching of octants in the neighborhood of a given octant requires $\mathcal{O}(1)$ time. Then, we can achieve $\mathcal{O}(N_{\text{oct}})$ complexity for the algorithm.

ALGORITHM 1: SequentialBalance

```
Input: $\mathcal{T}$ unbalanced octree, $L_{max}$ maximum tree depth
Output: $\widehat{\mathcal{T}}$ balanced octree

$\widehat{\mathcal{T}} \leftarrow \emptyset$, $S \leftarrow \emptyset$;
for $i \leftarrow L_{max}$ to 1 do
    $\mathrm{N} \leftarrow S \cup \{\mathcal{B} : \ \mathcal{B} \in \mathcal{T} \setminus \mathcal{L}(\mathcal{T}), \ \ \mathrm{Level}(\mathcal{B}) = i\}$;
    $S \leftarrow \mathrm{Parent}(\mathrm{Colleagues}(\mathrm{N}))$;
    $\widehat{\mathcal{T}} \leftarrow \widehat{\mathcal{T}} \cup \mathrm{Children}(\mathrm{N})$;
end
return $\widehat{\mathcal{T}}$;
```

2.3.5 Summary of Volume FMM

In volume FMM, the source density is represented by a polynomial approximation instead of the discrete sources in classical FMM. Therefore, translations involving the source term (S2M, X and U-list) need to be modified as discussed in Section 2.3.2. In addition, we want to represent the final result by a Chebyshev interpolation. We choose the target points to be the Chebyshev node points within each leaf octant and from that we compute the polynomial approximation using L^2 projection. After precomputing all the translation operators: S2M, M2M, L2L, L2T, U,V,W and X-list for each interaction direction and for each level in the octree, the volume FMM can be summarized as follows:

- *Tree Construction*: Construct a piecewise Chebyshev approximation of the source density using octree based domain decomposition. Perform 2:1 balance refinement using Algorithm 1.

- *Upward-Pass:* For all leaf octants apply S2M translation to construct the multipole expansion. For all non-leaf octants apply M2M translations in bottom-up order, to construct multipole expansion from the multipole expansion of children.

- *Downward-Pass:* For all octants, apply V-list and X-list translations to construct the local expansion of each octant. In top-down order apply the L2L translation to all octants and add the results to their local expansions. For all leaf octants, apply L2T, W-list and U-list translations to construct the final target potential as piecewise Chebyshev interpolation.

2.3.6 Gradients

To compute gradient of the potential, we differentiate its piecewise polynomial representation obtained from the volume FMM. The gradient obtained in this way effectively has the degree $q-1$. This also leads to one-sided derivatives at octant boundaries and therefore, this may not be suitable for some applications. An alternative method is to use the gradient of the Green's function to compute U and W-list interactions and local-to-target translation. This has $3\times$ extra cost for near interaction and no extra cost for multipole-to-local interactions.

2.3.7 Selecting Optimal Parameter Values

We are interested in computing a solution to some accuracy in the least amount of time. To do this we need to select the optimal values of the three parameters: the tolerance for adaptive refinement ϵ_{tree}, the degree of Chebyshev polynomials q and the multipole order m. The parameters ϵ_{tree} and m directly control the accuracy. The value of ϵ_{tree} should be the same as required solution accuracy in L_∞ norm. Similarly, the optimal value for m is determined by the required solution accuracy and can be looked up from the convergence studies in Section 2.6.

We select the remaining parameter q to optimize for the solve time. In the following discussion we estimate the optimal value for q for a fixed number of unknowns. The cost of FMM evaluation is given by the number of interactions between the octants, weighted by the cost of each translation. For the range of q and m considered in this work, the cost of U-list and V-list interactions dominate over the cost of other translations. For a uniform octree, the total runtime is estimated by,

$$T_{\text{FMM}} \;=\; 9(q+2)^3 N\,\tau_{\text{u}} \;+\; 4\text{E+}4\,\frac{m^3}{(q+2)^3} N\,\tau_{\text{v}} \;+\; \mathcal{O}\left(m^2 N + m^4 q^{-3} N\right) \tag{2.10}$$

where, N is the number of unknowns, τ_u and τ_v are the inverse FLOP -rates for U-list and V-list interactions respectively. The parameters ϵ_{tree} and m are determined by the desired accuracy of the solution. We choose q to minimize time to solution T_{FMM}. Assuming that the FLOP -rates for U-list and V-list interactions are equal, for a fixed total number of unknowns N, we have $q \approx 4.1\sqrt{m} - 2$.

2.3.8 Reducing Memory Requirement

Even after enforcing the 2:1 balance constraint, the memory required to store translation operators can be very large. For example, Helmholtz kernel with $q = 14$ and 20 octree levels requires 38GB of memory for storing U-list interaction matrices. We now describe how this memory usage can be reduced.

Scale-invariant Kernels. Several kernel functions are scale-invariant i.e. when the distance between a source and a target point is scaled by α, the interaction between them is scaled by α^{γ}. For d-dimensional space, the interaction operators computed for the root level in the octree can be applied to interaction at level l by scaling appropriately for the change in volume (by a factor of $2^{-d\times l}$ for S2M, U and X-list interactions) and the change in distance (by a factor of $2^{-\gamma\times l}$ for L2T, U and W-list operators).

Symmetries. Most kernel functions have rotational symmetry and this allows us to group interaction directions into classes and store one interaction matrix for each interaction class. We represent interaction directions by an integer triplet (i, j, k), representing the relative coordinates of the source octant relative to the target octant. The representative class for an interaction direction is determined by taking the absolute value of each integer in the triplet and sorting them in increasing order. The change of sign of an integer represents a reflection along the corresponding coordinate and the sorting can be accomplished by a sequence of swap operations. We represent these five transformations (reflection along X, Y or Z axis and swapping {X,Y} or {X,Z} axes) by $T_1, T_2, \cdots, T_5$.

The domain and range of the precomputed translation operators is either the equivalent density data or the Chebyshev coefficient data. For equivalent density data, reflection or swapping axes results in rearrangement of the vector elements corresponding to the reordering of points on the equivalent surface. For Chebyshev coefficient data a reflection along an axis results in a change of sign of all the odd order Chebyshev coefficients and swapping axes results in reordering of the Chebyshev coefficients. For tensor kernels, these transformations can be more complex. For example, when the domain or range of a kernel function is a spatial vector field (Stokes velocity, Laplace gradient, Biot-Savart kernels), reflection along an axis will also cause the vector field component along that axis to reverse direction and swapping axes will also require a rearrangement of the components of the vector field. There may be other issues to consider, such as for Biot-Savart kernel, reflection along an axis or swapping axes leads to reversal of the field direction. Nevertheless, these five transformations ($T_1, T_2, \cdots, T_5$) can be performed through permutation and scaling operations. For equivalent density data, we have: $P_1, P_2, \cdots, P_5$ operators and for Chebyshev data, we have: $Q_1, Q_2, \cdots, Q_5$ operators.

The transformation from a direction (i, j, k) to the representative direction in its class (i_0, j_0, k_0) is given by a sequence of transformations: $T_{\alpha_1}, T_{\alpha_2}, \cdots, T_{\alpha_n}$. An interaction in direction (i, j, k) between a source vector v_s and a target vector v_t through an interaction matrix M is given by, $v_t = v_t + Mv_s$. We can now represent this interaction using the interaction matrix M_0 for the direction (i_0, j_0, k_0) by suitable transformations on the source and target vectors. For example, for W-list interactions, this would correspond to:

$$v_t = v_t + Q^T_{\alpha_1} \cdots Q^T_{\alpha_n} \times M_0 \times P_{\alpha_n} \cdots P_{\alpha_1} v_s \tag{2.11}$$

The composition of permutation and scaling operators can be precomputed.

2.4 Intra-node Parallelism

We maximize intra-node performance of our algorithm by effectively utilizing parallelism at each level of the architecture. We first discuss parallelism in the context of coprocessors, by concurrently computing near and far interactions in volume FMM on the coprocessor and the CPU respectively. Then, we summarize the most important aspects of our work, re-organizing the data structure to optimize cache performance. Furthermore, we discuss multithreading and vector intrinsics to extract maximum intra-node performance.

2.4.1 Asynchronous Execution on Coprocessor

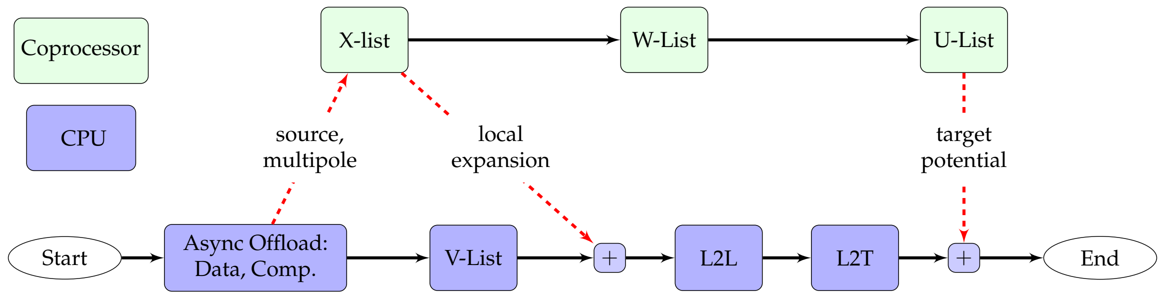


Figure 2.9 *Asynchronous computation of different interaction types on coprocessor (green) and CPU (blue), and data transfer (dashed arrows) between host and device memory in the downward-pass of FMM. The source density and multipole data is the input and output is the target potential.*

In heterogeneous architectures it is essential that we overlap computation on CPU with computation on coprocessor. In the downward pass of our volume FMM, we compute U,W and X-list interactions on coprocessor and the rest (V-list, L2L and L2T) are computed on CPU (Fig. 2.9). At the start of the downward-pass, we initiate asynchronously the following operations: transfer source Chebyshev data from host memory to coprocessor, execute X-list interactions on coprocessor, transfer downward-equivalent density (local expansion) from coprocessor to host, transfer upward-equivalent densities (multipole expansion) from host to coprocessor, execute W-list and U-list on coprocessor and transfer target potential from coprocessor to host. These operations are non-blocking, so the CPU can continue its execution; on coprocessor, each of these operations execute in sequence.

On the CPU, we compute V-list interactions. We wait for the downward-equivalent densities to finish transferring from coprocessor to host and then add the V-list contributions to it. We continue by evaluating L2L and L2T interactions on CPU and then wait for the target Chebyshev potential to complete transferring from coprocessor to CPU before adding the contributions from L2T to the target potential.

2.4.2 Near Interaction Optimizations for Volume FMM

In volume FMM, we precompute the translation operators for U,W and X-list translations as matrices. A source octant $\mathcal{B}_s$ interacts with a target octant $\mathcal{B}_t$ through an interaction matrix M_k. The contribution from v_s evaluated through M_k is added to v_t as: $v_t = v_t + M_k v_s$. Now, consider several source octants $\mathcal{B}_{s_i}$ interacting with target octants $\mathcal{B}_{t_i}$, where $(t_i, s_i) \in I_k$ and I_k is the list of index pairs for source and target octants interacting through the interaction matrix M_k. We combine these matrix-vector products into a single matrix-matrix multiplication as follows,

$$[v_{t_1}, v_{t_2}, \ldots v_{t_n}] = [v_{t_1}, v_{t_2}, \ldots v_{t_n}] + M_k\,[v_{s_1}, v_{s_2}, \cdots v_{s_n}] \tag{2.12}$$

where, $(s_i, t_i) \in I_k, \forall i = 1, .., n$. By doing so, we can now use matrix-matrix multiplication function which is a level-3 BLAS operation, instead of matrix-vector multiplication (a level-2 BLAS operation) and achieve better performance.

Many interaction matrices can be derived from another matrix through suitable permutation and scaling of its rows and columns, as discussed in Section 2.3.8. So, we can now assemble larger matrices and compute all interactions belonging to the same interaction class using a single matrix-matrix multiplication:

$$\left[w_1^1, w_2^1, \cdots w_1^2, \cdots w_n^m\right] = M_k \left[Q_1^T v_{s_1^1}, Q_1^T v_{s_2^1}, \cdots Q_2^T v_{s_1^2}, \cdots Q_m^T v_{s_n^m}\right] \tag{2.13}$$

$$\left[v_{t_1^1}, v_{t_2^1}, \cdots v_{t_1^2}, \cdots v_{t_n^m}\right] = \left[P_1 w_1^1, P_1 w_2^1, \cdots P_2 w_1^2, \cdots P_m w_n^m\right] \tag{2.14}$$

where, $(t_i^j, s_i^j) \in I_{k_j}$ and $M_{k_j} = P_j M_k Q_j^T \quad \forall j = 1, \cdots, m$. A similar technique is used [81] to optimize M2L interactions in black-box FMM. We have implemented highly optimized kernels for these permutation operations on Phi and GPUs. In addition, when assembling matrices, we try to minimize memory reads and writes by computing all the required permutations together for each vector loaded in cache. Using symmetries significantly improves performance for U,W and X-list interactions on both the CPU and on coprocessor, particularly for small problems.

2.4.3 Near Interaction Optimizations for Particle FMM

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 and then perform Newton iteration for higher accuracy. 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.

2.4.4 V-List Optimizations

For a source octant $\mathcal{B}_s$ interacting with a target octant $\mathcal{B}_t$ through multipole-to-local (or V-list) translation operator, the Hadamard product for the interaction is represented as: $v_t = v_t + M_k \circ v_s$. Since Hadamard product has $\mathcal{O}(n)$ floating-point operations, and $\mathcal{O}(n)$ memory accesses for vectors of length n, a naive scheme will be bound by the memory bandwidth. However, we note that V-list interactions have spatial locality, i.e. the same set of source and target vectors are used when evaluating interactions for a compact region in space. Therefore, if we can keep data in cache then we can significantly improve performance.

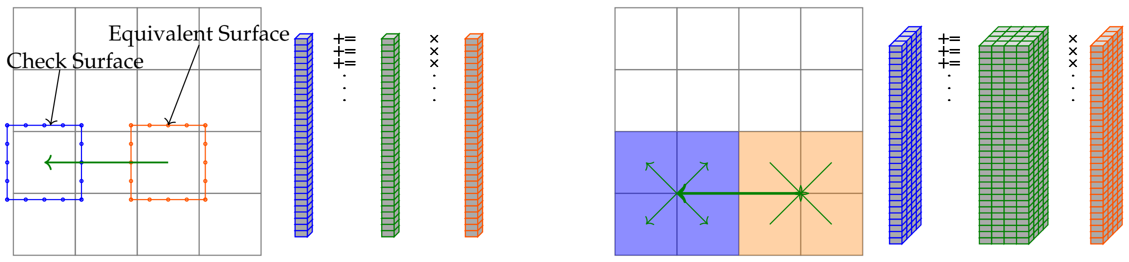


Figure 2.10 *Left: Interaction list for an octant using the conventional organization of the V-list interaction. Right: Interaction between two sibling groups.*

The first optimization that we make is to interleave the source and target vectors for sibling octants. We compute interactions between adjacent sibling groups by loading the first eight elements (one from each sibling) in the interleaved source and target vectors and computing all interactions between these (Fig. 2.10), represented as multiplication with an 8×8 matrix, then load the next eight elements from the vectors and so on for the length of the vectors. By doing so, we also compute interactions between adjacent octants which do not actually appear in V-list interactions, so the corresponding entry in the 8×8 matrix is zero. As a result, we perform some extra computation (about 10%), however the increased efficiency justifies the additional computational cost.

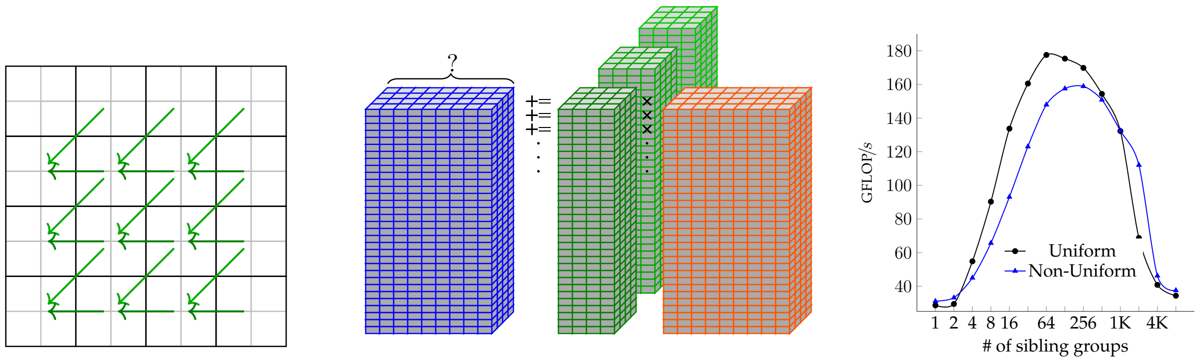


Figure 2.11 *Combining multiple sibling group interactions and determining the optimal block size for interactions. For the optimal block size, we achieve over* 50% *of peak performance even for highly adaptive octrees.*

Next, we note that as in the case of U,W and X-lists, we can combine several interactions in the same direction (Fig. 2.11) and replace matrix-vector multiplications by a single matrix-matrix multiplication. Since, these matrices are small it is not efficient to use BLAS and therefore, we implement our own matrix-matrix multiplication routine for 8×8 matrices optimized for the Sandy Bridge architecture by using AVX vector intrinsics. We can also look at this computation as a stack of matrix-matrix products. Each layer in the stack can be computed independently and we use OpenMP parallelism to distribute work across cores.

We further optimize cache usage by taking a Morton-sorted list of target octants (at the same level) and splitting into blocks which have spatial locality. By doing so, we ensure that we can keep the first eight elements from the source and target vectors of each sibling group (one layer of the stack) in cache when we loop over different interaction directions. In Fig. 2.11, the plot shows the performance in GFLOP/*s* on 16 CPU-cores (2×Xeon E5-2680) for different block sizes. A block size of about 128 sibling groups worked best in our experiments, achieving nearly 180GFLOP/*s* or about 50% of the theoretical peak. In Table 2.3, we give the arithmetic intensity (defined as the number of floating-point operations per word (8-bytes) of memory transfer) using our scheme on a uniform octree assuming that the block of data fits in the cache. As we increase the block size, the arithmetic intensity increases, however, the required cache also increases. For a block size of 128, we already require more memory than what is available in L1 cache and this prevents us from achieving higher performance.

BlockSize	CacheSize (kB)	FLOP	Mem.Transfer	Arith.Intensity
32	22	4.3E+5	6.6E+3	65
64	36	8.5E+5	8.8E+3	97
128	61	1.7E+6	1.3E+4	131
256	106	3.4E+6	2.1E+4	162
512	190	6.8E+6	3.6E+4	189
$\to\infty$	$\to\infty$	$\to\infty$	$\to\infty$	277

Table 2.3 *Arithmetic intensity (defined as* FLOP */word) and the required cache size for different block sizes in V-list computation. Memory transfers are the number of words (8-bytes) transferred.*

The overall algorithm (for one block of data) to compute Hadamard products is described in Algorithm 2. In lines 1-3, we interleave source data for siblings. Next, the outermost loop (line 4) is over the height of the stack and this is parallelized using OpenMP. Then, we loop over each of the 26 interaction directions. In the innermost loop, we compute the matrix-vector products for each source-target interaction pair. We then deinterleave target data (lines 11-13) for siblings to get the downward-check potential in Fourier space for each octant.

ALGORITHM 2: VLISTHADAMARD

```
Input: $v_s$ source vectors in Fourier space of length $(2m)^3$;  $M_k^i$ translation operators for $k = 1, .., 26$
       (each sibling group interaction direction) and $i = 1, \cdots, (2m)^3$
Output: $v_t$ target vectors in Fourier space of length $(2m)^3$

foreach $(s_1, \cdots, s_8) \in$ source sibling octants do                          // interleave data
    for $i \leftarrow 0$ to $(2m)^3 - 1$ do  $w_s(8i, \cdots, 8i+7) \leftarrow [\, v_{s_1}(i), \cdots, v_{s_8}(i) \,]$ ;
end
for $i \leftarrow 0$ to $(2m)^3 - 1$ do in parallel                                   // vector length
    for $k \leftarrow 1$ to $26$ do                                                    // directions
        foreach $(s, t)$ pair in direction $k$ do
            $w_t(8i, \cdots, 8i+7) \leftarrow M_k^i \times w_s(8i, \cdots, 8i+7)$
        end
    end
end
foreach $(t_1, \cdots, t_8) \in$ target sibling octants do                           // deinterleave data
    for $i \leftarrow 0$ to $(2m)^3 - 1$ do  $[\, v_{t_1}(i), \cdots, v_{t_8}(i) \,] \leftarrow w_t(8i, \cdots, 8i+7)$ ;
end
return $v_t$;
```

2.5 Distributed-Memory Parallelism

We first discuss the distributed-memory tree construction and explain the partitioning of the domain across processes. We also discuss the parallel 2:1 balance algorithm on this distributed octree. Finally, we explain the communication steps in the parallel FMM algorithm. To analyze the communication cost, we assume an uncongested network and therefore assume that the cost of point-to-point communication between any two compute nodes is given by the sum of the latency t_s and the message transfer time $t_w N_m$ where, t_w is the per-word transfer time and N_m is the message size [48].

2.5.1 Tree Construction

On a distributed-memory system we use Morton IDs for tree construction and load balancing [107]. We sort the initial seed points by their Morton ID using a distributed sort and partition the points equally between processes. Each process constructs a linear octree (a linear array of leaf octants sorted by their Morton ID) using its local point set. We then collect the Morton IDs of the first octant of each local octree and build the array $M_0, \cdots, M_{p-1}$. The domain belonging to a process with process ID p_r is given by the region between M_{p_r} and M_{p_r+1} on the Morton curve. We then proceed with the adaptive refinement. After each level of refinement, we load balance by redistributing the leaf octants equally across processes. The exchange of octant data requires only point-to-point communication with at most eight other processes (although determining which processes must exchange data requires collective communication). For an octree with n_d levels, N_{oct} local octants, the total communication cost is $T(n) = \mathcal{O}\left(t_s n_d \log p + t_w n_d (p \log p + N_{\text{oct}} q^3/6)\right)$.

ALGORITHM 3: PARALLELBALANCE

Input: $\mathcal{T}_{p_r}$ unbalanced local octree, L_{max} maximum tree depth
Output: $\widehat{\mathcal{T}}_{p_r}$ globally balanced octree

```
for $i \leftarrow L_{max}$ to $0$ do
    $\mathrm{N}_i \leftarrow \{\mathcal{B} :\ \mathcal{B} \in \mathcal{T}_{p_r} \setminus \mathcal{L}(\mathcal{T}_{p_r}),\ \ \mathrm{Level}(\mathcal{B}) = i\}$;
    $\mathrm{N}_i \leftarrow \mathrm{N}_i \cup \mathrm{Parent}(\mathrm{Colleagues}(\mathrm{N}_{i+1}))$;
end
$\mathrm{N} \leftarrow \mathrm{N}_0 \cup \cdots \cup \mathrm{N}_{L_{max}}$;
$\mathrm{N} \leftarrow \mathrm{ParallelSort}(\ \mathrm{N} \setminus \mathrm{Ancestors}(\mathrm{N})\ )$ ;        // HykSort [96]
$\mathrm{N} \leftarrow \mathrm{RemoveDuplicates}(\mathrm{N})$;
$\widehat{\mathcal{T}}_{p_r} \leftarrow \mathrm{Children}(\mathrm{N})$;
$\widehat{\mathcal{T}}_{p_r} \leftarrow \mathrm{CompleteOctree}(\widehat{\mathcal{T}}_{p_r})$ ;        // add missing octants
return $\widehat{\mathcal{T}}$;
```

2.5.2 2:1 Balance Refinement

For the parallel 2:1 balance algorithm, we start with the distributed linear octree, i.e. a distributed linear array of tree nodes sorted by their Morton ID. The first few lines of Algorithm 3 are similar to the sequential version (Algorithm 1). We generate the set of non-leaf octants N in the local balanced octree, however do not add the leaf octants at this point. The leaf octants are added later and this reduces the communication cost by about $8\times$. Next, we globally sort the set of non-leaf octants N using a variation of the hyperquick sort algorithm that we have developed called HykSort [96]. This is followed by removing duplicate octants, which is trivial given a sorted set of octants. Finally, the leaf octants ($\mathrm{Children}(\mathrm{N})$) are added and the tree is completed by adding missing octants in the Morton ID sequence. The advantage of this 2:1 balance algorithm is that it does not suffer from large load imbalance even for highly adaptive trees. Most other algorithms repartition octants as a post processing step and therefore, during the local refinement process, the load imbalance can potentially be unbounded.

2.5.3 Distributed-Memory FMM

In the upward-pass of FMM, we compute the multipole expansions for each octant. For non-leaf octants that are shared between processes, we need to perform a reduction to sum the contributions from regions owned by different processes.

In the downward-pass of the FMM, we compute interactions between octants. For a distributed octree, the interacting source octants may belong to different processes. Therefore, we build a local essential tree by communicating the ghost octants needed by a process for the downward-pass. Once we have constructed the local essential tree, the downward-pass of the FMM can proceed independently of all other processes.

Compared to [67], we have decoupled to reduction and broadcast operations. Although this does not change the overall complexity, the resulting algorithms are simpler. This also makes future optimization of the broadcast operation possible, by using point-to-point communication to exchange octants at finer levels in the octree and using the hypercube all-to-all scheme only for coarser octants.

ALGORITHM 4: MultipoleReduce

Input: p_r process rank, p process count, $\mathcal{T}_{p_r}$ local tree.
Output: $\mathcal{T}_{p_r}$ with correct multipole expansions.

```
$S_1 \leftarrow \{\mathcal{B} : \mathcal{B} \in \text{Ancestors}(\min \mathcal{L}(\mathcal{T}_{p_r}))\}$;
$S_2 \leftarrow \{\mathcal{B} : \mathcal{B} \in \text{Ancestors}(\max \mathcal{L}(\mathcal{T}_{p_r}))\}$;
for $i \leftarrow 0$ to $\log p$ do
    $p_0 \leftarrow p_r$ XOR $2^i$;
    $\text{Send}([S_1, S_2], p_0)$;
    $\text{Recv}([R_1, R_2], p_0)$;
    if $p_r \leq p_0$ then
        Reduce($S_2$, $R_1$);
        $S_2 \leftarrow R_2$;
    else
        Reduce($S_1$, $R_2$);
        $S_1 \leftarrow R_1$;
    end
end
```

Multipole Reduce. We give the pseudocode for reduction in Algorithm 4. For simplicity, we assume that the processor count p is of the form 2^k. The communication between processes required for the multipole reduction is mapped to a hypercube network topology. Each process identifies the list of octants it shares with other processes. These are the ancestors of either the first or the last leaf octant in the local octree. In each step, a process exchanges shared octants with another process that it is directly connected to in the hypercube topology, moving in the order of least significant dimension to the most significant dimension of the hypercube. The shared octants among pairs of adjacent regions are merged forming a bigger region such that the only octants which still need to be updated are the octants shared across the new bigger regions. At any stage in this process, each process maintains a list of octants which it's region shares with adjacent regions, i.e. a maximum of $2L_{max}$ octants, where L_{max} is the maximum depth of the tree. In each communication step, the shared octants are exchanged and then each process independently sums the multipole expansions of the octants shared between the two regions and builds the list of octants shared by the new region with adjacent regions. The time complexity for this algorithm is given by:

$$T(n) = \mathcal{O}\left(t_s \log p + t_w m^2 \log p L_{max} + m^2 \log p L_{max}\right) \tag{2.15}$$

Here, t_s, t_w are the communication latency and the per-word transfer time respectively, p is the number of processes, L_{max} is the maximum depth of the octree.

Multipole Broadcast. We build the local essential tree by sending ghost octants from its owner process to each of its user processes. The pseudocode for the hypercube broadcast is given in Algorithm 5. For simplicity, we have assumed that p is of the form 2^k. For each process p_r, we identify shared local octants Q. We split the processes into two groups ($\{p_1, \cdots, p_1 + 2^i - 1\}$ and $\{p_2, \cdots, p_2 + 2^i - 1\} \ni p_r$) each with 2^i processes. Each process communicates with a process p_0 in the other group $\{p_1, \cdots, p_1 + 2^i - 1\}$ and sends those octants from its shared set Q, which have user processes in $\{p_1, \cdots, p_1 + 2^i - 1\}$. Next, we retain the new received octants and only those shared octants which will be used in subsequent communication steps. We stop when the process set contains only p_r. Then, Q contains all the ghost octants which together with the local octants in $\mathcal{T}_{p_r}$ make up the local essential tree.

The communication cost for the hypercube communication scheme is discussed in detail in [67]. For an uncongested network, that work provides a worst case complexity which scales as $\mathcal{O}\left(t_s \log p + t_w N_s (q^3 + m^2)\sqrt{p}\right)$, where N_s is the maximum number of shared octants owned by any process. However, assuming that the messages are evenly distributed across processes in every stage of the hypercube communication, we get a cost of $\mathcal{O}\left(t_s \log p + t_w N_s (q^3 + m^2) \log p\right)$. For our experiments with uniform octrees, the observed complexity appears to agree with this estimate.

ALGORITHM 5: CONSTRUCTLET

Input: p_r process rank, p process count, $\mathcal{P}_u(\mathcal{B})$ user processes of octant $\mathcal{B}$, $\mathcal{T}_{p_r}$ local tree.
Output: $\widehat{\mathcal{T}}_{p_r}$ local tree with ghost octants added.

```
$Q \leftarrow \{\mathcal{B} : \mathcal{B} \in \mathcal{T}_{p_r}, |\mathcal{P}_u(\mathcal{B})| > 1\}$ ;    // all shared octants
for $i \leftarrow (\log p - 1)$ to 0 do
    $p_0 \leftarrow p_r$ XOR $2^i$;
    $p_1 \leftarrow p_0$ AND $(p - 2^i)$;
    $p_2 \leftarrow p_r$ AND $(p - 2^i)$;
    $S \leftarrow \{\mathcal{B} : \mathcal{B} \in Q, \mathcal{P}_u(\mathcal{B}) \cap \{p_1, \cdots, p_1 + 2^i - 1\} \neq \emptyset\}$;
    $\mathrm{Send}(S, p_0)$;
    $\mathrm{Recv}(R, p_0)$;
    $Q \leftarrow \{\mathcal{B} : \mathcal{B} \in Q, \mathcal{P}_u(\mathcal{B}) \cap \{p_2, \cdots, p_2 + 2^i - 1\} \neq \emptyset\}$;
    $Q \leftarrow Q \cup R$;
end
return $\widehat{\mathcal{T}}_{p_r} \leftarrow \mathcal{T}_{p_r} \cup Q$
```

2.6 Convergence Analysis

We conduct experiments to measure errors as a function of various parameters and show that they converge as predicted by the theory. All errors reported in this chapter are relative errors. We also report time to solution, CPU cycles per unknown and FLOP -rates for the evaluation phase. All results in this section were obtained on a single node of the Stampede platform at Texas Advanced Computing Center (TACC), 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. In the results, we indicate wherever we have used a large memory node. The peak theoretical double-precision performance using 16 CPU cores is 345.5GFLOP/s .

2.6.1 Convergence Results for Particle FMM

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.

N_{pt}	N_{leaf}	m	Error	T_{All}	$cycles/N_{pt}$	T_{Tree}	T_{Setup}	T_{FMM}(GFLOP/s)
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 2.4 *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.*

N_{pt}	N_{leaf}	m	Error	T_{All}	$cycles/N_{pt}$	T_{Tree}	T_{Setup}	T_{FMM}(GFLOP/s)
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 2.5 *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.*

In Table 2.4 and Table 2.5, we report results for Laplace kernel with uniform and non-uniform particle distributions respectively. In each table we report two sets of results, first 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 2.6 and Table 2.7 respectively. The Helmholtz problem corresponds to a wavenumber of ten in the length of the computational domain. 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.

N_{pt}	N_{leaf}	m	Error	T_{All}	$cycles/N_{pt}$	T_{Tree}	T_{Setup}	T_{FMM}(GFLOP/s)
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 2.6 *Convergence results for particle FMM with Stokes kernel.*

N_{pt}	N_{leaf}	m	Error	T_{All}	$cycles/N_{pt}$	T_{Tree}	T_{Setup}	T_{FMM}(GFLOP/s)
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 2.7 *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.*

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 $T_{\text{All}} \times 2.7\text{GHz} \times 16cores/N_{pt}$ 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 2.4 and Table 2.5, we observe that the cycles per particle is also 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 16FLOPs in single precision and 20FLOPs 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 2.7), we needed more than 32GB of memory and we had to use the large memory node on Stampede.

2.6.2 Convergence Results for Volume FMM

$\|\|e_f\|\|_\infty$	m	q	N_{leaf}	$\|\|e_u\|\|_\infty$	T_{All}	$cycles/N$	T_{Tree}	T_{Setup}	T_{FMM}(GFLOP/s)
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 2.8 *Convergence with multipole order m for a Poisson problem.*

Laplace Kernel. In Table 2.8, we solve the Poisson problem with free-space boundary conditions 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 exact solution is given by $u(x) = e^{-\alpha|x|^2}$. 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 with the matrix-vector product implemented using FMM.

$\|\|e_f\|\|_\infty$	m	q	N_{leaf}	$\|\|e_u\|\|_\infty$	T_{All}	*cycles/N*	T_{Tree}	T_{Setup}	T_{FMM}(GFLOP/s)
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 2.9 *Convergence with multipole order m for a Stokes problem.*

Stokes Kernel. In Table 2.9 we show convergence for the following Stokes problem, where we solve for the velocity field $u(x)$ with free space boundary conditions in the domain $(-0.5, 0.5)^3$,

$$-\mu\Delta u + \nabla p = 4L^2(5 - 2L\,|x|^2)e^{-L|x|^2}(x_3\mathbf{e_2} - x_2\mathbf{e_3}), \operatorname{div} u = 0, \tag{2.16}$$

$$u(x) = \frac{2L}{\mu}e^{-L|x|^2}(x_3\mathbf{e_2} - x_2\mathbf{e_3}) \tag{2.17}$$

where, $\mathbf{e_2}$, $\mathbf{e_3}$ 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.

Helmholtz Kernel. In Table 2.10, we demonstrate the case of an oscillatory kernel by solving the Helmholtz equation with wavenumber 10 and free space boundary conditions in the domain $(-0.5, 0.5)^3$,

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

$\|e_f\|_\infty$	m	q	N_{leaf}	$\|e_u\|_\infty$	T_{All}	$cycles/N$	T_{Tree}	T_{Setup}	T_{FMM}(GFLOP/s)
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 2.10 *Convergence with multipole order m for a Helmholtz problem with wavenumber 10. The results for the shaded row are computed on the large memory node of Stampede using 16 processor cores.*

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.

2.7 Single-Node Performance Results

We now present detailed double precision performance results on a single node of Stampede (TACC), which has 16 CPU-cores and an Intel Xeon Phi SE10P coprocessor. Each CPU core has a peak double precision performance of 21.6GFLOP/s and the Xeon Phi coprocessor has a peak double precision performance of 1.1TFLOP/s. Each compute node has a total peak performance of 1.4TFLOP/s.

2.7.1 Performance of M2L Translation

cores	N_{leaf}	HADAMARD	FFT+IFFT	ALL
1	512	0.308 (13.9)	0.127 (3.6)	0.434 (10.9)
4	512	0.081 (13.3)	0.033 (3.5)	0.114 (10.4)
16	512	0.021 (12.7)	0.010 (2.8)	0.035 (8.5)
1	4096	2.761 (12.4)	1.013 (3.6)	3.774 (10.1)
4	4096	0.720 (11.9)	0.258 (3.6)	0.977 (9.7)
16	4096	0.190 (11.1)	0.071 (3.2)	0.265 (8.9)

Table 2.11 *Results with timing and performance in GFLOP/s per core (in parenthesis) for shared-memory strong scaling of multipole-to-local translation for uniform octree with Laplace kernel and $m = 10$. We show the time spent in the Hadamard product stage which we have optimized and discussed in Section 2.4.4 and the FFT and IFFT computation stage through FFTW library.*

In Tables 2.11 and 2.12 , we demonstrate intra-node strong scalability for **our new multipole-to-local translation algorithm** for uniform and non-uniform octrees respectively. We only report OpenMP results since we get best performance with 16 OpenMP

cores	N_{leaf}	HADAMARD	FFT+IFFT	ALL
1	904	0.265 (11.1)	0.224 (3.6)	0.488 (7.7)
4	904	0.074 (9.9)	0.057 (3.5)	0.132 (7.1)
16	904	0.021 (8.6)	0.017 (3.1)	0.044 (5.3)
1	62483	24.989 (11.4)	15.439 (3.6)	40.428 (8.4)
4	62483	6.836 (10.4)	3.940 (3.6)	10.776 (7.9)
16	62483	1.883 (9.5)	1.040 (3.4)	2.925 (7.3)

Table 2.12 *Results with timing and performance in* GFLOP/s *per core (in parenthesis) for shared-memory strong scaling of V-List for non-uniform octree with Laplace kernel and $m = 10$. Even for highly non-uniform octrees and small problems sizes, we achieve very high* FLOP *-rates. This demonstrates the robustness of our scheme.*

threads and one MPI process per compute node and this is also the mode of operation in all runs with more than one compute node. As a result of our new optimized algorithm for Hadamard product, we achieve 203GFLOP/s per compute node or 60% of theoretical peak on one node and achieve 90% efficiency for intra-node strong scaling for the uniform case. This is a significant improvement over a naive Hadamard product, which was limited by the main-memory bandwidth and attained roughly 16GFLOP/s on one compute node.

2.7.2 Performance of Volume FMM

In Table 2.13 we show performance for various stages in the **downward-pass** for the Stokes kernel with a non-uniform octree. We show results for different values of parameters m (order of multipole expansion), q (degree of polynomial approximation) and for different problem sizes (number of leaf octants, N_{leaf}). In each case, we show the performance for three configurations: 1) CPU only configuration, 2) CPU+XEON PHI: with U,W and X-lists executing on Xeon Phi and everything else on CPU, 3) ASYNC: with Xeon Phi executing asynchronously and overlapped with CPU execution.

In Table 2.13 we first show performance for a low-order case with $m = 2$ and $q = 4$. For smaller problem sizes, we observe longer solve time when we use the Phi coprocessor. This is because there is insufficient parallelism to effectively utilize the coprocessor. As we increase the problem size, we observe about 20-40% speedup for the ASYNC case compared to the CPU only case.

For higher accuracy with $m = 8$ and $q = 13$, U,W and X-list evaluation dominates the execution time for the CPU only case and for CPU+XEON PHI configuration, it is comparable to V-list execution time. In the ASYNC mode, the CPU is idle for some time as

	N_{leaf}	U,W,X-List	V-List	L2L+L2T	Wait	All
Low-order ($m = 2,\ \ q = 4$)						
CPU	904	0.005 (100.2)	0.002 (121.4)	0.000 (19.3)	0.000	0.010 (90.7)
CPU+Phi	904	0.046 (12.8)	0.002 (132.5)	0.000 (19.3)	0.000	0.084 (10.4)
Async	904	0.000 (–NA-)	0.003 (107.9)	0.000 (19.3)	0.031	0.044 (19.8)
CPU	9654	0.059 (101.8)	0.027 (159.0)	0.002 (43.9)	0.000	0.090 (116.0)
CPU+Phi	9654	0.069 (87.0)	0.027 (160.2)	0.002 (43.9)	0.000	0.104 (100.3)
Async	9654	0.000 (–NA-)	0.029 (151.9)	0.002 (43.9)	0.043	0.076 (137.6)
High-order ($m = 8,\ \ q = 13$)						
CPU	904	0.700 (274.2)	0.172 (92.2)	0.045 (94.9)	0.000	0.921 (230.0)
CPU+Phi	904	0.327 (586.4)	0.126 (125.2)	0.031 (137.2)	0.000	0.494 (428.4)
Async	904	0.003 (–NA-)	0.124 (127.2)	0.031 (137.2)	0.173	0.333 (636.5)
CPU	9654	6.144 (305.4)	1.627 (141.7)	0.184 (250.6)	0.000	7.958 (270.6)
CPU+Phi	9654	2.773 (676.7)	1.632 (141.4)	0.189 (243.9)	0.000	4.674 (460.8)
Async	9654	0.003 (–NA-)	1.642 (140.4)	0.190 (242.6)	0.987	2.832 (760.3)

Table 2.13 *Results for timing and performance in* GFLOP/s *(in parenthesis) for downward-pass for Stokes kernel. We report results for low-accuracy case with $m = 2$ and $q = 4$ and high-accuracy case with $m = 8$ and $q = 13$ for different problem sizes on non-uniform octrees. We compare results for different stages of the downward-pass, for CPU only, CPU+Phi and asynchronous runs.*

it waits for computation on Xeon Phi to complete. Here we observe a significant speedup ($2.8\times$ for large problems) because we are able to keep the Xeon Phi busy.

In Table 2.14, we provide similar results for Laplace kernel. For the high-order case, we also show the performance for the Original code i.e. without the optimizations discussed in Section 2.4. We see speedup of about $3\times$ for the CPU version and $7-7.6\times$ for the Async case.

2.8 Distributed Memory Performance Results

For the majority of our experiments in this section we have used **TACC's Stampede** system in both strong and weak scaling regimes. Stampede is a high-performance Linux cluster consisting of 6400 compute nodes, each with 16 CPU-cores (2×Xeon E5-2680) and an Intel Xeon Phi SE10P coprocessor. Stampede has a 56Gb/s FDR Mellanox InfiniBand network connected in a fat tree configuration that carries all high-speed traffic (including both MPI and parallel file-system data).

We also ran experiments on **ORNL's Titan**, a Cray XK7 with a total of 18,688 nodes, each consisting of a single 16-core AMD Opteron 6274 (Interlagos) processor, for a total of

	N_{leaf}	U,W,X-List	V-List	L2L+L2T	Wait	All
			Low-order ($m = 4,\ \ q = 6$)			
CPU	904	0.005 (97.9)	0.002 (105.3)	0.000 (22.8)	0.000	0.009 (85.1)
CPU+Phi	904	0.040 (12.9)	0.002 (110.3)	0.000 (22.8)	0.000	0.080 (9.6)
Async	904	0.000 (–NA-)	0.003 (72.4)	0.000 (22.8)	0.035	0.060 (12.8)
CPU	9654	0.050 (99.9)	0.026 (133.0)	0.002 (72.6)	0.000	0.080 (108.4)
CPU+Phi	9654	0.064 (78.3)	0.025 (135.7)	0.003 (57.0)	0.000	0.099 (87.5)
Async	9654	0.000 (–NA-)	0.027 (125.7)	0.002 (66.7)	0.035	0.068 (127.1)
			High-order ($m = 10,\ \ q = 13$)			
Original	904	0.315 (85.0)	0.214 (14.3)	0.009 (106.0)	0.000	0.570 (55.9)
CPU	904	0.124 (215.0)	0.032 (116.2)	0.006 (153.7)	0.000	0.164 (191.6)
CPU+Phi	904	0.080 (332.1)	0.032 (115.1)	0.007 (146.9)	0.000	0.125 (250.3)
Async	904	0.000 (–NA-)	0.033 (112.0)	0.006 (156.2)	0.040	0.082 (380.5)
Original	9654	1.294 (194.1)	2.724 (16.5)	0.069 (153.8)	0.000	4.258 (74.5)
CPU	9654	0.973 (257.5)	0.379 (140.1)	0.046 (228.9)	0.000	1.403 (224.0)
CPU+Phi	9654	0.536 (467.3)	0.380 (139.7)	0.048 (220.8)	0.004	0.999 (314.6)
Async	9654	0.000 (–NA-)	0.387 (137.0)	0.047 (223.1)	0.115	0.558 (563.0)

Table 2.14 *Results for timing and performance in* GFLOP/s *(in parenthesis) for downward-pass for Laplace kernel. We compare results for different stages of the downward-pass, for CPU only, CPU+Phi and asynchronous runs. For the high accuracy test case, we also compare with an original version of the code without the V-list optimizations and without symmetry optimizations for U,W and X-lists.*

299,008 cores. Each node has 32GB of memory. It is equipped with a Gemini interconnect. Most nodes are also equipped with NVIDIA GPUs but we have not used them in the experiments we report here.

For our GPU results in Section 2.8.4, we have used **TACC's Maverick** system. It 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 and an NVIDIA Tesla K40 GPU.

2.8.1 Strong Scalability of Particle FMM

In Fig. 2.12 and Fig. 2.13 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 uni-

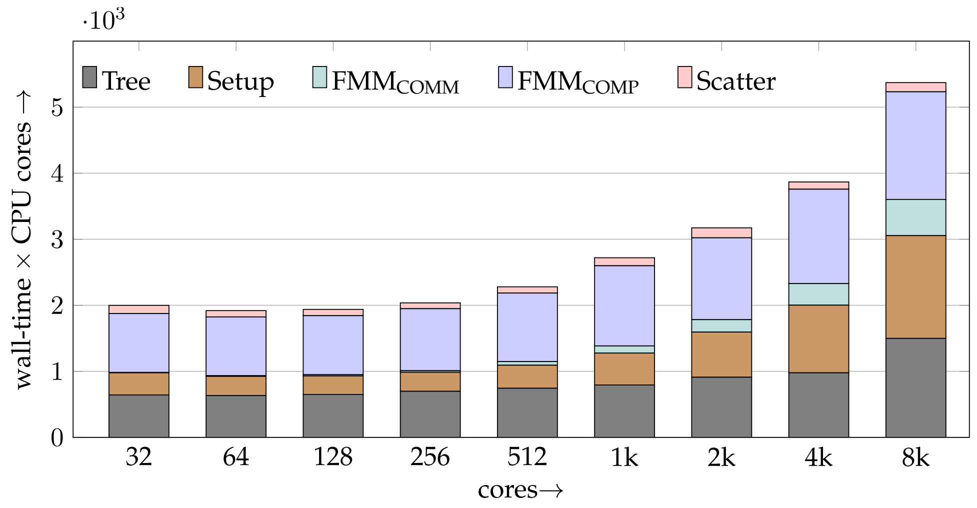


Figure 2.12 *Strong scalability results for Laplace kernel with multipole order $m = 6$ and a highly non-uniform distribution of 1E+8 particles with 18 levels of octree refinement. We increase the number of processor cores from 32 to 8k and observe a 95× speedup and 37% parallel efficiency. We achieve 14TFLOP/s of double precision performance on 8k CPU cores.*

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

2.8.2 Strong Scalability of Volume FMM

We present strong scaling results for the volume FMM in Fig. 2.14 and Fig. 2.15. We report the total CPU time, computed as wall-time × CPU cores, for the FMM evaluation

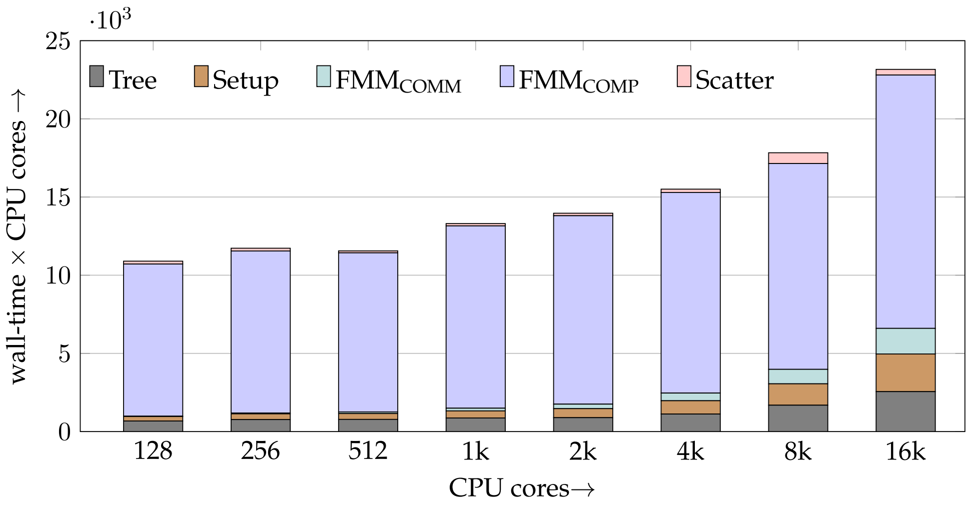


Figure 2.13 *Strong scalability results for Helmholtz kernel (wavenumber=10) with multipole order* $m = 10$ *and a non-uniform distribution of* 1E+8 *particles with 9 levels of octree refinement. We increase the number of processor cores from* 128 *to* 16k *and observe a* $60\times$ *speedup and* 47% *parallel efficiency. We achieve* 37TFLOP/s *of performance on 16k CPU cores.*

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 (Fig. 2.14), we achieve 439GFLOP/s on a single compute node. 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 $43\times$ speedup and 19TFLOP/s 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 Fig. 2.15, we present strong scaling results for the Helmholtz kernel as we scale from 64 to $4k$ CPU cores. We achieve $20\times$ speedup with 31% parallel efficiency and 35TFLOP/s of performance. Although we observe good performance on a small number of cores (1.7TFLOP/s 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 each level in the octree.

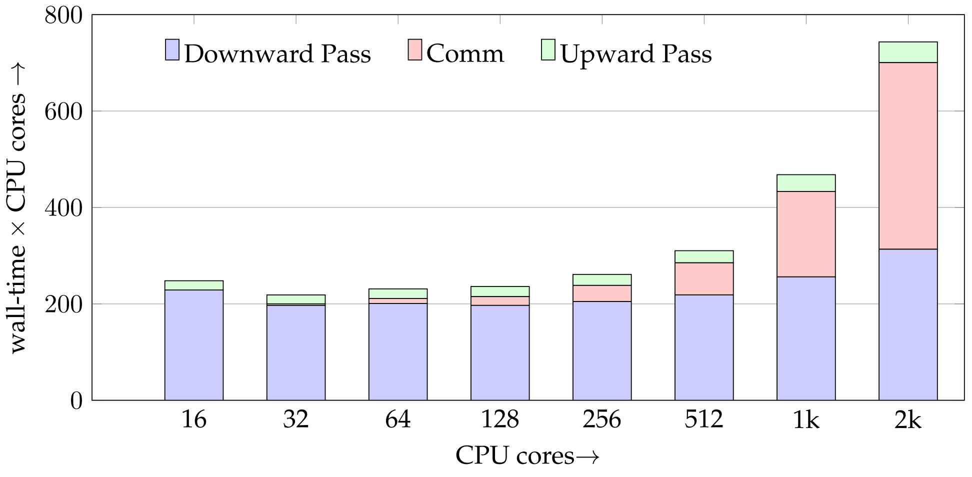


Figure 2.14 *Strong scaling on Stampede while using Phi coprocessor for Laplace kernel with* $m = 10$, $q = 13$, 2E+5 *leaf octants and* 16 *levels of refinement for* 1.13E+8 *unknowns. We observe* $43\times$ *speedup as we scale up to* $2k$ *CPU cores with* 33% *parallel efficiency. We achieve* 19TFLOP/s *of performance with a wall-time of 0.36s on* $2k$ *cores.*

2.8.3 Weak Scalability of Volume FMM

In Fig. 2.16 (left) we demonstrate weak scalability on Stampede for the Laplace kernel. Here, the octree is highly non-uniform with a maximum tree depth of 23 levels. However, we still achieve about 93% efficiency for the upward and downward passes combined. For the overall FMM evaluation, we achieve about 78.4% efficiency with about 457TFLOP/s or 33% of peak theoretical performance. In Fig. 2.16 (right), we present weak scalability results on ORNL's Titan, for the same problem as above, going up to 25 levels and 73.6 billion unknowns, while scaling from 1 MPI process to $16K$ MPI processes on $16K$ nodes of Titan and achieve 567TFLOP/s.

In Fig. 2.17 we show weak scalability with uniform octrees for Laplace kernel (left) and low-frequency Helmholtz kernel (right) for a grain size of 16K octants per compute node. In both cases, the communication cost appears to scale logarithmically as predicted for a network without congestion. The remaining stages, the upward-pass and the downward-pass, also scale well with about $85 - 90\%$ efficiency. Overall, for 1024 compute nodes, we have 74% efficiency and 359TFLOP/s for Laplace kernel and 80% efficiency and 351TFLOP/s for Helmholtz kernel. We have better scalability for Helmholtz, since it is a 2×2 tensor kernel, therefore it has $4\times$ the computation but only $2\times$ the communication compared to the Laplace problem for the same number of octants and therefore communication overhead for Helmholtz is lower. This can be clearly seen from the bar graph.

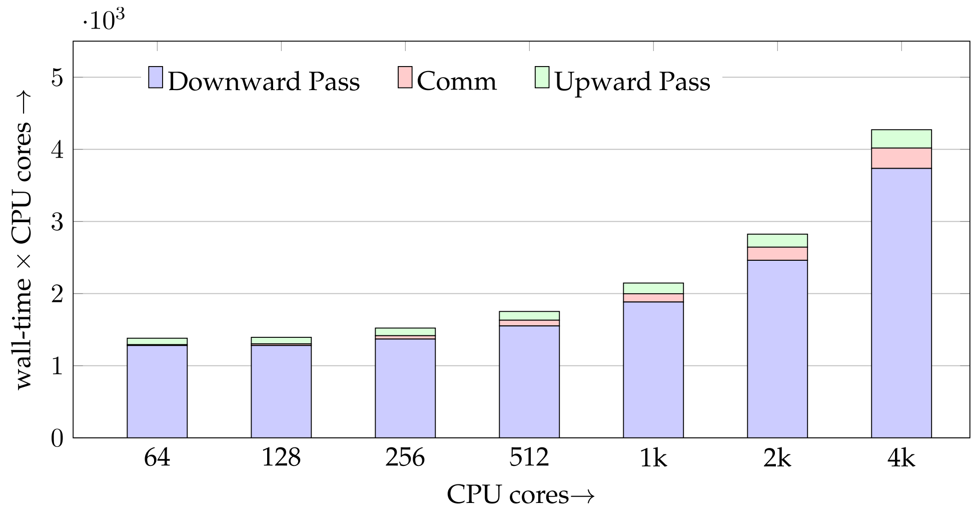


Figure 2.15 *Strong scaling on Stampede for Helmholtz kernel with wavenumber=10,* $m = 10$*,* $q = 14$*,* 2.6E+5 *leaf octants and* 6 *levels of refinement for* 3.6E+8 *unknowns. We observe* 20× *speedup as we scale up to* 4*k CPU cores with* 31% *parallel efficiency. We achieve* 35TFLOP/*s of performance with a wall-time of 1.04s on* 4*k cores.*

2.8.4 Performance with GPU Accelerators

In Table 2.15, we report results on Maverick using 32 compute nodes. We observe high FLOP rates (∼ 800GFLOP/*s* per compute node) due to the use of GPUs for computing near (U,W and X-list) interactions.

kernel	N	L_{max}	T_{Tree}	T_{FMM}	TFLOP/*s*
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 2.15 *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.*

2.8.5 Scalability of 2:1 Balance Refinement

Although 2:1 balance refinement is not part of the evaluation phase, it is an important component of the setup phase for our solver. In the past we have used an algorithm described

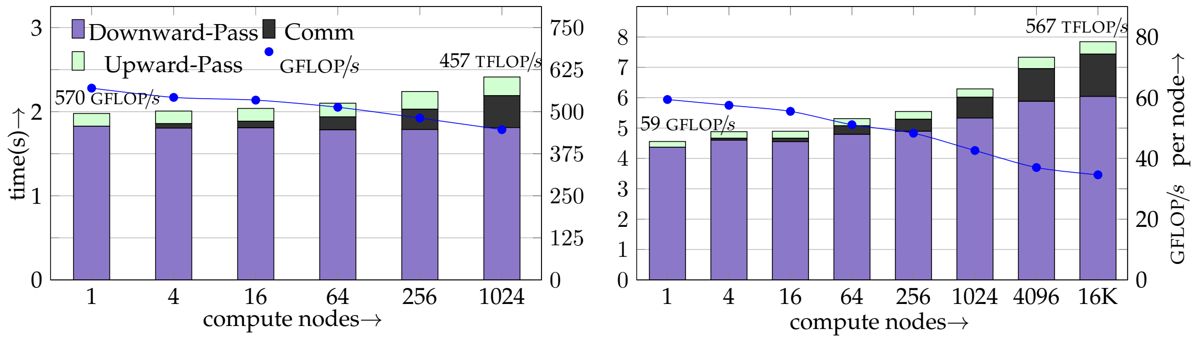


Figure 2.16 *Weak scaling for highly non-uniform octrees with* $m = 10$, $q = 13$. *Left: Stampede (asynchronous execution on Phi) with 32K octants per compute node and 23 levels of tree refinement. Right: Titan (without using GPUs) with 8K octants per compute node and 25 levels of refinement.*

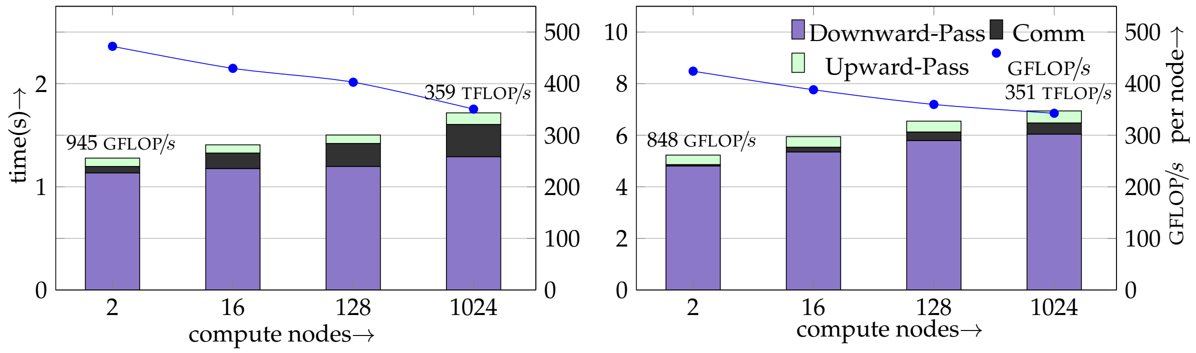


Figure 2.17 *Weak scaling on Stampede (asynchronous execution on Phi) for Laplace (left) and low-frequency Helmholtz (right) kernels with* $m = 10$, $q = 14$ *and uniform octrees with grain size of* $16K$ *octants per compute node.*

in [94]. However, that algorithm did not take into account the load imbalance arising from local refinement and therefore, its performance degraded rapidly for large, highly non-uniform octrees, which we use in our scalability experiments. Table 2.16 compares weak scalability of the original algorithm and the new algorithm presented in Section 2.5.2. The new algorithm is over 50 times faster for this input case.

cores	$N_{\text{leaf}}/core$	ORIGINAL	NEW
16	16752	0.2271	0.0126
64	15664	0.5969	0.0152
256	15143	1.6212	0.0186
1024	15107	5.5016	0.0255
2048	15101	9.2205	0.0291
4096	15106	12.9548	0.0441
8192	15113	27.7878	0.4992

Table 2.16 *Weak scaling for 2:1 balance refinement for the original algorithm [94] and Algorithm 3.*

2.9 Comparison with Other Methods

In this section we present numerical results to compare the performance of our PVFMM library with other codes.

		MADNESS			PVFMM					
ν	cores	$\|\|e_u\|\|_2$	T_{Tree}	T_{solve}	N	$\|\|e_u\|\|_2$	T_{Tree}	T_{Setup}	T_{FMM}	*cycles/N*
20	16	1.2E-5	1.8	1.9	1.6E+6	9.1E-6	0.22	0.11	0.19	4.9E+3
40	16	1.3E-5	7.9	6.6	6.7E+6	9.3E-6	0.53	0.56	0.77	5.0E+3
80	16	2.1E-5	33.1	23.2	2.5E+7	1.5E-5	1.74	2.77	3.07	5.2E+3
160	64	1.8E-5	60.2	48.9	1.0E+8	1.6E-5	4.77	3.07	3.14	5.3E+3

Table 2.17 *Comparison with* MADNESS *for a Poisson problem with analytical solution* $u = \exp(-(r/R)^\nu)$ *for different values of* ν*, solved to about 5-digits of accuracy using 10th order discretization.*

Comparison with MADNESS. In Table 2.17, we compare our method with the approach using wavelet decomposition [57, 35] implemented in the MADNESS library. We solve an analytical Poisson problem with the solution $u = \exp(-(r/R)^\nu)$ to about 5-digits of accuracy in L_2 norm for different values of ν and fixed $R = 0.3$. Increasing ν by $2\times$ increases the number of unknowns by roughly $4\times$. For the case $\nu = 160$, we use four compute nodes since MADNESS required more memory than what was available on a single compute node. We used $q = 10$ and $m = 6$ for PVFMM and 10th order discretization for MADNESS. We report the time for tree construction, setup and solve. We also report the number of unknowns (N) and cost per unknown ($cycles/N$) for PVFMM . In each case, our PVFMM code is about an order of magnitude faster than MADNESS.

EXAFMM						PVFMM				
error	N_{crit}	m	θ	T	$cycles/N$	error	N_{crit}	m	T	$cycles/N$
6.1E-2	40	4	0.99	0.28	8.7E+3	1.3E-1	100	2	0.44	1.4E+4
9.9E-3	40	4	0.52	0.89	2.8E+4	8.2E-3	100	4	0.63	2.0E+4
1.2E-3	40	6	0.43	1.50	4.7E+4	2.3E-4	100	6	1.09	3.4E+4
9.1E-6	40	8	0.27	12.32	3.8E+5	4.5E-6	250	8	1.39	4.3E+4
1.1E-6	40	8	0.21	23.77	7.4E+5	3.5E-7	250	10	1.62	5.0E+4
9.2E-8	40	8	0.15	49.67	1.5E+6	7.7E-9	250	12	2.13	6.6E+4
–	–	–	–	–	–	4.1E-12	250	18	4.69	1.5E+5

Table 2.18 *Results for computing Coulombic potential and force for $N = $ 1E+5 source and target particles on a single core of Intel Xeon E5-2687W. We optimized parameter values of both codes to minimize the total solve time for the given accuracy.*

Comparison with EXAFMM. In Table 2.18, we compare our particle code with EXAFMM [111] for an N-body problem with 1E+5 source and target particles randomly distributed in the domain. We compute the Coulombic potential and force at each target point and report the total time (tree construction, setup and solve time) for different solution accuracies. For extremely low accuracy the two methods have comparable performance; however, for single precision and higher accuracies our KIFMM based scheme is over an order of magnitude faster.

In addition to MADNESS and EXAFMM, we have also compared our method to other parallel fast solvers for the constant-coefficient Poisson's equation in [40]. Our volume FMM was an order of magnitude faster than a high-order geometric multigrid (GMG) method for the same accuracy. In that work, we solved a Poisson problem with over half-trillion unknowns in 92s on $229k$ CPU cores.

2.10 Conclusions

We have discussed our implementation of the PVFMM software library. It is a high-order accurate, adaptive and scalable library for computing particle and volume potentials for elliptic kernels such as Laplace, Stokes, Biot-Savart, Helmholtz (low frequency) and modified Laplace. With new modifications using backward stable pseudoinverse, we can achieve convergence up to 14-digits of accuracy. We have discussed several performance improvements for near and far-field interactions. Our volume FMM is optimized to utilize NVIDIA GPU or Intel Phi coprocessors. 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. We have demonstrated scalability of our method to thousands of processors on the Stampede platform at TACC.

3 A Volume Integral Equation Solver for Stokes Flow in Porous Media Geometries

We present a novel volume integral equation method for solving the Stokes equation with variable coefficients on the unit box. Our scheme is based on our volume FMM discussed in Chapter 2 and is high-order, adaptive and scalable. As an application example, we simulate Stokes flow in a porous medium with highly complex pore structure using a penalty formulation to enforce the no slip condition. In our largest scalability test, we solved a problem with 20 billion unknowns on 2048 nodes of the Stampede system at the Texas Advanced Computing Center and achieved 0.656 PFLOP/s for the overall code and one PFLOP/s for the volume integrals.

3.1 Introduction

We propose an algorithm for the Stokes equation in the unit cube with variable coefficients, which can be stated as

$$\rho u - \operatorname{div}\left(\mu(\nabla u + \nabla u^T)\right) + \nabla p = f, \quad \operatorname{div} u = 0. \tag{3.1}$$

Here, $u = u(x)$ is the velocity, $p = p(x)$ is the pressure, $f = f(x)$ is a momentum source, and $x \in [0, 1]^3$. The first equation is the conservation of momentum and the second the conservation of mass—also known as the incompressibility condition. Periodic and free-space boundary conditions on the faces of the unit cube can be applied. The variable coefficients $\rho = \rho(x)$ and $\mu = \mu(x)$ are related to the fluid density and viscosity respectively.

Eq. (3.1) models incompressible flows (steady or unsteady upon temporal discretization) in which the convective inertia is neglected. Such flows are found in microfluidics, biofuels, emulsions, polymers, porous and fractured media flows, and geophysical flows. It can also be used to model incompressible elastic materials with a homogeneous matrix phase and a random distribution of micron-sized particles in the matrix like reinforced

This chapter is based on work that has been published in [77].

elastomers, microgel suspensions, and biological tissues. Solvers for Eq. (3.1) can be used as part of implicit-explicit time-stepping schemes for Navier-Stokes problems, in which the nonlinear convection is treated explicitly.

Designing numerical methods for Eq. (3.1) is challenging. The main difficulties are summarized below.

- It requires four unknowns per spatial grid point in three dimensions (three velocities and one pressure).
- Satisfying the incompressibility condition accurately is hard but crucial for obtaining the correct results.
- We cannot use arbitrary discretization spaces for the velocity and pressure because the $\inf\sup$ condition [55] must be satisfied.
- Discretizations of Eq. (3.1) result in ill-conditioned systems. The need for different discretization spaces for velocity and pressure necessitates block preconditioners.
- It is an elliptic but indefinite problem, which further complicates the construction of fast linear algebraic solvers and preconditioners, especially for problems with highly variable coefficients or high-order discretizations [15].

Due to the importance of Stokes solvers, sophisticated techniques have emerged that can tackle the challenges described above. Discretizing and solving Eq. (3.1) is typically done using finite element methods (FEM) and, to a lesser extent, using finite-difference or finite volume methods [55]. Many theoretically-optimal technologies have been developed for constant and variable coefficients. In practice, however, most existing codes that have been scaled to large core counts have demonstrated scalability only for low-order implementations, typically first- or second-order accurate [16, 17].

3.1.1 Contributions

We propose a scheme that circumvents most of the challenges associated with stencil-based discretizations. Writing $\rho(x)$ and $\mu(x)$ as perturbations around constant values ρ_0 and μ_0, that is $\rho(x) = \rho_0 + \tilde{\rho}(x)$ and $\mu(x) = \mu_0 + \tilde{\mu}(x)$, it is possible to transform Eq. (3.1) to a second-kind volume integral equation for the velocity u *only*:

$$u + \mathcal{G}[\tilde{\rho}u] + \mathcal{D}[\tilde{\mu}(\nabla u + \nabla u^T)] = \mathcal{G}[f], \tag{3.2}$$

where $\mathcal{G}$ is a convolution operator with a boundary condition-dependent Green's function for the Stokes problem and $\mathcal{D}$ is a convolution operator with the transpose of the gradient of the Green's function. The Green's function correspond to any constant-coefficient values ρ_0 and μ_0, e.g., the average values of μ and ρ. The derivation and precise expressions are stated in Section 3.2. In other words, to solve Eq. (3.1) we solve Eq. (3.2) for the velocity u. Once the velocity has been computed, the pressure and overall stress can also be computed by evaluating appropriate convolution integrals.

Our formulation has several nice characteristics. It results in a *second-kind Fredholm equation* with a condition number that is independent of the mesh refinement; it depends only on the magnitude of $\tilde{\rho}$ and $\tilde{\mu}$. The *incompressibility condition* is satisfied pointwise due to the Green's function formulation. The equations for velocity and pressure are *decoupled*. We can evaluate the pressure as a post-processing step. There is *no* $\inf\sup$*-type restriction* on approximating p.

Formulations like Eq. (3.2) for variable-coefficient boundary value problems are well known [24]. We showed a solution of the Lippmann-Schwinger problem using our solver in Fig. 1.1b. However, such formulations have not been used for Stokes problems. One possible reason is the lack of technologies for evaluating $\mathcal{G}$ and $\mathcal{D}$ accurately and efficiently. $\mathcal{G}$ and $\mathcal{D}$ are formally dense operators. They are convolutions, but for non-uniform discretizations, fast Fourier transforms cannot be used. A second possible reason is that the singularities in $\mathcal{G}$ and $\mathcal{D}$ make their computation extremely expensive, resulting in huge constants in the complexity estimates. The third possible reason is that the method requires the knowledge of a Green's function that accounts for the boundary conditions. Thus, the method is restricted only to simple geometries. In this chapter, we try to address these issues.

Since our solver supports variable coefficients, the case of complex geometries can be treated with several methods. For low-order approximations a penalty formulation can be used. We present an example in this chapter. For high-order accuracy ideas similar to overset grids [20] and domain decomposition methods [19] can be used. The geometry is first decomposed into subdomains that are diffeomorphic to the unit cube and matched with appropriate boundary conditions. Each subdomain can be solved with our scheme. Algorithms for high-order accuracy and complex geometries are discussed in Chapter 4.

Our contributions can be summarized as follows:

- We present a volume integral formulation of Stokes equation and demonstrate the feasibility of the approach.

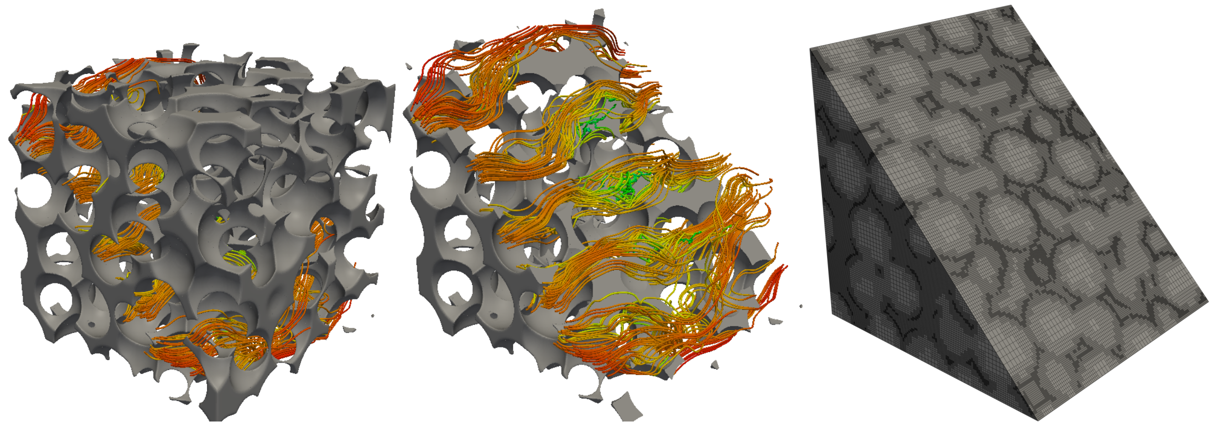


Figure 3.1 *Here we illustrate the capabilities of our solver. We simulate Stokes flow through a porous medium. From left: in the* ***first*** *figure, the gray color indicates the solid phase geometry and the space in between is the pore space. We also visualize the velocity field using streamlines. In the* ***second*** *figure we show the same geometry with clipping to better visualize the streamlines. The* ***third*** *figure shows the leaves of the octree. Weak scalability results for this problem are reported in Fig. 3.4 and Table 3.7.*

- We conduct a performance study and report time-to-solution for various smooth and discontinuous problems.
- We demonstrate scalability on single core, GPUs, MIC, and MPI architectures (Section 3.4) and report, to our knowledge, one of the largest, high-order Stokes runs.
- We apply it to porous medium flows in complex geometries using a penalization approach. An example is shown in Fig. 3.1 and scalability results in Fig. 3.4 and Table 3.7.
- We make our code freely available[1].

3.1.2 Related Work

An integral equation formulation for Navier-Stokes in two dimensions was reported in [50], but the formulation and algorithms were specific to a disk geometry and do not generalize to arbitrary geometries in 2D or to 3D. We are not aware of any other work on volume integral equation formulations for Stokes equation with variable coefficients. Of course, there is a lot of work on solvers for Stokes *boundary integral* equations [82, 86] but they are not applicable for discretizing Eq. (3.1) for arbitrary ρ and μ.

[1] http://padas.ices.utexas.edu/sc14stokes.tgz

For finite element methods for steady Stokes problems we refer the reader to [12, 55]. For unsteady Navier-Stokes, most solvers are based on pressure-projection schemes, which do not work well for stationary problems. For Stokes solvers, state-of-the art implementations include [16, 17], and [64]. The latter is done with `Deal.II` [9, 10], an open source package. One of the most scalable Stokes runs is reported in [18] in which the authors solve problems with up to two billion unknowns on 120K cores using linear elements.

The majority of scalable finite element codes for the Stokes equation use low-order discretizations. An exception is the work in [15] in which the author studies the convergence rates of different high-order discretizations along with the costs of solving the related algebraic system of equations. Multigrid-accelerated block preconditioners result in mesh-independent behavior for all orders. However, the constants deteriorate with increasing approximation order. A cubic velocity-linear pressure ($Q3-Q1$) discretization required 41 iterations for six orders of magnitude reduction in the Krylov residual, whereas a seventh-order velocity-fifth order pressure ($Q7-Q5$) discretization required 95 iterations.

For porous media flow there is a lot of work for Darcy models, but less work on scalable algorithms for Stokes flow. Typically meshing is prohibitively expensive or not scalable and a penalty formulation similar to ours (described in Section 3.2.3) is used. A discussion on the need and importance of solvers for porous media flows can be found in [92] which also uses a penalty formulation (with a finite volume scheme).

3.1.3 Limitations

Our work is the first study of its kind and as such is not comprehensive. We consider the formulation, analysis, convergence tests and scalability studies only for the case of variable density in free space. Here we are not considering high-order accurate scheme for complex geometries, generic boundary conditions, or variable viscosity problems. Also, we are not considering the design of preconditioners for Eq. (3.2). As we can see from the results in Sections 3.3 and 3.4, although the condition number of Eq. (3.2) is mesh-independent, it does depend on the magnitude of the variable coefficients.

3.1.4 Organization of the Chapter

We discuss the formulation and the numerical algorithms in Section 3.2. In Section 3.3, we show convergence results for our methods. We provide strong and weak scalability results in Section 3.4. In Table 3.1, we list some frequently used symbols for easy reference.

Symbol	Definition
Ω	Computational domain: $[0,1]^3$
G	Green's function (kernel function)
m	Multipole order
q	Chebyshev polynomial degree
ϵ_{tree}	Tolerance for adaptive refinement
L	Maximum tree depth
N_{oct}	Number of leaf octants
$N_{\text{cell}} = (q+1)(q+2)(q+3)/2$	Degrees of freedom per leaf
$N_{\text{dof}} = N_{\text{oct}} N_{\text{cell}}$	Number of unknowns
p	Number of compute nodes
T_{solve}	Total solve time
ϵ_{gmres}	GMRES residual
N_{iter}	GMRES iterations

Table 3.1 *Index of frequently used symbols.*

3.2 Methodology and Algorithms

To explain our scheme we will consider the case with free-space boundary conditions. Periodic boundary conditions can be implemented using either the same free-space Green's function by tiling $\mathbf{R}^3$ or by using a problem-specific Green's function. We assume that $\tilde{\rho}$, $\tilde{\mu}$, and f are compactly supported in $\Omega = [0,1]^3$. We only consider the variable density case and set $\mu = 1$. Under these simplifications, Eq. (3.1) becomes

$$\rho_0 u + \tilde{\rho} u - \Delta u + \nabla p = f, \quad \operatorname{div} u = 0. \tag{3.3}$$

To derive the integral equation we need to introduce the Green's function $G(x,y)$. By construction, $G(x,y)$ satisfies

$$\begin{aligned} \rho_0 G(x,y) - \Delta_x G(x,y) + \nabla_x G_p(x,y) &= \delta(x-y), \\ \operatorname{div}_x G(x,y) &= 0, \end{aligned} \tag{3.4}$$

where $G_p(x,y)$ is the corresponding pressure kernel and δ is the Dirac delta function. For a function f, the convolution of G with f is denoted by $\mathcal{G}[f]$. For the free-space case and with $\rho_0 = 0$, $\mathcal{G}$ is given by [82]

$$\mathcal{G}[f](x) := \int_{y\in\Omega} G(x-y) f(y) = \int_{y\in\Omega} \frac{1}{8\pi} \left(\frac{1}{|r|} + \frac{rr^T}{|r|^3} \right) f(y), \tag{3.5}$$

where $r = x - y$. We also set $\rho_0 = 0$ so that $\rho = \tilde{\rho}$. Then, by taking the convolution of the momentum equation in Eq. (3.3) with G, integrating by parts the $G\Delta u$ and $G\nabla p$ terms, and using Eq. (3.4), we obtain

$$u(x) + \mathcal{G}[\rho u](x) = \mathcal{G}[f](x), \ \forall x \in \Omega. \tag{3.6}$$

Eq. (3.6) is the main equation we will be considering here. As mentioned in the introduction, Eq. (3.6) is a second-kind Fredholm integral equation. Thus, it has a bounded condition number [63]. The condition number increases with increase in the norm of ρ.

The expressions for G and its gradient for the case with variable viscosity or for the case with non-zero ρ_0 are quite cumbersome and are not discussed here. Nothing changes in the formulation for those cases, but the case of $\rho_0 \neq 0$ is computationally more expensive because G is not scale invariant anymore.

3.2.1 Discretization

The numerical problem can be stated as follows: given function evaluators for ρ, f and a target accuracy, compute an evaluator function for u. To do this, we use a Galerkin scheme in which we represent ρ, f and u in the same basis. This basis is constructed by decomposing Ω into a set of disjoint cells Ω_i (in our case the leaf-octants of an octree partitioning of Ω). In each Ω_i, we approximate $\rho(x)$, $u(x)$, and $f(x)$ using Chebyshev polynomials of degree q. We call this representation the **Chebyshev tree** of a function. Using a Galerkin projection on Eq. (3.6), we obtain a finite-dimensional algebraic system for the coefficients of u at each Ω_i. This system is non-symmetric. We solve it with unrestarted, unpreconditioned, Generalized Minimum Residual method (GMRES) implemented in the PETSc library [8]. The evaluator function for u will use the Chebyshev coefficients of u at each Ω_i. We outline these steps in detail below.

Building the Chebyshev tree. We partition Ω to Ω_i so that ρ and f are both resolved to a prespecified accuracy in the Chebyshev space. Let's consider the case for f. A Chebyshev octree representation of f is a tree in which at every leaf Ω_i, we represent f by its Chebyshev coefficients $\hat{f}_{nm\ell}$ so that

$$f(x) = \sum_{n=0,m=0,\ell=0}^{n+m+\ell\leq q} \hat{f}_{nm\ell} T_{nm\ell}(x), \quad x \in \Omega_i,$$

where $T_{nm\ell}(x) = T_n(x_1)T_m(x_2)T_\ell(x_3)$ and T_n is the n^{th} degree Chebyshev polynomial. We proceed in a top-down fashion. Recall that we are given an evaluator, which for any x returns $f(x)$. To construct the tree, we start at the root level and we evaluate $\hat{f}$ by sampling f at Chebyshev points and then taking a discrete Chebyshev transform [102]. That is, we use Chebyshev quadrature to evaluate $\hat{f}_{nm\ell} = \int_{\Omega_i} T_{nm\ell} f(x) w(x)$, where $w(x) = (1-x^2)^{-1/2}$ is required for orthogonality. Then, we estimate the truncation error from the tails: if $\sum_{n+m+\ell=q} |\hat{f}_{nm\ell}| \leq \epsilon$ we terminate the recursion, otherwise we subdivide and continue recursively for each child octant of the root. If f is discontinuous we stop when we reach a prespecified maximum tree level. Once the tree is constructed, we have a representation for f in terms of piecewise Chebyshev polynomials. Notice that we are *not* using a tensor product basis. Instead we truncate so that the highest degree of the polynomial is q. This results in N_{cell} coefficients per leaf octant (roughly $(q+2)^3/2$ instead of $3(q+1)^3$).

Convolution with $\mathcal{G}$. Once we have built the Chebyshev trees for ρ and f, we merge them to a single tree that represents both functions accurately. Then, we evaluate the action of $\mathcal{G}$ on Chebyshev tree functions. For a given function f, computing $u = \mathcal{G}[f]$ corresponds to solving a constant-coefficient Stokes problem with right-hand side equal to f. This is the most expensive step in our method and is done using our volume FMM described in Chapter 2. The result is a piecewise polynomial discretization of u using the same basis as the Chebyshev tree for f.

Evaluating $\mathcal{G}[\rho u]$. We assume that we have an evaluator for $\rho(x)$ and N_{cell} Chebyshev coefficients per octant for u. To compute the convolution of their product, we first evaluate u at the $(q+1)^3$ Chebyshev node points in every octant and we multiply the values pointwise to get $\rho(x_k)u(x_k)$ at the Chebyshev points. We take their Chebyshev transform to compute the Chebyshev coefficients for $\rho(x)u(x)$ and then, we apply $\mathcal{G}$ as described above. To evaluate u at the Chebyshev node points, we use tensor-product transformations after padding the coefficients with zero. This has $\mathcal{O}\left(q^4\right)$ complexity.

Overall Scheme. The overall scheme can be summarized as follows

- Create a Chebyshev tree based on approximating f and ρ to a desired accuracy. The resulting tree has N_{oct} octants.
- Evaluate $\mathcal{G}[f]$ using volume FMM.
- Solve Eq. (3.6) using GMRES, in which the operator $\mathcal{G}[\rho u]$ is implemented by computing ρu pointwise followed by computing a Chebyshev transform and then computing the convolution with $\mathcal{G}$ using the volume FMM.

To summarize, let P_N be the projection operator that restricts f, u and ρ, to the space spanned by N_{dof} polynomials. Then the discretized system can be written as

$$u_N + \mathcal{G}_N[\rho u_N] = \mathcal{G}_N[f], \tag{3.7}$$

where $\mathcal{G}_N[\cdot] = P_N \mathcal{G}[P_N \cdot]$ denotes the discretization of $\mathcal{G}$. Notice the degrees of freedom depend on q but for notational simplicity we suppress this dependence. We remark again that this linear system is not symmetric, unlike the original Stokes problem.

3.2.2 Error Analysis

We follow standard error analysis for projection methods for second kind operators [63]. Let u_N to be the solution of Eq. (3.6). Then the overall error can be estimated by,

$$\|u_N - u\|_2 = \mathcal{O}\left(\|P_N u - u\|_2 + \|\mathcal{G}_N - P_N \mathcal{G}\|_2 + \|P_N f - f\|_2\right).$$

The constant in the estimate is proportional to the norm of $\mathcal{G}$, which in turn depends on $\|\rho\|$. The first term is the approximation error due to the projection, the second term is error due to quadratures (related to the smoothness of ρ) and FMM, and the last term is the approximation error of f.

Assuming standard regularity, i.e., $\rho(x) \in L^\infty$ and $f \in L^2$, then $u \in H^2$. If ρ and f are in C^∞ the convergence is of order q. Otherwise the convergence depends on the regularity of u and ρ. For constant-coefficients $\rho = 0$ and the middle term drops. Then the error in u becomes directly proportional to the error in f.

3.2.3 Formulation for Porous Media Flow

Let $\xi(x)$ be the characteristic function of the fluid phase and $1-\xi(x)$ the characteristic function of the solid phase. Then Eq. (3.6) is satisfied in the fluid phase and $u = 0$ in the solid phase. For regular pore geometries the right approach is to use a double-layer potential formulation using boundary integral equations. But for complex geometries like the one in Fig. 3.1, constructing surface meshes can be complicated and expensive due to the scalability of meshing and constructing multilevel preconditioners. When engineering accuracies are acceptable (say 1% error), one can approximate the solution using a penalty formulation. This is a classical approach similar to fictitious domain methods, immersed boundary methods, embedded methods and others. To force the fluid to have zero velocity we use a volume penalty method in which $\rho(x) = \eta(1 - \xi(x))$, where η is the penalty parameter. This can easily be derived by a constrained variational formulation of the Stokes equation in which $u(x) = 0$ for $\xi(x) = 0$. With this approximation, the formulation becomes

$$\eta(1 - \xi(x))u(x) - \Delta u + \nabla p = f, \quad \operatorname{div} u = 0,$$

and its volume integral formulation becomes

$$u(x) + \eta \mathcal{G}[(1-\xi)u] = \mathcal{G}[f]. \tag{3.8}$$

The theoretical analysis of the scheme for Stokes equation can be found in [4]. Let us denote the solution of Eq. (3.8) as u_η and u_* be the exact solution. It can be shown that u_η converges to u_* as η goes to infinity, in the L^2 norm. If the solution u is regular, the convergence rate is expected to be $\mathcal{O}(1/\eta)$. If the solution is not regular the convergence rate can deteriorate to $1/\eta^{1/a}$, with $a = 2$ or even $a = 4$. Let us remark that the use of penalty formulations for porous media is not new, for example it has been used in [92].

3.3 Numerical Results

In Chapter 2, we have already showed convergence results for the constant-coefficient case. Here, we consider two problems with analytic solutions. The first test cases correspond to a C^∞ velocity field. For the second test the solution is in H^2. We consider convergence as a function of the polynomial degree q, the GMRES tolerance ϵ_{gmres}, the discretization error $\epsilon_{\text{tree}} = \|P_N f - f\|_\infty$ and the order of multipole expansions m. All timing results reported here are for single Sandy Bridge core running at 2.7GHz.

In the **first test**, we consider the case with $\rho = \rho_0 \exp(-500|x|^2)$ and study the convergence of the method as we decrease the tree-refinement tolerance ϵ_{tree} and correspondingly choose the best Chebyshev degree q and multipole order m for the fastest time to solution for a given accuracy. The analytical solution for velocity is given by $u(x) = \exp(-500|x|^2)(x_3\mathbf{e_2} - x_2\mathbf{e_3})$, where $\mathbf{e_2}$ and $\mathbf{e_3}$ are orthogonal unit vectors. We report the L^∞ and L^2 errors in the velocity field, the number of GMRES iterations and the total time to solution in Table 3.2. For fixed ρ, the number of iterations increase with increasing mesh size because we tighten the GMRES tolerance as we refine. The dependence of the GMRES iterations on large variations of ρ is mild up to fiver orders of magnitude variations. We observe fast convergence but the conditioning deteriorates with increasing $\|\rho\|_\infty$. For four digits of relative accuracy pointwise, the number of GMRES iterations is six for $\|\rho\|_\infty = 1\text{E}{+}5$. For higher $\|\rho\|_\infty$, we need tighter GMRES residuals due to the deterioration of conditioning and the number of iterations jumps to 100s of iterations. Although the timings are not bad (just 13 minutes on a single core for the most expensive run), for large variations of ρ preconditioning should be used.

In the **second test case**, we consider a problem with discontinuous ρ. We solve for Stokes flow around a sphere and check our result with the analytic solution. The solu-

ϵ_{tree}	$\|\rho\|_\infty$	N_{oct}	ϵ_{gmres}	N_{iter}	L^∞	L^2	T_{solve}
1E-1	1E+5	64	1E-3	5	8.3E-3	1.5E-2	3.91
1E-2	1E+5	120	1E-4	6	5.7E-4	7.0E-4	12.4
1E-3	1E+5	176	1E-7	14	4.6E-7	8.4E-7	45.4
1E-6	1E+5	736	1E-8	17	6.8E-8	1.9E-7	229
1E-1	1E+7	64	1E-4	12	2.2E-2	5.0E-2	9.45
1E-2	1E+7	120	1E-7	61	2.3E-4	3.4E-4	127
1E-3	1E+7	176	1E-9	103	7.6E-7	1.9E-6	337
1E-1	1E+9	64	1E-4	12	2.3E-2	5.5E-2	9.44
1E-2	1E+9	120	1E-7	82	6.2E-4	2.4E-3	171
1E-3	1E+9	176	1E-9	246	1.9E-5	1.6E-4	818

Table 3.2 *Convergence for a variable-coefficient Stokes flow, with reducing tree refinement tolerance ϵ_{tree} and GMRES tolerance ϵ_{gmres} for different values of ρ. Chebyshev degree $q = 14$, multipole order $m = 10$; T_{solve} is the overall solve time in seconds on a single core.*

tion in the exterior of the sphere is smooth but if we extend the velocity by zero inside the sphere, its derivatives are discontinuous at the boundary of the sphere. We approximate the solution of this problem using the penalty formulation and solving in the whole domain. For convergence, we must increase η as we refine and the problem becomes

$\|\rho\|_\infty$	L	N_{oct}	ϵ_{gmres}	N_{iter}	L^∞	L^2	T_{solve}
2.5E+5	1	1	4.0E-6	4	2.6E-1	2.4E-1	0.1
5.0E+5	2	8	2.0E-6	29	1.7E-1	8.9E-2	1.2
1.0E+6	3	64	1.0E-6	36	1.2E-1	4.8E-2	4.9
2.0E+6	4	120	5.0E-7	43	4.6E-2	6.8E-3	17
4.0E+6	5	512	2.5E-7	44	2.0E-2	8.8E-4	43

Table 3.3 *Convergence for Stokes flow around a sphere of radius=0.15 (**variable and discontinuous coefficients**) in $\Omega = [0,1]^3$ with decreasing GMRES residual ϵ_{gmres} and increasing penalty $\|\rho\|_\infty$ and tree refinement level L. Chebyshev degree $q = 14$, multipole order $m = 10$; T_{solve} is the overall solve time in seconds on a single core.*

increasingly ill conditioned. Notice that we do *not* advocate using our method for this particular problem. Boundary integrals should be used instead. We just use it to illustrate the convergence rate of our scheme in this setting. The L^2 and L^∞ errors are measured on the exterior of the sphere only.

In Table 3.4, we examine the cost for different discretization orders. We keep the solution accuracy fixed (1% error) and we vary the polynomial order q and the FMM far-field accuracy m. In all of these examples, we set the penalty parameter to $\eta = $ 1E+7. As we expected, higher-order elements will not help with the convergence rate but they can help

q	m	N_{oct}	N_{iter}	L^∞	T_{solve}
14	10	$1,856$	45	1.4E-2	140
14	6	$1,856$	45	1.4E-2	89
6	6	$5,328$	45	1.0E-2	38
4	6	$5,328$	53	1.4E-2	35
2	6	$20,952$	46	1.2E-2	103

Table 3.4 *Single node performance results for flow around a sphere with fixed accuracy and different Chebyshev degrees q and multipole orders m. Here we examine the effect of using a high-order discretization for a non-smooth problem. The timings are on a single compute node using 16 CPU cores.*

with the constants by allowing faster convergence away from the discontinuity. From this table we observe that using high-order approximation (high q and high m) increases the cost significantly and should not be used. The cases of $q = 6$ and $q = 4$ give similar timings.

3.4 Performance Analysis

In this section, we analyze the performance of our implementation. Below, we briefly describe the experimental setup used in this work.

Hardware. All experiments were performed on the Stampede system at TACC, consisting of 6,400 compute nodes connected by 56GB $/s$ FDR Mellanox InfiniBand network in a fat tree configuration. Each compute node has dual eight-core Intel Xeon E5-2680 CPUs running at 2.7GHz and 32GB of memory. In addition, most nodes have an Intel Xeon Phi SE10P co-processor, while a few have an NVIDIA K20 GPU co-processor. The system has a theoretical peak performance of 1.42TFLOP/s per node (345.5GFLOP/s for CPU and 1.07TFLOP/s for Phi) and about 9PFLOP/s for the entire system. Of this, Stampede achieved 5.2PFLOP/s with 6,006 compute nodes on the LINPACK benchmark. For this work, we had access to 2048 compute nodes and for our largest run we achieved 656TFLOP/s or 22.4% of the theoretical peak performance.

Software. We use the Intel compiler version 13.1.0 along with Intel MPI Library 4.1 to compile our code. We use PETSc-3.4.3, FFTW3/3.3.2 and Intel MKL-11.0.1 libraries for the BLAS operations and NVIDIA CudaBLAS for the GPU version.

						CPU		CPU+Phi		CPU+GPU	
q	N_{oct}	L	N_{iter}	L^∞	L^2	T_{solve}	GFLOP/s	T_{solve}	GFLOP/s	T_{solve}	GFLOP/s
8	32,768	6	48	3.1E-6	1.0E-6	1026.4	148	940.0	162	942.9	161
10	32,768	6	46	1.0E-7	3.0E-7	1157.9	161	963.3	194	952.1	196
12	4,096	5	49	3.2E-6	1.4E-6	183.5	184	129.3	262	123.9	273
14	4,096	5	47	1.7E-7	3.1E-7	259.3	210	138.0	394	128.5	424
16	4,096	5	46	8.3E-8	3.1E-7	384.3	242	159.5	582	140.2	662
6	29,240	9	41	4.8E-7	4.0E-7	722.6	148	681.0	157	669.1	159
8	4,656	8	41	6.6E-7	6.7E-7	118.8	160	101.0	188	97.4	195
10	2,024	7	41	2.5E-7	3.5E-7	66.9	166	44.3	251	42.4	262
12	1,240	6	44	1.1E-7	3.1E-7	63.4	179	31.4	360	30.3	374
14	736	6	42	9.2E-8	3.0E-7	56.4	192	23.2	467	19.1	565
16	232	5	41	1.5E-7	3.2E-7	38.8	147	15.6	366	9.8	579

Table 3.5 *Single node performance results for a variable coefficient problem ($\|\rho\|_\infty = 1$E+6) for uniform and non-uniform meshes with increasing Chebyshev degree q and fixed multipole order $m = 10$, GMRES tolerance $\epsilon_{\text{gmres}} = 1$E-9. This example demonstrates two orders of magnitude speed-up from a 9th order, uniform grid approximation, to a 17th order GPU-accelerated, adaptive scheme reducing 1026 seconds to 9.8 seconds.*

3.4.1 Single Node Performance

In this section we present runtime and performance (in GFLOP/s) on a single node of Stampede. We use the smooth coefficient test case, described in Section 3.3, with $\rho(x) = 10^6 \exp(-500x^2)$. In Table 3.5, we show results for both uniform and adaptive meshes. We vary the discretization order q for a fixed solution accuracy of about 1E-6. With higher order approximation, we require significantly fewer octree nodes and consequently solve the problem faster. We achieve a speedup of $2.7\times$ and $3.1\times$ on CPU for the uniform and adaptive cases respectively by increasing the discretization order q from 8 to 16. Furthermore, using adaptive mesh requires an order of magnitude fewer unknowns and is about $10\times$ faster for the same q. We can also use either an Intel Phi or NVIDIA GPU to accelerate the near interactions. For low-order discretization, since there is not enough work in near interactions and we do not observe a significant advantage in using co-processors. However, for $q = 16$, we see $2.5\times$ and $2.7 - 4.0\times$ speedups for CPU+Phi and CPU+GPU cases respectively, with GPUs giving significantly better performance even for small problems with just 232 octants. Overall, we have over $100\times$ speedup going from low-order CPU case to high-order, adaptive CPU+GPU case and achieve 579GFLOP/s or about 40% of the theoretical peak performance of the compute node.

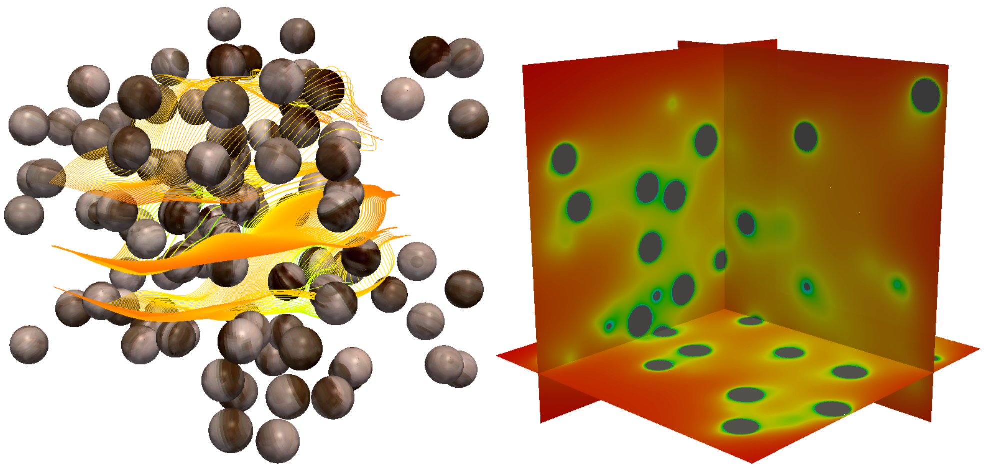


Figure 3.2 *Here, we solve Stokes flow around a random distribution of spheres. On the left we visualize the streamlines around the spheres. The figure on the right shows cross sections of the magnitude of the velocity field with velocity field increasing from green to red.*

3.4.2 Weak Scalability

We now present isogranular or weak scalability results on Stampede. In Fig. 3.3 and Table 3.6, we present results for a low-order case with discretization order $q = 6$ and multipole order $m = 6$. We solve for flow around a distribution of 250 spheres each of radius 5E-2 in a unit cube. The configuration is similar to that visualized in Fig. 3.2. We use the penalty method with $\eta = 1\text{E+}9$ inside the spheres and zero outside. We adaptively refine our mesh on the boundary of the spheres and the problem size is determined by the maximum refinement level L. Here, we vary L from 5 to 10 and increase the number of compute nodes while keeping the number of unknowns per processor (N_{dof}/p) fixed at roughly one million. In Fig. 3.3, we present a breakdown of the time spent in each stage of the solver. Of the total solve time, GMRES corresponds to the time spent internally in the PETSc's Krylov subspace iterative solver. In each iteration of GMRES, a matrix multiplication operation (the LHS in Eq. (3.7)), labeled as MatMul, is performed. In this operation, the convolution with the Green's function is implemented using our volume FMM and we show the time spent in computation (FMMcomp) and communication (FMMcomm). We show results for execution on CPU and CPU+Phi as we increase the number of compute nodes from 1 to 2048. We observe that the FMM accounts for over 60% of the total solve time. As expected, the FMM computation stage is about 15% faster with the co-processor than without it. The communication cost of FMM increases gradually as we increase the number of MPI processes. This trend appears to be consistent with the expected $\mathcal{O}(\log p)$

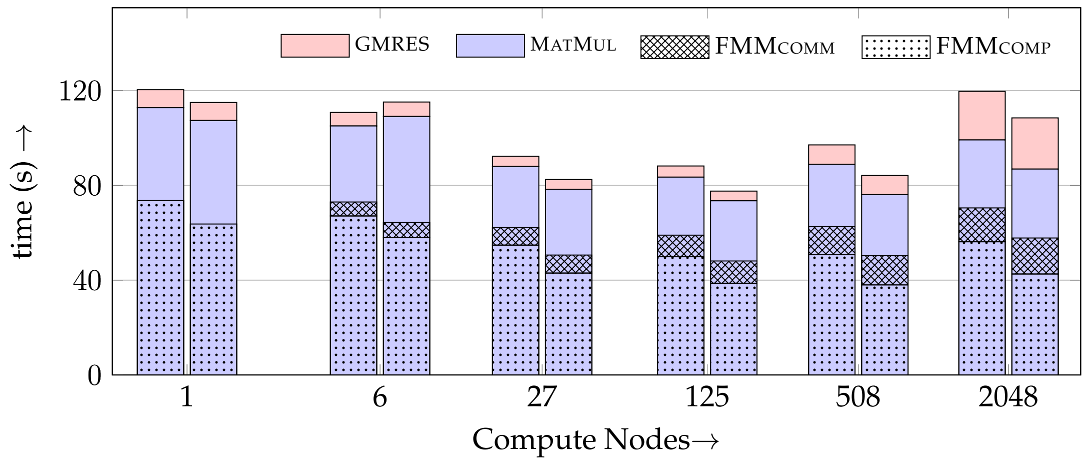


Figure 3.3 *Weak scalability results for low-order discretization ($q = 6$, $m = 6$) showing a breakdown of time in seconds for different stages of the solver for CPU and CPU+Phi runs respectively. We solve for flow around a random distribution of 250 spheres. We report more information in Table 3.6.*

p	N_{dof}/p	N_{iter}	T_{solve}	TFLOP/s	η
1	9.8E+5	200	115.0	0.12	1.00
6	9.5E+5	163	115.2	0.62	0.86
27	1.0E+6	116	82.5	3.12	0.96
125	1.0E+6	99	77.6	13.0	0.87
508	1.1E+6	92	84.2	47.9	0.78
2048	1.1E+6	90	108.5	150	0.61

Table 3.6 *Solve time, total* FLOP *rate and parallel scaling efficiency of the solver for flow around 250 spheres using $q = 6$ and $m = 6$, for weak scaling up to 2K compute nodes on Stampede using Phi co-processor.*

complexity estimate. For very large process counts, the scalability of GMRES appears to suffer and there is a significant increase in the time spent in this stage for 2048 compute nodes. In Table 3.6, we show performance results for CPU+Phi case. Overall the code scales well and on 2048 compute nodes we achieve 150TFLOP/s with 61% parallel scaling efficiency η. We lose some performance due to communication overhead and increase in time spent in the GMRES stage.

In Fig. 3.4 and Table 3.7 we solve for Stokes flow using the geometry visualized in Fig. 3.1. Here we have used a high-order discretization with $q = 14$ and multipole order $m = 10$. In this test case, we see significant speedup ($\sim 2\times$) using the Phi co-processor since we have enough work in the near interactions. Consequently, we also get signif-

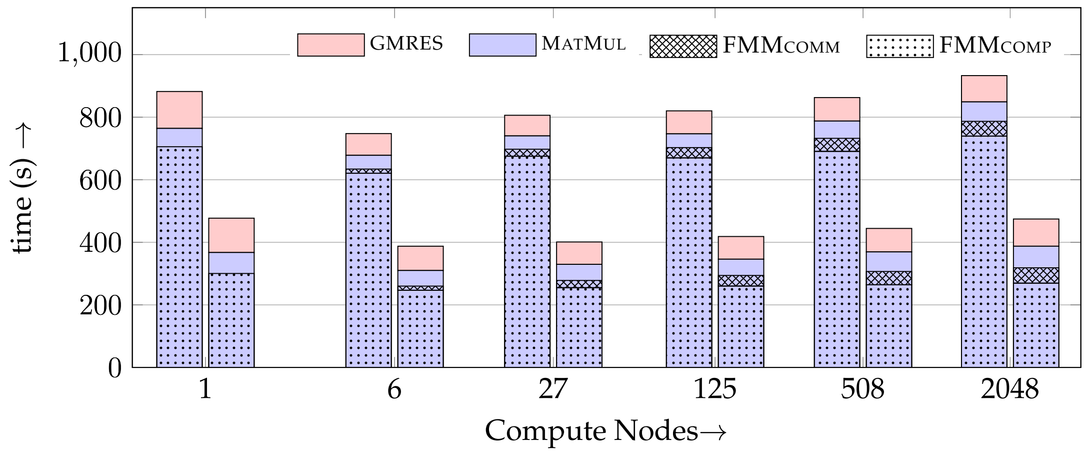


Figure 3.4 *Breakdown of time in seconds for different stages of the solver for CPU and CPU+Phi runs respectively using $q = 14$ and $m = 10$ for the geometry visualized in Fig. 3.1. We get nearly $2\times$ speedup for the CPU+Phi runs over CPU only runs. We observer excellent weak scalability from one to 2048 compute nodes. We report more information for the CPU+Phi run in Table 3.7.*

p	N_{dof}/p	N_{iter}	T_{solve}	TFLOP/s	η
1	8.0E+6	155	477	0.36	1.00
6	7.8E+6	115	388	2.27	1.04
27	8.6E+6	101	401	10.3	1.05
125	8.5E+6	98	419	45.3	0.99
508	8.9E+6	92	444	173	0.94
2048	9.1E+6	90	474	656	0.88

Table 3.7 *Solve time, total FLOP rate and parallel scaling efficiency of the solver for porous media flow, for weak scaling up to 2K compute nodes on Stampede using CPUs and the Intel Phi accelerator. The number of GMRES iterations appear to be independent of the mesh size (in fact they are reducing with increasing mesh size).*

icantly higher FLOP rates, achieving 656TFLOP/s on 2048 compute nodes and a parallel scaling efficiency $\eta = 0.88$. Again the number of GMRES iterations is significant due the large $\|\rho\|_\infty$.

3.4.3 Strong Scalability

We report strong scalability results where we fix the problem size and increase the number of compute nodes. In Fig. 3.5 and Table 3.8, we simulate flow around a random distribution of 250 spheres using high-order discretization ($q = 14$, $m = 10$) and with an adaptive mesh refined to 7 levels corresponding to 232 million unknowns. We solved GMRES to a residual

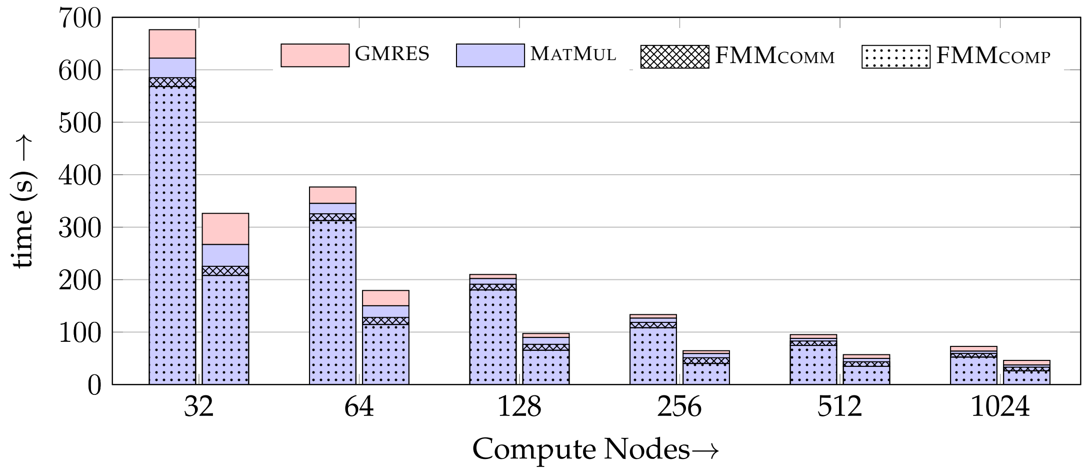


Figure 3.5 *Strong scalability up to 1024 compute nodes using high-order discretization ($q = 14$, $m = 10$) for flow around spheres with 232 million unknowns. We report breakdown of time in seconds for different stages of the solver for CPU and CPU+Phi runs respectively. We report more information in Table 3.8.*

	CPU			CPU+Phi		
p	T_{solve}	TFLOP/s	η	T_{solve}	TFLOP/s	η
32	676	6.11	1.00	326	12.7	1.00
64	377	11.0	0.90	179	23.1	0.91
128	210	19.8	0.81	97	42.7	0.84
256	133	31.4	0.63	65	64.9	0.63
512	95	44.6	0.44	57	74.6	0.36
1024	73	60.0	0.29	46	94.8	0.22

Table 3.8 *Solve time, total FLOP rate and parallel scaling efficiency of the solver for flow over 250 spheres using $q = 14$ and $m = 10$, for strong scaling up to 1K compute nodes on Stampede.*

tolerance $\epsilon_{\text{gmres}} = 1\text{E-}8$ and the solve required 101 iterations. As we increase the number of compute nodes from 32 to 1024, we get $9\times$ and $7\times$ speedup for the CPU and CPU+Phi respectively. Overall, we achieve 94.8TFLOP/s and a parallel efficiency of 22%.

Next, in Fig. 3.6 and Table 3.9, we use the porous medium geometry as a test case and use low order discretization for the solver. The octree is refined to 8 levels and we have 169 million unknowns. Due to the low-order discretization, we gain little from using the Phi accelerator and get about 24TFLOP/s for both CPU and CPU+Phi on 1024 compute nodes. Scaling from 32 compute nodes to 1024 compute nodes, we get similar parallel efficiency as before.

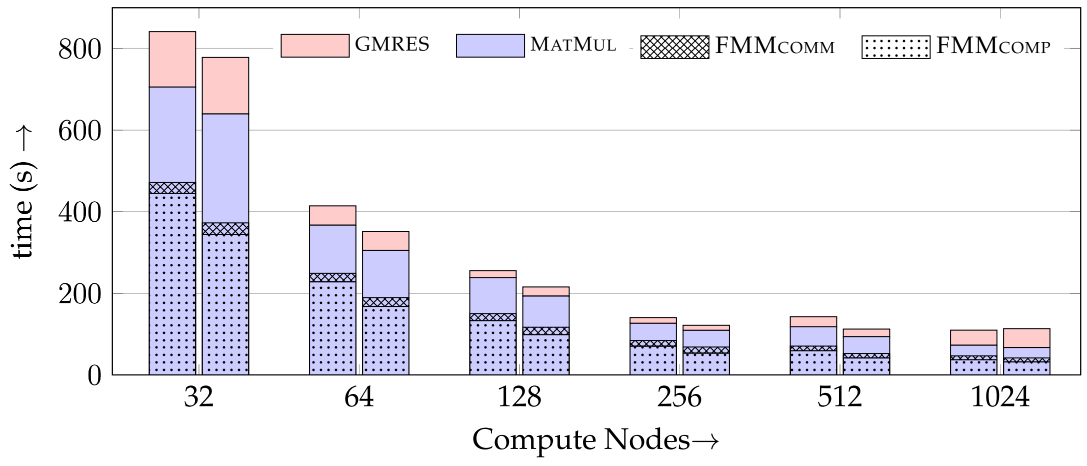


Figure 3.6 *Strong scalability using low-order discretization ($q = 6$, $m = 6$) for flow through a porous medium using 169 million unknowns. We report breakdown of time in seconds for different stages of the solver for CPU and CPU+Phi runs respectively. We give more details in Table 3.9.*

	CPU			CPU+Phi		
p	T_{solve}	TFLOP/s	η	T_{solve}	TFLOP/s	η
32	841	3.0	1.00	778	3.2	1.00
64	414	6.1	1.02	351	7.2	1.11
128	255	9.9	0.83	216	11.7	0.91
256	140	18.1	0.76	122	20.9	0.81
512	142	18.1	0.38	112	22.9	0.44
1024	109	24.0	0.25	113	23.1	0.22

Table 3.9 *Solve time, total* FLOP *rate and parallel scaling efficiency of the solver for flow through a porous media using $q = 6$ and $m = 6$, for strong scaling up to 1K compute nodes on Stampede.*

3.5 Conclusions

We have presented a novel scheme for solving the Stokes equation with variable coefficients. We demonstrated the convergence of our scheme and its efficiency. We showed scalability on hybrid architectures. For smooth problems, we demonstrated that the combination of high-order accuracy, adaptivity, integration with accelerators, algorithmic optimality, and distributed memory parallelism can result in many orders of magnitude speedups.

Here we have only scratched the surface of the capabilities and challenges of the proposed methodologies. There remain challenges in terms of performance, preconditioners, general geometries, and of course careful verification for porous media flows.

For more general geometries and boundary conditions, one can use block solvers as we discussed before. The transformation Jacobian from an arbitrary hexahedral domain to the unit cube, can be easily handled by our variable coefficient solver. As long the number of hexahedral elements is reasonably small and the elements are well shaped (for example in the absence of large anisotropy), we can handle non-cubic geometries by using ideas similar to macromesh construction codes [16], overset grids and other similar ideas. We explore this approach in Chapter 4.

Another issue that we do not discussed here is the need for preconditioning. We saw an increase in the number of iterations as ρ_∞ increases significantly. Many algorithms for preconditioning integral equations exist but none has been scaled to such complexity. Promising schemes include multilevel solvers and hierarchical inexact factorizations.

There are several other issues regarding performance. When the average ρ_0 is not zero, the Green's function is not scale invariant. This means that the precomputed matrices for near interactions will depend on the level of the source leaf octant. This will increase the storage requirements and reduce performance. Variability on the viscosity does not cause any problems.

4 Volume Integral Equation Solver for Stokes Flow in Complex Geometries

In this chapter, we will discuss new volume integral equation (VIE) formulations for Poisson and Stokes equations under coordinate transforms. We will extend our VIE solver discussed in Chapter 3 to these new formulations and use it to solve problems on certain non-regular geometries which can be mapped to a cubic domain.

4.1 Introduction

We consider the solution of constant and variable coefficient elliptic boundary value problems (BVPs) on complex geometries. Embedded boundary methods can be used for such problems. In this approach, a fast solver is used to compute a solution to the inhomogeneous elliptic problem on a regular domain enclosing the target geometry and a fast boundary solver for the homogeneous elliptic problem is used to enforce the required boundary conditions on the geometry of interest. While a lot of progress has been made in this direction, there continue to be many challenges. Due to the presence of cut-cells these methods often require excessive refinement near the boundary and result in low-order convergence. For embedded boundary integral methods, computing the solution close to the boundary requires evaluating near-singular integrals which are often expensive.

In this chapter, we propose a new approach for certain classes of geometries which can be mapped to cubic domains. We reformulate constant and variable coefficient elliptic problems on such geometries to problems on cubic domains. We have developed novel volume integral equation (VIE) formulations for the transformed problem. On cubic domains, we can solve these VIE formulations efficiently using our volume fast multipole method (discussed in Chapter 2) together with iterative linear solvers like GMRES. Our method uses a 2:1 level-restricted adaptive octree and high-order piecewise polynomials to discretize the boundary and volume data in the cubic computational domain. The use of a regular octree discretization allows us to precompute expensive singular and near-singular quadratures for volume and boundary integration and apply them efficiently, achieving high FLOP -rates.

4.1.1 Contributions

We have developed new VIE formulations by mapping constant and variable coefficient elliptic BVPs on certain classes for non-regular geometries to cubic domains. We show well-posedness of our formulation for the case of constant-coefficient Poisson equation.

We have developed an efficient solver for our VIE formulations with Dirichlet boundary conditions. We show feasibility of our approach and present results for convergence and timing-to-solution for constant-coefficient Stokes equation on different geometries.

4.1.2 Related Work

Much work has been done on developing fast solvers for elliptic PDEs in non-regular geometries. This includes boundary integral methods for homogeneous elliptic PDEs on complex domains [6, 58]. For inhomogeneous problems, embedded boundary integral methods can be used, where, the inhomogeneous equation is solved using FFT [110], volume FMM [66] or other fast solvers [80, 78, 13] on a regular domain. However, accurate representation of the density function near the domain boundary (on the cut-cells) can be problematic with regular grids. In [5], this has been addressed by constructing a C^0 extension of the density function for 2D problems. This approach can also be applied to problems in 3D; however, we are not aware of any existing implementation.

Solutions to variable coefficient PDEs can be obtained by iteratively solving a second-kind Fredholm integral equations, where the kernel function in the integral corresponds to a constant-coefficient elliptic PDE [89]. Such formulations have been presented for Poisson, Stokes and Helmholtz problems [31, 77, 3].

4.1.3 Limitations

Our formulation applies to very simple geometries which can be smoothly mapped to cubic domains. It further assumes that the mapping is provided in the form of a function evaluator. This may not be true for problems where only a description of the domain boundary is given and constructing a mapping to a cubic domain may not be trivial.

Our analysis of the existence and uniqueness of the solution only considers the case for constant-coefficient Poisson problem.

4.1.4 Organization of the Chapter

In Section 4.2, we introduce our VIE formulations for the Poisson and Stokes equations and also analyze the well-posedness for the Poisson problem. In Section 4.3, we discuss the discretization and the numerical algorithms for solving the discretized VIE. Finally, we present convergence results for Stokes flow in different geometries in Section 4.4. In Table 4.1, we list some frequently used symbols for easy reference. In Table 4.2, we summarize the kernel functions used in this chapter.

Symbol	Definition
X	Cubic domain
Y	Non-regular domain
$\mathbf{x}(\mathbf{y})$	Coordinate map from Y to X
$\mathbf{y}(\mathbf{x})$	Coordinate map from X to Y
J	$= (\partial y_i / \partial x_j)_{ij}$ (Jacobian)
A	$= J^{-1} J^{-T}$ (Metric tensor)
$\mathcal{G}, \mathcal{D}$	Poisson single and double-layer kernel function
$\mathcal{S}, \mathcal{P}$	Stokes single-layer velocity and pressure kernels
$\mathcal{D}, \mathcal{K}$	Stokes double-layer velocity and pressure kernels
$\bar{\mathcal{G}}, \bar{\mathcal{D}}, \bar{\mathcal{S}}, \bar{\mathcal{P}}, \bar{\mathcal{D}}, \bar{\mathcal{K}}$	kernel functions with homogeneous Dirichlet boundary conditions

Symbol	Definition
ω_i	Leaf node in octree partitioning of X
N_{oct}	Number of leaf octants
L	Maximum tree depth
q	Chebyshev polynomial degree
ϵ_{tree}	Tolerance for adaptive refinement
P_{N_v}	Projection to piecewise polynomial discretization for volume data
P_{N_b}	Projection to piecewise polynomial discretization for boundary data
m	Multipole order
ϵ_{GMRES}	GMRES residual
N_ϕ	GMRES iterations for VIE solve
N_σ	GMRES iterations for BIE solve
T_{solve}	Total solve time

Table 4.1 *Index of frequently used symbols.*

4.2 Formulation

In this section, we discuss the mathematical formulation of our method. Starting with the integral equation formulation for the constant-coefficient Poisson problem on the unit cube, we incrementally build more complex formulations. We discuss variable coefficient Poisson problems and discuss problems under coordinate transformations in Section 4.2.2. In Section 4.2.3 show results for well-posedness of our formulation. We extend our formulation to the incompressible Stokes equation in Section 4.2.4. In Section 4.2.5, we discuss a boundary integral equation formulation for imposing Dirichlet boundary conditions on non-smooth domains. We summarize the overall scheme in Section 4.2.6.

Laplace single-layer	$\mathcal{G}(\mathbf{r}) = \frac{1}{4\pi}\frac{1}{\lvert\mathbf{r}\rvert}$
Laplace double-layer	$\mathcal{D}(\mathbf{r}) = -\frac{1}{4\pi}\frac{\mathbf{r}}{\lvert\mathbf{r}\rvert^3}$
Stokes single-layer velocity	$\mathcal{S}(\mathbf{r}) = \frac{1}{8\pi}\left(\frac{I}{\lvert\mathbf{r}\rvert} + \frac{\mathbf{r}\,\mathbf{r}^T}{\lvert\mathbf{r}\rvert^3}\right)$
Stokes single-layer pressure	$\mathcal{P}(\mathbf{r}) = \frac{1}{4\pi}\frac{\mathbf{r}^T}{\lvert\mathbf{r}\rvert^3}$
Stokes double-layer velocity	$\mathcal{D}(\mathbf{r}) = \frac{3}{4\pi}\frac{(\mathbf{r}\cdot\mathbf{n})\,\mathbf{r}\,\mathbf{r}^T}{\lvert\mathbf{r}\rvert^5}$
Stokes double-layer pressure	$\mathcal{K}(\mathbf{r}) = \frac{1}{2\pi}\left(-\frac{\mathbf{n}^T}{\lvert\mathbf{r}\rvert^3} + \frac{3\,(\mathbf{r}\cdot\mathbf{n})\,\mathbf{r}^T}{\lvert\mathbf{r}\rvert^5}\right)$

Table 4.2 *Summary of free-space kernel functions.*

4.2.1 Integral Equation Formulations

We will now discuss integral equation formulations for constant-coefficient and simple variable coefficient Poisson problems on cubic domains. Through these formulations, we will introduce some basic concepts and notations which will be useful in developing our formulation for complex geometries.

Constant-Coefficient Problems. We consider the constant-coefficient Poisson problem $\Delta u(\mathbf{x}) = f(\mathbf{x})$, on a cubic domain X with $f \in L^2(X)$ and free-space boundary conditions. The solution $u(\mathbf{x})$ is given by the convolution,

$$u(\mathbf{x}) = \int\limits_{\mathbf{y}\in X} \mathcal{G}(\mathbf{x},\mathbf{y}) f(\mathbf{y}) \tag{4.1}$$

where, $\mathcal{G}(\mathbf{x},\mathbf{y})$ is the free-space Green's function for the Poisson problem. This convolution is computed numerically using the volume fast multipole method discussed in Chapter 2. For homogeneous Dirichlet boundary conditions on the domain X, we must use the Green's function $\bar{\mathcal{G}}$ that satisfies the same boundary conditions.

Solutions with general Dirichlet boundary conditions $g = u\big|_{\partial X}$ can be obtained by computing boundary integrals,

$$u(\mathbf{x}) = \int\limits_{\mathbf{y}\in X} \bar{\mathcal{G}}(\mathbf{x},\mathbf{y}) f(\mathbf{y}) + \int\limits_{\mathbf{y}\in \partial X} g(\mathbf{y}) \frac{\partial}{\partial \mathbf{n}_y} \bar{\mathcal{G}}(\mathbf{x},\mathbf{y}) \tag{4.2}$$

where, the first term is the solution with homogeneous Dirichlet boundary conditions and the second term solves the Laplace equation with the prescribed boundary conditions.

Variable Coefficient Problems. We now consider variable-coefficient Poisson problem of the form $\nabla\cdot a(\mathbf{x})\nabla u(\mathbf{x}) + b(\mathbf{x})\cdot\nabla u(\mathbf{x}) + c(\mathbf{x})u(\mathbf{x}) = f(\mathbf{x})$ with homogeneous Dirichlet boundary conditions on ∂X. The coefficients a, b and c are sufficiently smooth and $a(\mathbf{x}) \neq 0$ for all $\mathbf{x} \in X$. Furthermore, we assume that the coefficients are such that the elliptic PDE is well-posed. The integral equation formulation for this problem is given by writing the solution as $u = \bar{\mathcal{G}}[\phi]$, for an unknown density ϕ. The notation $\bar{\mathcal{G}}[\phi]$ denotes the convolution of $\bar{\mathcal{G}}$ with ϕ. After simplification we have the volume integral equation,

$$a\phi + (\nabla a + b)\cdot\nabla\bar{\mathcal{G}}[\phi] + c\,\bar{\mathcal{G}}[\phi] = f \tag{4.3}$$

Since $\bar{\mathcal{G}}[\cdot]$ and $\nabla\bar{\mathcal{G}}[\cdot]$ are weakly singular integral operators, the second and third terms are compact operators acting on ϕ. Therefore, Eq. (4.3) is a second-kind Fredholm integral equation. Since we assume that the original elliptic PDE is well-posed, therefore, for $f = 0$, it has only the trivial solution $u = 0$. Then, for $f = 0$, Eq. (4.3) has only the trivial solution $\phi = 0$ and by the Fredholm alternative the non-homogeneous equation has a unique solution. We solve the above integral equation for ϕ and compute the solution $u = \bar{\mathcal{G}}[\phi]$. Many variable coefficient elliptic PDEs can similarly be converted to second-kind integral equations. Such VIE formulations can also be constructed for variable viscosity Stokes problem and the Lippmann-Schwinger equation for acoustic and electro-magnetic scattering [24].

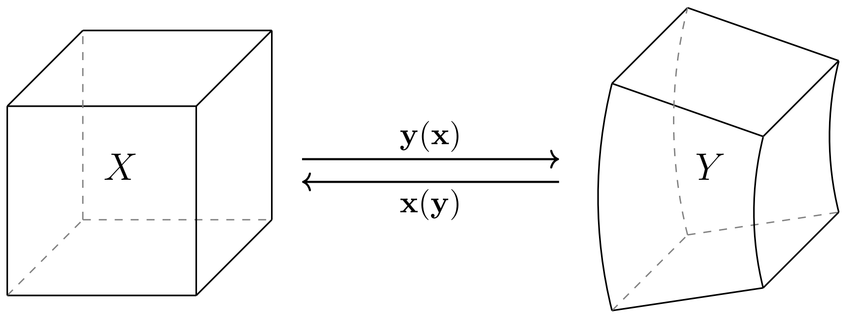


Figure 4.1 *We will consider fast algorithms for Stokes problems in domains Y that can be smoothly mapped to a unit cube X. That is, we will assume that there is a smooth diffeomorphism $\mathbf{y} : X \longrightarrow Y$ that maps points from X to Y and we will use it for a coordinate-transformation based scheme.*

4.2.2 Coordinate Transformations

We will now derive an integral equation formulation for the constant-coefficient Poisson problem on non-regular domains by mapping the problem to cubic domains. We consider a non-regular domain $Y \subset \mathbf{R}^3$ such that there exists a smooth diffeomorphism $\mathbf{y} : X \longrightarrow Y$ which maps points $\mathbf{x} \in X = (0,1)^3$ to points $\mathbf{y}(\mathbf{x}) \in Y$ as shown in Fig. 4.1. The inverse transformation $\mathbf{x} = \mathbf{y}^{-1}$, maps points $\mathbf{y} \in Y$ to points $\mathbf{x}(\mathbf{y}) \in X$. We define the Jacobian $J = (\partial y_i / \partial x_j)_{ij}$ and the metric tensor $A = J^{-1} J^{-T}$.

We consider the Poisson problem $\Delta u = f$ with homogeneous Dirichlet boundary conditions on ∂Y. We apply a change of coordinates and rewrite this problem on the unit cube X,

$$\begin{aligned} \nabla\cdot A \nabla (u \circ \mathbf{y}) - (\nabla\cdot J^{-1}) \cdot J^{-T} \nabla (u \circ \mathbf{y}) = f \circ \mathbf{y} \qquad & \text{in } X \\ u \circ \mathbf{y} = 0 \qquad & \text{on } \partial X. \end{aligned} \tag{4.4}$$

When $\det(J)$ is constant, then $\nabla\cdot J^{-1} = 0$ and the second term in Eq. (4.4) vanishes. We make this assumption only to simplify the analysis of our formulation for the Poisson equation in Section 4.2.3. In general, this constraint is not required and we do not assume this in our formulation for the Stokes equation.

The integral equation formulation is now given by substituting $u \circ \mathbf{y} = \bar{\mathcal{G}}[\phi]$,

$$\nabla\cdot A \nabla \bar{\mathcal{G}}[\phi] = f \circ \mathbf{y} \tag{4.5}$$

where, ϕ is an unknown density, $\bar{\mathcal{G}}$ is the Green's function for the Poisson problem on X and satisfies homogeneous Dirichlet boundary conditions. Formally, we can rewrite the above VIE as,

$$\lambda\phi + \nabla\cdot (A - \lambda I) \nabla \bar{\mathcal{G}}[\phi] = f.$$

When $\|A - \lambda I\|_{H^1}$ is small, the linear system above is well conditioned. However, in general, the second term in this equation may not be compact for any choice for λ. Therefore, we cannot apply the Fredholm alternative to show existence and uniqueness of the solution. Also, Eq. (4.5) requires computing second order derivatives of the potential $\bar{\mathcal{G}}[\phi]$. When seeking solutions in the space H^2, the derivatives must be understood in the weak sense. In the following section we will provide a weak formulation for Eq. (4.5).

4.2.3 Weak-Formulation for Poisson Problem

We now derive a weak formulation for the constant-coefficient Poisson problem under change of coordinates discussed above in Section 4.2.2 and we will show well-posedness of the formulation. Starting from Eq. (4.5) and assuming the solution $\phi \in L^2(X)$, it can be shown that $\nabla\cdot A\nabla\bar{\mathcal{G}}[\phi] \in L^2(X)$. For $f \in L^2(Y)$, the weak formulation is obtained by computing an L^2-inner product of Eq. (4.5) with test functions $v \in L^2(X)$,

$$\left(\nabla\cdot A\nabla\bar{\mathcal{G}}[\phi], v\right)_{L^2(X)} = (f \circ \mathbf{y}, v)_{L^2(X)} \qquad \forall v \in L^2(X). \tag{4.6}$$

We now provide some basic results which will be used to prove existence and uniqueness of the solution to Eq. (4.6) in Theorem 4.2.3.

Lemma 4.2.1. *Let $X = (0,1)^3$ and $Y \subset \mathbf{R}^3$ such that there exists a smooth diffeomorphism $\mathbf{y} : X \longrightarrow Y$ and $\det(J)$ is constant (where J is the Jacobian for the coordinate map $\mathbf{y}$). If $f \in L^2(Y)$ and u is a weak solution to the constant-coefficient Poisson problem $\Delta u = f$ with homogeneous Dirichlet boundary conditions on Y, then:*

(a) $u \in H^2(Y)$

(b) $\|u\|_{L^2(Y)} \leq c_1\|f\|_{L^2(Y)}$

(c) $\|D^2u\|_{L^2(Y)} \leq c_2\|f\|_{L^2(Y)}$

(d) $\|\nabla u\|_{L^2(Y)} \leq c_3\|f\|_{L^2(Y)}$

where, the constants c_1, c_2 and c_3 depend only on the coordinate map $\mathbf{y}$.

Proof. (a) Applying a change of coordinates, we have that $u \circ \mathbf{y} \in H^1(X)$ is the solution to the variable coefficient elliptic problem $\nabla\cdot A\nabla(u \circ \mathbf{y}) = f \circ \mathbf{y}$ with homogeneous Dirichlet boundary conditions on X. Since X is a convex domain, from theorem 3.2.1.2 in [54], $u \circ \mathbf{y} \in H^2(X)$; therefore $u \in H^2(Y)$.

(b) Since $u \in H^1(Y)$ is the unique weak solution to the Poisson problem with homogeneous Dirichlet boundary conditions, we can apply Theorem 6 in §6.2 of [34] to conclude $\|u\|_{L^2(Y)} \leq c_1\|f\|_{L^2(Y)}$.

(c) Let $X_e = (-1,2)^3$ and $f_e \in L^2(X_e)$ be the extension of $f \circ \mathbf{y} \in L^2(X)$ constructed by tiling negative mirror images on either side of X. Also, let $A_e \in C^0(X_e)$ be the extension of A constructed by tiling positive mirror images on either side of X. Then, the solution $u_e \in H^2(X_e)$ of the variable coefficient Poisson problem $\nabla\cdot A_e\nabla u_e = f_e$ with homogeneous Dirichlet boundary conditions on X_e, is an extension of $u \circ \mathbf{y} \in H^2(X)$. Since $X \subset\subset X_e$, by interior H^2-regularity (Theorem 8.8 in [41]), we have $\|u_e\|_{H^2(X)} \leq c' \left(\|u_e\|_{L^2(X_e)} + \|f_e\|_{L^2(X_e)}\right)$. Then, using (b) and the fact that u_e

and f_e are bounded linear extensions of $u \circ \mathbf{y}$ and $f \circ \mathbf{y}$ respectively, we have $\|u \circ \mathbf{y}\|_{H^2(X)} \leq c'' \|f \circ \mathbf{y}\|_{L^2(X)}$. Since $\mathbf{y}^*$ and $\mathbf{y}^{-1*}$ (the duals of $\mathbf{y}$ and $\mathbf{y}^{-1}$) are bounded linear isomorphisms between $H^2(X)$ and $H^2(Y)$, we have $\|D^2 u\|_{L^2(Y)} \leq \|u\|_{H^2(Y)} \leq c_2 \|f\|_{L^2(Y)}$.

(d) We note that Y has a Lipschitz boundary, $u \in H^2(Y)$ and $u\big|_{\partial Y} = 0$. Therefore, $\nabla u \in H^1(Y)$ and $\int_Y \nabla u = 0$. Using the Poincaré-Wirtinger inequality and (c) gives the required result, $\|\nabla u\|_{L^2(Y)} \leq c\|D^2 u\|_{L^2(Y)} \leq c_3 \|f\|_{L^2(Y)}$

□

Lemma 4.2.2. *Let $X = (0,1)^3$ and $Y \subset \mathbf{R}^3$ such that there exists a smooth diffeomorphism $\mathbf{y} : X \longrightarrow Y$. We define $A = J^{-1} J^{-T}$ where $J = (\partial y_i / \partial x_j)_{ij}$. We assume that $\det(J)$ is constant. Also, let $\bar{\mathcal{G}}$ and $\bar{\mathcal{G}}^Y$ be the Green's functions for the constant-coefficient Poisson problems in the domains X and Y respectively with homogeneous Dirichlet boundary conditions. Then,*

(a) $\|\nabla \cdot A \nabla \bar{\mathcal{G}}[\phi]\|_{L^2(X)} \leq c_x \|\phi\|_{L^2(X)} \qquad \forall \phi \in L^2(X)$

(b) $\|\nabla \cdot A^{-1} \nabla \bar{\mathcal{G}}^Y [f]\|_{L^2(Y)} \leq c_y \|f\|_{L^2(Y)} \qquad \forall f \in L^2(Y)$

(c) $\nabla \cdot A^{-1} \nabla \bar{\mathcal{G}}^Y \Big[\big(\nabla \cdot A \nabla \bar{\mathcal{G}}[\phi]\big) \circ \mathbf{y}^{-1} \Big] = \phi \circ \mathbf{y}^{-1} \qquad \forall \phi \in L^2(X)$

where, the constants c_x and c_y depend only on the coordinate map $\mathbf{y}$.

Proof. (a) We note that

$$\begin{aligned} \nabla \cdot A \nabla \bar{\mathcal{G}}[\phi] &= A : D^2 \bar{\mathcal{G}}[\phi] + (\nabla \cdot A) \cdot \nabla \bar{\mathcal{G}}[\phi] \\ \Longrightarrow \|\nabla \cdot A \nabla \bar{\mathcal{G}}[\phi]\|_{L^2(X)} &\leq \|A\|_{L^\infty} \|D^2 \bar{\mathcal{G}}[\phi]\|_{L^2(X)} + \|\nabla \cdot A\|_{L^\infty} \|\nabla \bar{\mathcal{G}}[\phi]\|_{L^2(X)}. \end{aligned}$$

Then, from Lemma 4.2.1 (c) & (d), we have the required result,

$$\|\nabla \cdot A \nabla \bar{\mathcal{G}}[\phi]\|_{L^2(X)} \leq (c_2 \|A\|_{L^\infty} + c_3 \|\nabla \cdot A\|_{L^\infty}) \|\phi\|_{L^2(X)}.$$

(b) Following the steps used for (a), we have the required result,

$$\|\nabla \cdot A^{-1} \nabla \bar{\mathcal{G}}^Y [f]\|_{L^2(Y)} \leq (c_2 \|A^{-1}\|_{L^\infty} + c_3 \|\nabla \cdot A^{-1}\|_{L^\infty}) \|f\|_{L^2(Y)}.$$

(c) Under change of coordinates, we have the following two relations,

$$(\Delta u) \circ \mathbf{y}^{-1} = \nabla \cdot A^{-1} \nabla (u \circ \mathbf{y}^{-1}) \qquad \text{in } Y \tag{4.7}$$

$$(\Delta v) \circ \mathbf{y} = \nabla \cdot A \, \nabla (v \circ \mathbf{y}) \qquad \text{in } X. \tag{4.8}$$

Let $v = u \circ \mathbf{y}^{-1}$. Then, from Eq. (4.8), we have

$$\begin{aligned} \Delta(u \circ \mathbf{y}^{-1}) = \quad & (\nabla \cdot A \nabla u) \circ \mathbf{y}^{-1} \\ \implies \quad u \circ \mathbf{y}^{-1} = \bar{\mathcal{G}}^{Y} & \left[(\nabla \cdot A \nabla u) \circ \mathbf{y}^{-1} \right]. \end{aligned} \tag{4.9}$$

Let $u = \bar{\mathcal{G}}[\phi]$. Then, from Eq. (4.7) and Eq. (4.9), we have the result (c)

$$\phi \circ \mathbf{y}^{-1} = \nabla \cdot A^{-1} \nabla \bar{\mathcal{G}}^{Y} \left[(\nabla \cdot A \nabla \bar{\mathcal{G}}[\phi]) \circ \mathbf{y}^{-1} \right] \qquad \forall \phi \in L^2(X).$$

□

Theorem 4.2.3. Existence and Uniqueness of Solution. *The weak formulation Eq. (4.6) has a unique solution $\phi \in L^2(X)$.*

Proof. Since the bilinear form in Eq. (4.6) is an indefinite operator, we use the generalized Lax-Milgram theorem. For the bilinear form $B(\phi, v) = \left(\nabla \cdot A \nabla \bar{\mathcal{G}}[\phi], v\right)_{L^2(X)}$, we need to show the following,

(a) $|B(\phi, v)| \leq M \|\phi\|_2 \|v\|_2 \qquad \forall v, \phi \in L^2(X)$

Use Hölder's inequality and Lemma 4.2.2 (a),

$$|B(\phi, v)| \;\leq\; \|\nabla \cdot A \nabla \bar{\mathcal{G}}[\phi]\|_2 \|v\|_2 \;\leq\; c_x \|\phi\|_2 \|v\|_2$$

(b) $\displaystyle \sup_{v \in L^2(X),\ \|v\|_2 \neq 0} \frac{|B(\phi, v)|}{\|v\|_2} \geq c\|\phi\|_2 \qquad \forall \phi \in L^2(X)$

Choose $v = \nabla \cdot A \nabla \bar{\mathcal{G}}[\phi]$, then, we need to show

$$\frac{|B(\phi, v)|}{\|v\|_2} = \|\nabla \cdot A \nabla \bar{\mathcal{G}}[\phi]\|_2 \geq c\|\phi\|_2$$

Substitute $f = (\nabla \cdot A \nabla \bar{\mathcal{G}}[\phi]) \circ \mathbf{y}^{-1}$ in Lemma 4.2.2 (b) and use Lemma 4.2.2 (c),

$$c_y \|(\nabla \cdot A \nabla \bar{\mathcal{G}}[\phi]) \circ \mathbf{y}^{-1}\|_{L^2(Y)} \;\geq\; \|\nabla \cdot A^{-1} \nabla \bar{\mathcal{G}}^{Y} [(\nabla \cdot A \nabla \bar{\mathcal{G}}[\phi]) \circ \mathbf{y}^{-1}]\|_{L^2(Y)} \;=\; \|\phi \circ \mathbf{y}^{-1}\|_{L^2(Y)}$$

Since $\mathbf{y}^*$ and $\mathbf{y}^{-1*}$ are bounded linear isomorphisms between $L^2(X)$ and $L^2(Y)$,

$$\|\nabla \cdot A \nabla \bar{\mathcal{G}}[\phi]\|_{L^2(X)} \;\geq\; c\|\phi\|_{L^2(X)}$$

(c) $\displaystyle \sup_{\phi \in L^2(X)} |B(\phi, v)| > 0 \qquad \forall v \in L^2(X), v \neq 0$

Choose $\phi = (\nabla \cdot A^{-1} \nabla \bar{\mathcal{G}}^{Y} [v \circ \mathbf{y}^{-1}]) \circ \mathbf{y}$ and use Lemma 4.2.2 (c),

$$B(\phi, v) \;=\; \left(\nabla \cdot A \nabla \bar{\mathcal{G}}[(\nabla \cdot A^{-1} \nabla \bar{\mathcal{G}}^{Y} [v \circ \mathbf{y}^{-1}]) \circ \mathbf{y}], v\right)_{L^2(X)} \;=\; \|v\|_{L^2(X)} > 0$$

□

General Boundary Conditions. For general Dirichlet boundary conditions $g_{\partial Y} = u\big|_{\partial Y}$, we map the boundary data to ∂X and construct an extension g_X on the entire domain X. One way to construct this extension is using boundary integrals:

$$g_X(\mathbf{x}) = \int\limits_{\mathbf{z}\in\partial X} g_{\partial Y}(\mathbf{y}(\mathbf{z}))\frac{\partial}{\partial n_z}\bar{\mathcal{G}}(\mathbf{x},\mathbf{z}) \tag{4.10}$$

We assume that the boundary data is sufficiently regular so that $g_X \in H^2(X)$. Then, we solve the following weak formulation for the density $\phi \in L^2(X)$,

$$\left(\nabla\cdot A\nabla\bar{\mathcal{G}}[\phi], v\right)_{L^2(X)} = (f \circ \mathbf{y} - \nabla\cdot A\nabla g_X, v)_{L^2(X)} \qquad \forall v \in L^2(X). \tag{4.11}$$

The solution to the BVP in Y is then given by $u = (\bar{\mathcal{G}}[\phi] + g_X) \circ \mathbf{y}$.

4.2.4 Formulation for Stokes Flow

We will now discuss incompressible Stokes flow problems. We will introduce some basic notation and the fundamental solutions for the constant coefficient Stokes equation. Then, we will derive a VIE formulation for Stokes flow under coordinate transformation.

Constant-Coefficient Problem. The constant-coefficient Stokes problem is given by the momentum equation, $\Delta u(\mathbf{x}) - \nabla p(\mathbf{x}) = f(\mathbf{x})$ and the incompressibility constraint, $\nabla\cdot u(\mathbf{x}) = 0$. The pressure $p(\mathbf{x})$ and velocity $u(\mathbf{x})$ are the unknowns to be determined. For free-space boundary conditions, the Green's function $\mathcal{S}$ for the velocity and $\mathcal{P}$ for the pressure are well known (see Table 4.2). $\mathcal{S}$ is also called the Stokeslet. The free-space solution is then given by $u = \mathcal{S}[f]$ and $p = \mathcal{P}[f]$.

Unlike the Poisson problem, for computing solutions for Stokes flow with Dirichlet boundary conditions on a unit cube, we cannot use the method of images. Adding an image of the Stokeslet cancels only the normal component of the velocity. Similarly, subtracting an image of the Stokeslet cancels only the tangential component of the velocity. To completely cancel the potential from a Stokeslet on a half-plane, requires its image, a Stokes doublet and a source doublet at the location of the image [14]. The method has also been extended to two parallel plates in [69]. Another method for half-space uses Papkovich-Neuber potentials [46]. However, we are not aware of any work which extends the method of images to more general geometries for the Stokes equation. Therefore, for Dirichlet boundary conditions on a cubic domain, we will use a boundary integral equation (BIE) formulation. This formulation will be discussed in Section 4.2.5. In the following discussion we assume that the velocity and pressure Green's functions, for the Stokes problem with homogeneous Dirichlet boundary conditions on the unit cube X, are implemented by the operators $\bar{\mathcal{S}}[\cdot]$ and $\bar{\mathcal{P}}[\cdot]$ respectively.

Coordinate Transformations. We now derive a formulation for Stokes flow in a non-regular domain $Y \subset \mathbf{R}^3$ with homogeneous Dirichlet boundary conditions by extending the discussion in Section 4.2.2. The variable viscosity Stokes equation is given by,

$$\nabla\cdot\mu\left(\nabla u + \nabla u^T\right) - \nabla p = f \qquad \text{in } Y \qquad \text{and} \tag{4.12a}$$

$$\nabla\cdot u = 0 \qquad \text{in } Y, \tag{4.12b}$$

where, the viscosity μ is a non-zero scalar variable; u and p are the unknown velocity and pressure. Let $X = (0,1)^3$ and $\mathbf{y} : X \longrightarrow Y$ be a smooth diffeomorphism (see Fig. 4.1). We apply a coordinate transform to map this problem to the cubic domain X and rewrite Eq. (4.12) as follows,

$$\nabla_{\mathbf{y}}\cdot(\mu\circ\mathbf{y})\left(\nabla_{\mathbf{y}} u\circ\mathbf{y} + (\nabla_{\mathbf{y}} u\circ\mathbf{y})^T\right) - \nabla_{\mathbf{y}} p\circ\mathbf{y} = f_X \qquad \text{in } X \qquad \text{and} \tag{4.13a}$$

$$\nabla_{\mathbf{y}}\cdot u\circ\mathbf{y} = 0 \qquad \text{in } X, \tag{4.13b}$$

where, $\nabla_{\mathbf{y}}\cdot\mathbf{v} = \operatorname{trace}\left((\nabla\mathbf{v})J^{-1}\right)$ and $\nabla_{\mathbf{y}} v = J^{-T}\nabla v$ denote the divergence and gradient with respect to $\mathbf{y}$ and $f_X = f\circ\mathbf{y}$. For the transformed problem, we represent the solution velocity field and pressure by u_X and p_X on X. We map u_X and p_X to the solution of our original problem in Y by the following transformation,

$$u\circ\mathbf{y} = J\det(J)^{-1}u_X \qquad \text{and} \qquad p\circ\mathbf{y} = \det(J)^{-1}p_X. \tag{4.14}$$

The mapping for the velocity field has been chosen in such a way that u is divergence-free *if-and-only-if* the velocity field u_X is divergence-free. In our integral equation formulation, this allows us to construct u_X in X by using a divergence-free fundamental solution $\bar{\mathcal{S}}$ and the mass conservation equation is satisfied implicitly for the solution u in Y. The mapping for pressure has been chosen in such a way that for scaling transformations (*i.e.* $J = sI$ for a constant scalar s), $\bar{\mathcal{S}}$ is the fundamental solution for the transformed problem in X with constant μ. Therefore, the convolution with $\bar{\mathcal{S}}$ directly solves the transformed problem.

The solution u_X and p_X are constructed by computing a convolution with an unknown density ϕ, so that $u_X = \bar{\mathcal{S}}[\phi]$ and $p_X = \bar{\mathcal{P}}[\phi]$. Substituting in Eq. (4.14) and rewriting the momentum conservation equation in Eq. (4.13) gives the following VIE formulation,

$$\nabla_{\mathbf{y}}\cdot(\mu\circ\mathbf{y})\left(\nabla_{\mathbf{y}} J\det(J)^{-1}\bar{\mathcal{S}}[\phi] + (\nabla_{\mathbf{y}} J\det(J)^{-1}\bar{\mathcal{S}}[\phi])^T\right) - \nabla_{\mathbf{y}}\det(J)^{-1}\bar{\mathcal{P}}[\phi] = f_X \quad \text{in } X. \tag{4.15}$$

As discussed before, the mass conservation equation is satisfied implicitly since $\nabla_{\mathbf{y}}\cdot J\det(J)^{-1}\bar{\mathcal{S}}[\phi] = 0$. We apply chain rule in Eq. (4.15) to move the derivatives to the kernel functions. We do this to avoid having to compute derivatives numerically. Then, the VIE can be written as follows,

$$c_0\phi + c_1\bar{\mathcal{S}}[\phi] + c_2\nabla\bar{\mathcal{S}}[\phi] + c_3\nabla\nabla\bar{\mathcal{S}}[\phi] + c_4\bar{\mathcal{P}}[\phi] + c_5\nabla\bar{\mathcal{P}}[\phi] = f_X \qquad \text{in } X, \tag{4.16}$$

where, $c_0, c_1, \cdots, c_5$ are the coefficients defined pointwise in X; $\nabla\bar{\mathcal{S}}[\cdot]$ and $\nabla\bar{\mathcal{P}}[\cdot]$ denote the convolution with the gradient of the Stokes single-layer velocity and pressure kernels, and $\nabla\nabla\bar{\mathcal{S}}[\cdot]$ denotes the convolution with the Hessian of the Stokes single-layer kernel. Once we have solved Eq. (4.16) for the density ϕ, the solution u_X in X is given by $u_X = \bar{\mathcal{S}}_X[\phi]$ and the solution u for the original problem in Y is given by $u = J\det(J)^{-1}u_X \circ \mathbf{x}$.

In the previous discussion, we have assumed that $\bar{\mathcal{S}}$ is the fundamental solution which satisfies homogeneous Dirichlet boundary conditions. Since for the Stokes equation, $\bar{\mathcal{S}}$ is not known analytically, we will use a boundary integral equation formulation to achieve the same effect. We discuss this in the following section.

4.2.5 Dirichlet Boundary Conditions on Cube

We now discuss boundary integral equation formulations for constructing solutions to homogeneous Poisson and Stokes equations with prescribed Dirichlet boundary conditions.

From Green's third identity, for a harmonic function u in X, we have the relation,

$$u(\mathbf{x}) = \int\limits_{\mathbf{y}\in\partial X} u(\mathbf{y})\frac{\partial\mathcal{G}(\mathbf{y},\mathbf{x})}{\partial\mathbf{n}} - \mathcal{G}(\mathbf{y},\mathbf{x})\frac{\partial u(\mathbf{y})}{\partial\mathbf{n}} \qquad \text{for all} \quad \mathbf{x}\in X\setminus\partial X \tag{4.17}$$

where, $\mathcal{G}$ is any fundamental solution (Green's function) of the Laplace equation in X and $\mathbf{n}$ is the outward unit normal vector at $\mathbf{y}$. For the Laplace equation and other scalar elliptic PDEs on cubic domains, it is possible to construct a Green's function such that $\bar{\mathcal{G}}(\mathbf{y},\mathbf{x}) = 0$ for all $\mathbf{y}\in\partial X$ and $\mathbf{x}\in X$. This can be done using the method of images. Then, the solution u with Dirichlet boundary conditions $g = u\big|_{\partial X}$ is given by the double-layer formulation,

$$u(\mathbf{x}) = \int\limits_{\mathbf{y}\in\partial X} g(\mathbf{y})\frac{\partial\bar{\mathcal{G}}(\mathbf{y},\mathbf{x})}{\partial\mathbf{n}} \qquad \text{for all} \quad \mathbf{x}\in X\setminus\partial X. \tag{4.18}$$

However, in the general case, when a boundary condition dependent Green's function is not known analytically or cannot be constructed directly (such as for Stokes), we try to use boundary integral equation formulations to construct such solutions. We discuss such formulations below.

Single-Layer Formulation. We try to constructed a solution to the homogeneous elliptic PDE with prescribed Dirichlet boundary conditions using the convolution of an unknown boundary density function σ with the single-layer kernel function $\mathcal{G}$. The potential is given by $u = \mathcal{G}_{\partial X}[\sigma]$, where, the notation $\mathcal{G}_{\partial X}[\cdot]$ denotes the convolution of the Green's function $\mathcal{G}$ with a density function on ∂X. The solution constructed in this way satisfies the homogeneous elliptic PDE everywhere in X. To determine the unknown density σ, we need to solve the boundary integral equation (BIE): $\mathcal{G}_{\partial X}[\sigma](\mathbf{x}) = g(\mathbf{x})$ for all $\mathbf{x} \in \partial X$. This formulation requires inverting a compact integral operator and therefore leads to ill-conditioned discretizations.

Double-Layer Formulation. Similarly, a double-layer formulation can be used to construct a solution $u = \mathcal{D}_{\partial X}[\sigma]$, using an unknown boundary density σ and the double-layer kernel function $\mathcal{D}$. The density σ must be determined by solving the BIE: $\frac{1}{2}\sigma + \mathcal{D}_{\partial X}[\sigma] = g$ on ∂X. For domains with smooth boundaries, this is a second-kind integral equation and it leads to discretizations with bounded condition numbers. Such discretizations can be solved efficiently using iterative solver.

For domains with non-smooth boundaries, such as cubic domains with edges and corners, discretizations of both the single-layer and double-layer formulations converge slowly with mesh refinement due to the presence of corner singularities. A common method for dealing with corner singularities is to use a graded mesh near corners and edges. While this is often an acceptable solution in 2D, it becomes very expensive in 3D. Another approach uses explicit representation of the corner solutions using non-integer powers [91]; however, there is no existing work in 3D. In the following discussion, we present a direct BIE formulation for such domains.

Direct Formulation. For domains with non-smooth boundaries, we directly use Eq. (4.17) to construct the solution. For the solutions that we are seeking, we expect to be able to accurately resolve both $u\big|_{\partial X}$ and $\partial u/\partial \mathbf{n}\big|_{\partial X}$ with standard Galerkin discretizations. In the RHS of Eq. (4.17), we replace u with the Dirichlet boundary data $g = u\big|_{\partial X}$ and replace $\partial u/\partial \mathbf{n}$ with an unknown density σ. Therefore, we will construct solutions of the form $u = \mathcal{D}_{\partial X}[g] - \mathcal{G}_{\partial X}[\sigma]$ in X. This gives the relation $\mathcal{G}_{\partial X}[\sigma] = -\frac{1}{2}g + \mathcal{D}_{\partial X}[g]$ on ∂X, which we solve to obtain σ. While this formulation resolves the issues with corners and edges, it requires inverting the single-layer convolution operator which leads to ill-conditioned discretizations with the condition number worsening with mesh refinement. We will discuss preconditioning strategies to try to alleviate this drawback. We will next discuss the direct formulation for the Stokes equation.

Direct Formulation for Incompressible Stokes Flow. For the homogeneous Stokes equation, Green's third identity can be written as,

$$u = \mathcal{D}_{\partial X}[u] - \mathcal{S}_{\partial X}[\mu(\nabla u + \nabla u^T) - pI] \qquad \text{in} \quad X, \tag{4.19}$$

where, $\mathcal{S}_{\partial X}[\cdot]$ denotes the convolution with the Stokeslet and $\mathcal{D}_{\partial X}[\cdot]$ denotes the convolution with the Stokes double-layer kernel function on the domain boundary ∂X. Replacing u with the Dirichlet boundary data $g = u\big|_{\partial X}$ and the Cauchy stress tensor $\mu(\nabla u+\nabla u^T)-pI$ with an unknown density σ in the RHS gives an expression for the velocity field,

$$u = \mathcal{D}_{\partial X}[g] - \mathcal{S}_{\partial X}[\sigma] \qquad \text{in} \quad X. \tag{4.20}$$

On the boundary of the domain, this gives the relation for the unknown density σ,

$$\mathcal{S}_{\partial X}[\sigma] = \mathcal{D}_{\partial X}[g] - \frac{1}{2}g \qquad \text{on} \quad \partial X. \tag{4.21}$$

Once the unknown density σ is determined, we can then compute the velocity field using Eq. (4.20). Similarly, pressure p can be computed by substituting $\mathcal{S}$ and $\mathcal{D}$ with the corresponding single-layer and double-layer pressure kernels $\mathcal{P}$ and $\mathcal{K}$ in Eq. (4.20).

Inhomogeneous Stokes Equation with Dirichlet Boundary Conditions. For the constant-coefficient Stokes equation with non-zero body force f in the momentum equation and Dirichlet boundary conditions $u\big|_{\partial X} = g$, we construct a solution of the form,

$$u = \mathcal{S}_X[f] + \mathcal{D}_{\partial X}[\sigma_d] - \mathcal{S}_{\partial X}[\sigma_s], \tag{4.22}$$

where, σ_s and σ_d are the single-layer and the double-layer densities on ∂X given by,

$$\sigma_d = g - \mathcal{S}_X[f]\big|_{\partial X} \qquad \text{and} \qquad \mathcal{S}_{\partial X}[\sigma_s] = \mathcal{D}_{\partial X}[\sigma_d] - \frac{1}{2}\sigma_d. \tag{4.23}$$

By setting $g = 0$, we can use this formulation to compute $\bar{\mathcal{S}}[f]$, the result of convolution with the Green's function $\bar{\mathcal{S}}$ which satisfies homogeneous Dirichlet boundary conditions.

4.2.6 Overall Formulation for Stokes Flow

We now summarize the overall formulation for Stokes flow in a non-regular domain Y with general Dirichlet boundary conditions.

Convolution Operators. We denote the convolution of a volume density ϕ with the Stokeslet $\mathcal{S}$ on the cubic domain X by the notation $\mathcal{S}_X[\phi]$. Similarly, we also define the volume convolution with other kernel functions ($\nabla\mathcal{S}$, $\nabla\nabla\mathcal{S}$, $\mathcal{P}$, $\nabla\mathcal{P}$). These convolutions are implemented using our volume FMM discussed in Chapter 2.

The convolution of a boundary density σ with $\mathcal{S}$ on the cubic domain is denoted by $\mathcal{S}_{\partial X}[\sigma]$. Similarly, we also define the boundary convolution with other kernel functions ($\nabla\mathcal{S}$, $\nabla\nabla\mathcal{S}$, $\mathcal{P}$, $\nabla\mathcal{P}$, $\nabla\mathcal{D}$, $\nabla\nabla\mathcal{D}$, $\mathcal{K}$, $\nabla\mathcal{K}$). These are also computed numerically using FMM similar to the volume convolutions.

We denote the velocity and pressure Green's functions for the constant coefficient Stokes equation with homogeneous Dirichlet (zero velocity) boundary conditions on the cubic domain by $\bar{\mathcal{S}}$ and $\bar{\mathcal{P}}$ respectively. The convolution with $\bar{\mathcal{S}}$ and $\bar{\mathcal{P}}$ cannot be computed directly since these kernel functions are not known analytically. Instead, we construct the result of the volume convolution with ϕ as $\bar{\mathcal{S}}_X[\phi] = \mathcal{S}_X[\phi] + \mathcal{D}_{\partial X}[\sigma_d] - \mathcal{S}_{\partial X}[\sigma_s]$, where, $\sigma_d = -\mathcal{S}_X[\phi]\big|_{\partial X}$ and σ_s is given by the solution of the BIE: $\mathcal{S}_{\partial X}[\sigma_s] = \mathcal{D}_{\partial X}[\sigma_d] - \frac{1}{2}\sigma_d$.

VIE Formulation for Stokes Flow with Dirichlet Boundary Conditions. We consider the incompressible Stokes equation: $\nabla\cdot\mu\left(\nabla u + \nabla u^T\right) - \nabla p = f$ and $\nabla\cdot u = 0$ in the domain Y with Dirichlet boundary conditions $u\big|_{\partial Y} = g$. We will now derive a VIE formulation resulting from the mapping of this problem to a cubic domain X (see Fig. 4.1).

As discussed in Section 4.2.4, for homogeneous Dirichlet boundary conditions, we reformulate this problem on the cubic domain X by mapping the velocity u and pressure p in Y to u_X and p_X respectively in X using the transformation in Eq. (4.14). We also map the body force f in Y to $f_X = f \circ \mathbf{y}$ in X. For the transformed problem in X, we have the Dirichlet boundary conditions $u_X\big|_{\partial X} = g_X := J^{-1}\det(J) g \circ \mathbf{y}$.

We construct the solution u_X and p_X to the transformed problem using an unknown volume density ϕ and unknown single-layer and double-layer boundary densities σ_s and σ_d as follows,

$$u_X = \bar{\mathcal{S}}_X[\phi] + \mathcal{D}_{\partial X}[\sigma_d] - \mathcal{S}_{\partial X}[\sigma_s], \tag{4.24a}$$

$$p_X = \bar{\mathcal{P}}_X[\phi] + \mathcal{K}_{\partial X}[\sigma_d] - \mathcal{P}_{\partial X}[\sigma_s]. \tag{4.24b}$$

Applying the transformation in Eq. (4.14), then substituting in Eq. (4.13) and simplifying gives a VIE for the unknown density ϕ as following,

$$c_0\phi + c_1\bar{\mathcal{S}}_X[\phi] + c_2\nabla\bar{\mathcal{S}}_X[\phi] + c_3\nabla\nabla\bar{\mathcal{S}}_X[\phi] + c_4\bar{\mathcal{P}}_X[\phi] + c_5\nabla\bar{\mathcal{P}}_X[\phi] = f_X - f_1, \tag{4.25}$$

where, the volume density f_1 is given by,

$$\begin{aligned} f_1 =& c_1\mathcal{D}_{\partial X}[\sigma_d] + c_2\nabla\mathcal{D}_{\partial X}[\sigma_d] + c_3\nabla\mathcal{D}_{\partial X}[\sigma_d] + c_4\mathcal{K}_{\partial X}[\sigma_d] + c_5\nabla\mathcal{K}_{\partial X}[\sigma_d] \\ & - \left(c_1\mathcal{S}_{\partial X}[\sigma_s] + c_2\nabla\mathcal{S}_{\partial X}[\sigma_s] + c_3\nabla\mathcal{S}_{\partial X}[\sigma_s] + c_4\mathcal{P}_{\partial X}[\sigma_s] + c_5\nabla\mathcal{P}_{\partial X}[\sigma_s]\right). \end{aligned} \tag{4.26}$$

Here, the coefficients $c_0, c_1, \cdots, c_5$ are same as those in Eq. (4.16).

In Eq. (4.24), the kernel function $\bar{\mathcal{S}}$ satisfies homogeneous Dirichlet boundary conditions. Therefore, for the given Dirichlet boundary conditions $u_X\big|_X = g_X$, we require that $u_1 = \mathcal{D}_{\partial X}[\sigma_d] - \mathcal{S}_{\partial X}[\sigma_s]$ should satisfy the Dirichlet boundary conditions $u_1\big|_{\partial X} = g_X$. From the direct formulation, discussed in the previous section, we have the double-layer boundary density $\sigma_d = g_X$. The single-layer boundary density σ_s is given by the solution of the BIE: $\mathcal{S}_{\partial X}[\sigma_s] = \mathcal{D}_{\partial X}[\sigma_d] - \frac{1}{2}\sigma_d$.

Summary of Steps. The steps for solving the VIE formulation above are as follows:

1. Set $\;f_X = f \circ \mathbf{y}\;$ and $\;g_X = J^{-1}\det(J) g \circ \mathbf{y}$.

2. Set $\sigma_d = g_X$ and compute σ_s by solving the BIE: $\mathcal{S}_{\partial X}[\sigma_s] = \mathcal{D}_{\partial X}[\sigma_d] - \frac{1}{2}\sigma_d$.

3. Compute f_1 by evaluating Eq. (4.26).

4. Solve the VIE Eq. (4.25) for the unknown density ϕ. The convolution operations in the VIE require kernel functions which satisfy homogeneous Dirichlet boundary conditions. Therefore, each evaluation of the LHS of the VIE requires solving a BIE.

5. Construct u_X by evaluating Eq. (4.24) and map it to Y using the relation $u = J\det(J)^{-1}u_X \circ \mathbf{x}$.

For the boundary and volume integral equations discussed in this section, we will use a discontinuous Galerkin formulation on adaptive meshes to discretize these problems. We implement the convolution operators using our volume FMM discussed in Chapter 2 and solve the discretized integral equations using GMRES. We discuss the details of our solver in the following section.

4.3 Numerical Methods

In this section, we will discuss the numerical algorithms for solving the integral equation formulations discussed in Section 4.2. We will first discuss a discontinuous Galerkin discretization scheme in Section 4.3.1. Then, we will discuss the construction of the convolution operators in Section 4.3.2. In Section 4.3.3, we will discuss the solver for the BIE formulation and in Section 4.3.4, we will discuss the VIE solver. We will restrict our discussion to the case for Stokes equation; however, the algorithms also apply to the Poisson equation.

4.3.1 Discretization

To discretize our boundary and volume integral equations, we will use a high-order discontinuous Galerkin scheme on adaptive meshes. We use an adaptive octree to partition the domain X into smaller subdomains such that $X = \cup_i \omega_i$ where, each subdomain ω_i is a leaf node in the tree. For a function f defined at each point in X, we discretize using a piecewise polynomial representation of the function. On each leaf node ω_l, we use an n^{th} order polynomial approximation,

$$f(x,y,z) \approx \sum_{i+j+k<n} \hat{f}_{ijkl} T_i(x) T_j(y) T_k(z) \qquad \text{for all} \quad (x,y,z) \in \omega_l, \tag{4.27}$$

where, $T_i(x)$ is the Chebyshev polynomial of degree i in x and $\hat{f}_{ijkl}$ are the coefficient in the Chebyshev approximation. The coefficients are computed using an orthogonal projection $\hat{f}_{ijkl} = (f, T_{ijk})_{\omega_l}$. We denote the projection by $\hat{f} = P_{N_v} f$, where $\hat{f}$ is the vector of N_v coefficients in the piecewise polynomial approximation. Once the coefficients are known, we can evaluate the piecewise polynomial representation at any point in the domain. We can also compute gradients by differentiating the polynomial representation. This is useful when computing the Jacobian J from the coordinate map $\mathbf{y}(\mathbf{x})$. We use this to discretize the coordinate map $\mathbf{y}(\mathbf{x})$, the body force density f, the coefficients in the VIE formulation, the unknown density function ϕ and the solution u.

We similarly construct discretizations for functions defined on the boundary ∂X. We use Chebyshev polynomials to approximate the function on the faces of the leaf nodes in the tree. For a function g on ∂X, we denote the projection to the space of piecewise polynomials by $\hat{g} = P_{N_b} g$, where $\hat{g}$ is the vector of N_b coefficients. We use this to discretize the Dirichlet boundary data g, the boundary density functions σ_s and σ_d and the boundary solution u.

When constructing a piecewise polynomial approximation of the boundary data or volume data, we specify an error tolerance ϵ_{tree} and the maximum depth L for the octree. We recursively refine leaf nodes in the octree until the truncation error in the Chebyshev approximation on each leaf node is smaller than ϵ_{tree} or we reach the maximum depth L.

Next, we discuss the construction of the convolution operators which take a piecewise polynomial discretization as the input density and return a piecewise polynomial discretization of the potential.

4.3.2 Convolution Operators

In Chapter 2, we discussed the implementation of our PVFMM library for evaluating volume potentials on cubic domains. For a given kernel function $\mathcal{G}$ (with free-space boundary conditions) and an input density function f defined at each point in X, our volume FMM algorithm projects f to the space of piecewise polynomials, computes the convolution with $\mathcal{G}$ and returns the result as piecewise polynomial approximation. In short, it implements the discretized convolution operation: $\hat{u} = P_{N_v}\mathcal{G}[P_{N_v} f]$.

We have similarly extended our volume FMM to also compute boundary integrals on the faces of the cubic domain. The details of the algorithm are exactly the same as discussed in Chapter 2; however, instead of computing integrals over the volume of the octree leaf nodes, we can also compute integrals over the faces of the leaf nodes. Just as for the volume FMM, the translation operators for the near-interaction are precomputed using special quadratures for singular and near singular integrals and the far-field interactions are computed using the kernel independent FMM [109]. Therefore, our FMM algorithm implements the discretized convolution operation over boundary data. The result of the convolution can be evaluated either on the boundary of the domain $\hat{u} = P_{N_b}\mathcal{G}[P_{N_b}\sigma]$ or evaluated on the entire domain $\hat{u} = P_{N_v}\mathcal{G}[P_{N_b}\sigma]$.

For Stokes flow, we have implemented convolution operators for Stokes single-layer velocity $(\mathcal{S}[\cdot])$ and pressure $(\mathcal{P}[\cdot])$ kernels, their gradient $(\nabla\mathcal{S}[\cdot]$ and $\nabla\mathcal{P}[\cdot])$ and the Hessian $(\nabla\nabla\mathcal{S}[\cdot])$. In addition, for the boundary convolutions, we also implement the Stokes double-layer velocity $(\mathcal{D}[\cdot])$ and pressure $(\mathcal{K}[\cdot])$ kernels, their gradient $(\nabla\mathcal{D}[\cdot]$ and $\nabla\mathcal{K}[\cdot])$ and the Hessian $(\nabla\nabla\mathcal{D}[\cdot])$. We use these convolution operators to discretize our boundary and volume integral equations. We will next discuss the solvers for our BIE and VIE formulations.

4.3.3 Boundary Integral Equation Solver

We now discuss our solver for the BIE formulation. For the homogeneous Stokes equation with Dirichlet boundary conditions, we use the direct formulation discussed in Section 4.2.5. We discretize Eq. (4.21) by projecting g to the space of piecewise polynomials and replacing the convolution operators by their discrete versions,

$$P_{N_b}\mathcal{S}_{\partial X}[\hat{\sigma}] = P_{N_b}\mathcal{D}_{\partial X}[P_{N_b}g] - \frac{1}{2}P_{N_b}g. \tag{4.28}$$

We solve this BIE using GMRES to obtain the discretized unknown density $\hat{\sigma}$. Then, we compute the velocity field in X by evaluating the discretized form of Eq. (4.20) as follows,

$$\hat{u} = P_{N_v}\mathcal{D}_{\partial X}[P_{N_b}g] - P_{N_v}\mathcal{S}_{\partial X}[\hat{\sigma}]. \tag{4.29}$$

We can similarly evaluate the pressure and gradients of the pressure and the velocity fields by substituting the corresponding kernel functions in Eq. (4.29).

In Table 4.3, we show convergence results for our method for a Stokes flow problem with Dirichlet boundary conditions on a unit cube. The boundary data for this example is generated by placing a single Stokeslet outside of the domain X. For comparison, we also show results for the single-layer and double-layer formulations using the same mesh refinement. We observe that due to corner singularities in the solution, the single-layer and double-layer schemes converge slowly; achieving only 3-digits of accuracy. However, our scheme based on direct formulation converges to nearly 8-digits. Since the single-layer convolution operator is ill-conditioned, both the single-layer formulation and the direct formulation require excessively large number of GMRES iterations N_σ. For $\epsilon_{\text{GMRES}} = 1\text{E-}9$, we require over 200 GMRES iterations. On the other hand, the double-layer formulation being a second-kind integral equation is well-conditioned and converges in just 29 iterations. Consequently, the double-layer formulation is significantly faster than the single-layer and the direct formulations. Comparing the solve time for different discretization orders, we observe that for the same accuracy, using high-order discretization requires significantly fewer leaf octant N_{oct}. Therefore, the higher order scheme is up to $2\times$ faster.

Preconditioner. To improve the convergence rate of the GMRES solve, we have implemented a block-diagonal preconditioner. We precompute the inverse of the single-layer convolution operator matrix for a single leaf node. Since we use the free-space Green's function (Stokeslet), the convolution operator is translation invariant and also scale invariant (up to a constant scaling factor). Therefore, we compute just one preconditioner matrix block each for the faces, the edges and the corners of the cubic domain. In Fig. 4.2, we plot the GMRES residual as a function of the number of GMRES iterations for the unpreconditioned (Direct) and the preconditioned (Direct + Precond) schemes. For comparison, we also show results for the double-layer formulation (DL). We observe significant improvement in the convergence rate using the block-diagonal preconditioner; with the

					Double-Layer			Single-Layer			Direct		
ϵ_{tree}	ϵ_{GMRES}	m	q	N_{oct}	N_σ	L^∞	T_{solve}	N_σ	L^∞	T_{solve}	N_σ	L^∞	T_{solve}
1E-0	4E-3	4	8	1	6	1.6E-2	0.1	12	9.8E-3	0.1	22	1.0E-2	0.2
1E-1	4E-5	8	8	8	14	2.5E-3	0.2	39	2.6E-3	0.2	54	1.1E-3	0.8
1E-2	4E-6	10	8	50	17	1.4E-3	1.7	66	2.0E-3	5.3	83	2.7E-5	6.9
1E-3	1E-7	10	8	99	23	1.2E-3	5.0	123	2.3E-3	23.8	139	1.8E-6	28.4
1E-4	5E-8	12	8	323	24	9.5E-4	9.9	132	2.0E-3	51.9	158	1.7E-7	60.4
1E-5	1E-9	14	8	820	29	1.1E-3	37.5	182	1.7E-3	233.4	203	1.4E-8	248.2
1E-0	4E-3	4	10	1	6	8.5E-3	0.1	12	7.1E-3	0.1	22	5.2E-3	0.1
1E-1	4E-5	8	10	8	14	1.7E-3	0.2	42	2.2E-3	0.2	60	3.3E-4	1.2
1E-2	4E-6	10	10	22	17	1.6E-3	1.0	73	1.9E-3	4.0	96	3.4E-5	5.5
1E-3	1E-7	10	10	64	23	1.2E-3	4.3	118	1.8E-3	21.1	142	5.8E-7	25.8
1E-4	5E-8	12	10	99	24	9.6E-4	5.0	150	2.0E-3	31.1	166	1.2E-7	31.1
1E-5	1E-9	14	10	253	29	7.5E-4	19.2	193	1.6E-3	131.1	204	2.1E-8	136.9

Table 4.3 *Convergence results for Stokes equation with Dirichlet boundary conditions on a cubic domain using different BIE formulations. We report relative L^∞ norm of the error and show convergence as we decrease the tolerance for mesh refinement ϵ_{tree}, increase the order of multipole expansion m and decrease the GMRES tolerance ϵ_{GMRES}. We show results for different spatial discretization orders q. We also report the number of leaf nodes in the octree N_{oct}, the number of GMRES iterations N_σ and the total solve time T_{solve}.*

convergence rate being comparable to that of the double-layer formulation. However, we also notice that the convergence rate deteriorate slightly as we increase mesh refinement. This is because, with mesh refinement, the diagonal blocks act on a smaller region of the domain.

Stokes Equation with Non-Zero Body Force. We now consider the constant-coefficient Stokes equation in X with non-zero body force f in the momentum equation and Dirichlet boundary conditions $u\big|_{\partial X} = g$. We compute the solution using the formulation in Eqs. (4.22, 4.23). The discrete solution is computed by first constructing piecewise polynomial approximations $\hat{f} = P_{N_v} f$ and $\hat{g} = P_{N_b} g$. Then, we compute the discrete double-layer density,

$$\hat{\sigma}_d = \hat{g} - P_{N_b}\mathcal{S}_X[\hat{f}]. \tag{4.30}$$

To compute the single-layer density, we solve the following discretized BIE using GMRES,

$$P_{N_b}\mathcal{S}_{\partial X}[\hat{\sigma}_s] = P_{N_b}\mathcal{D}_{\partial X}[\hat{\sigma}_d] - \frac{1}{2}\hat{\sigma}_d. \tag{4.31}$$

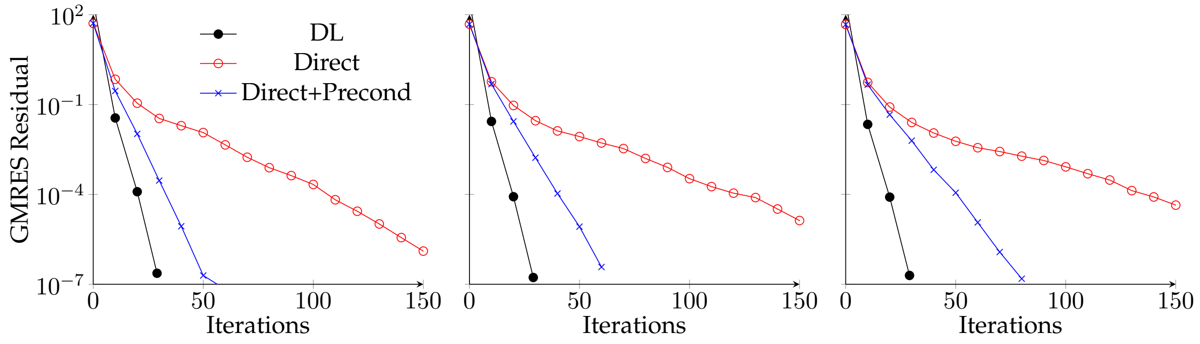


Figure 4.2 *We plot the GMRES residual as a function of the number of GMRES iterations. In each plot, we show results for the double-layer formulation (DL), the direct formulation (Direct) and the direct formulation with block-diagonal preconditioner (Direct+Precond). From left to right, the plots show results for 2, 3 and 4-levels for mesh refinement.*

Then, we compute the discretized solution $\hat{u}$ as follows,

$$\hat{u} = P_{N_v}\mathcal{S}_X[\hat{f}] + P_{N_v}\mathcal{D}_{\partial X}[\hat{\sigma}_d] - P_{N_v}\mathcal{S}_{\partial X}[\hat{\sigma}_s]. \tag{4.32}$$

On cubic domains (and other domains with non-smooth boundaries), the result of $\mathcal{S}_X[f]$ has unbounded second derivatives at the edges and corners of the domain. This affects the convergence rate of our scheme. To resolve this issue, we construct a C^0 extension f_e of f by using reflections of f from each face of the cubic domain. This scheme requires extra work to compute convolutions with the extended density; however, this cost is not significant.

					No-Extension			Extension		
q	m	ϵ_{tree}	ϵ_{GMRES}	N_{oct}	N_σ	L^∞	T_{solve}	N_σ	L^∞	T_{solve}
12	6	1E-01	1E-03	160	21	9.5E-02	2.9	20	9.5E-02	3.2
12	8	1E-02	1E-05	160	33	5.4E-03	4.9	30	5.4E-03	4.2
12	10	1E-04	1E-07	160	47	1.2E-04	10.1	45	1.2E-04	9.7
12	12	1E-05	1E-08	216	55	1.6E-05	20.9	52	1.6E-05	18.9
12	14	1E-06	1E-09	216	61	7.8E-07	39.6	59	2.4E-09	36.1
12	14	1E-08	1E-11	986	85	2.2E-07	121.2	84	2.5E-10	122.9

Table 4.4 *Convergence results for Stokes equation with $f \neq 0$ and Dirichlet boundary conditions. We compare results with and without extension of the volume force density f. We show convergence in the relative L^∞ norm of the error as we increase the multipole order m, decrease tolerance ϵ_{tree} for mesh refinement and decrease tolerance ϵ_{GMRES} for the GMRES solver. For both schemes, we also report the number of GMRES iterations N_σ and the total solve time on 16-CPU cores.*

In Table 4.4, we report convergence results for an analytical test case for Stokes flow with $f \neq 0$ and Dirichlet boundary conditions on a cubic domain. We observe that the scheme without extension for volume force density f converges to only 7-digits, while, the scheme using the extended density converges to nearly 10-digits. The total solve time T_{solve} for both schemes is similar and the extra cost associated with convolution over the extended density is not noticeable.

4.3.4 Volume Integral Equation Solver

We now discuss our solver for the Stokes equation with non-zero body force and general Dirichlet boundary conditions on a non-regular domain Y. The integral equation formulation for the problem reformulated on the cubic domain X is given in Section 4.2.6. We discretize and solve this formulation using the steps described below.

Let f_X be the volume force density and g_X be the Dirichlet boundary data mapped to the cubic domain as defined in Section 4.2.6. We construct piecewise polynomial approximations $\hat{f}_X = P_{N_v} f_X$ and $\hat{g}_X = P_{N_b} g_X$. We evaluate the coefficients $c_0, c_1, \cdots, c_5$ in the VIE (Eq. (4.25)) at the Chebyshev node points in each leaf node of the octree. We set $\hat{\sigma}_d = \hat{g}_X$ and solve Eq. (4.31) for the unknown single-layer boundary density $\hat{\sigma}_s$. Next, we evaluate $\hat{f}_1$ as follows,

$$\begin{aligned}\hat{f}_1 =& P_{N_v} c_1 \mathcal{D}_{\partial X}[\hat{\sigma}_d] + P_{N_v} c_2 \nabla \mathcal{D}_{\partial X}[\hat{\sigma}_d] + P_{N_v} c_3 \nabla \mathcal{D}_{\partial X}[\hat{\sigma}_d] + P_{N_v} c_4 \mathcal{K}_{\partial X}[\hat{\sigma}_d] + P_{N_v} c_5 \nabla \mathcal{K}_{\partial X}[\hat{\sigma}_d] \\ &- \left(P_{N_v} c_1 \mathcal{S}_{\partial X}[\hat{\sigma}_s] + P_{N_v} c_2 \nabla \mathcal{S}_{\partial X}[\hat{\sigma}_s] + P_{N_v} c_3 \nabla \mathcal{S}_{\partial X}[\hat{\sigma}_s] + P_{N_v} c_4 \mathcal{P}_{\partial X}[\hat{\sigma}_s] + P_{N_v} c_5 \nabla \mathcal{P}_{\partial X}[\hat{\sigma}_s]\right).\end{aligned} \tag{4.33}$$

To compute $P_{N_v} c_1 \mathcal{D}_{\partial X}[\hat{\sigma}_d]$, we first compute the convolution using our volume FMM and evaluate the result at the Chebyshev node points in each leaf node. We compute the product with the coefficient c_1 at each node point and then compute the orthogonal projection to the space of piecewise polynomials. We similarly compute the remaining terms.

Then, we solve the following discretized VIE using GMRES for the unknown density function $\hat{\phi}$,

$$\begin{aligned} c_0 \hat{\phi} + P_{N_v} c_1 \bar{\mathcal{S}}_X[\hat{\phi}] + P_{N_v} c_2 \nabla \bar{\mathcal{S}}_X[\hat{\phi}] + P_{N_v} c_3 \nabla\nabla \bar{\mathcal{S}}_X[\hat{\phi}] \\ + P_{N_v} c_4 \bar{\mathcal{P}}_X[\hat{\phi}] + P_{N_v} c_5 \nabla \bar{\mathcal{P}}_X[\hat{\phi}] = \hat{f}_X - \hat{f}_1. \end{aligned} \tag{4.34}$$

To compute $c_0\hat{\phi}$, we evaluate $\hat{\phi}$ at the Chebyshev node points on each leaf node in the tree, compute the pointwise product with c_0 and then compute a projection to the space of piecewise polynomials. The remaining terms are computed as discussed before for Eq. (4.33), except that instead of the convolution over boundary data, now the convolution is over the volume density ϕ. Also, notice that the convolution operations are not with the free-space kernel function $\mathcal{S}$, but instead with $\bar{\mathcal{S}}$ which has zero Dirichlet boundary conditions. We implement convolution with $\bar{\mathcal{S}}$ by following the steps in Eqs. (4.30–4.32) for zero boundary data. This requires solving a BIE in each evaluation of the LHS of Eq. (4.34).

Finally, once we have computed the unknown volume density $\hat{\phi}$ and the unknown boundary densities $\hat{\sigma}_s$ and $\hat{\sigma}_d$, we evaluate $\hat{u}_X$ as follows,

$$\hat{u}_X = P_{N_v}\bar{\mathcal{S}}_X[\hat{\phi}] + P_{N_v}\mathcal{D}_{\partial X}[\hat{\sigma}_d] - P_{N_v}\mathcal{S}_{\partial X}[\hat{\sigma}_s], \tag{4.35}$$

and construct the solution $\hat{u}$ using the mapping $u = J\det(J)^{-1}u_X \circ \mathbf{x}$.

Preconditioner. In many cases, we observed slow convergence of the GMRES solver for Eq. (4.34). Therefore, we have tried to precondition the solver using two different schemes described below.

- *Diagonal Preconditioner.* We divide Eq. (4.34) throughout by c_0.
- *Block-Diagonal Preconditioner.* We precondition using the inverse of the diagonal blocks (for each leaf node) in Eq. (4.34).

The block-diagonal preconditioner requires us to precompute and store the diagonal blocks for each leaf node in the tree which adds significant computational and memory overheads compared to the diagonal preconditioner. We found that both preconditioners had similar effect on the convergence rate of the GMRES solve; therefore, we choose to use the diagonal preconditioner since it is less expensive to apply.

4.4 Numerical Results

In Section 4.3.3, we presented convergence results for Stokes flow on cubic domains with Dirichlet boundary conditions. In this section, we will show results for the VIE solver on complex geometries with Dirichlet boundary conditions. All results presented in this chapter are on a single node with dual eight-core Intel Xeon E5-2687W CPUs running at 3.1GHz and 64GB of memory.

In Table 4.5, we show convergence results for our VIE solver for two different geometries shown in Fig. 4.3. We solve the incompressible Stokes equation with constant viscosity ($\mu\Delta u - \nabla p = f$ and $\nabla\cdot u = 0$ in Y with viscosity $\mu = 1$) and Dirichlet boundary conditions ($u\big|_{\partial Y} = g$). The velocity fields are generated analytically. The boundary condition is determined by evaluating the velocity field on the boundary of the domain ∂Y. The body force is determined by evaluating $f = \Delta u$. We show convergence in the relative L^∞ norm of the error as we increase m and decrease ϵ_{GMRES} and ϵ_{tree}. For both geometries,

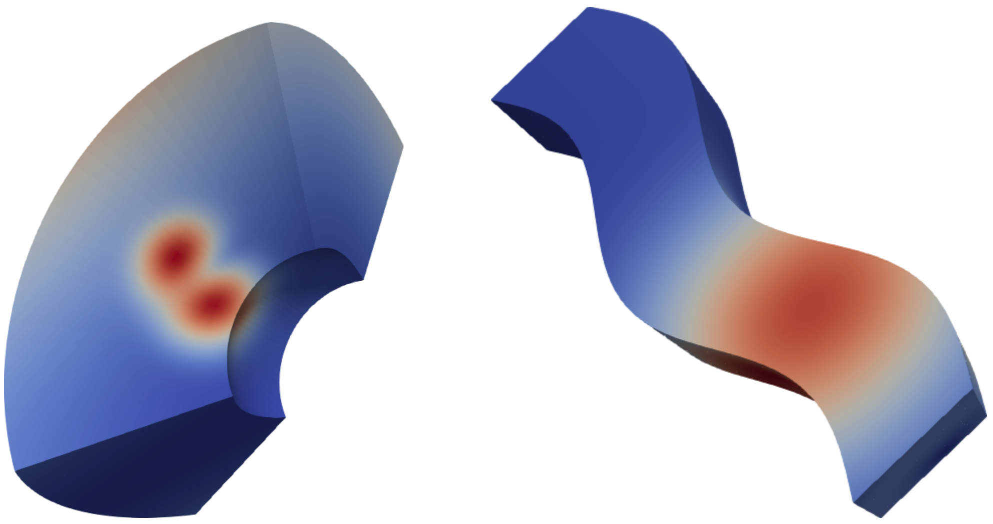


Figure 4.3 *Complex geometry shapes used for convergence results in Table 4.5. The colors show the magnitude of the analytical velocity field.*

				Geometry1					Geometry2				
q	m	ϵ_{tree}	ϵ_{GMRES}	N_{oct}	N_ϕ	$\overline{N}_\sigma$	L^∞	T_{solve}	N_{oct}	N_ϕ	$\overline{N}_\sigma$	L^∞	T_{solve}
10	6	2E-2	2E-2	8	7	14	3.4E-03	4.2	8	18	14	2.5E-2	7.9
10	6	7E-4	7E-4	8	13	22	1.2E-03	7.9	8	30	22	3.7E-3	17.0
10	8	2E-5	2E-5	216	19	41	1.5E-05	96.2	216	51	40	4.6E-5	243.3
10	10	8E-7	8E-7	216	26	53	8.8E-07	276.6	216	75	52	1.5E-6	734.8
10	12	3E-8	3E-8	2541	34	84	6.9E-09	4130.9	2506	97	83	4.3E-8	10863.0
10	14	1E-9	1E-9	2723	41	97	3.7E-10	9336.4	2744	123	97	1.4E-9	27032.3

Table 4.5 *Convergence results for constant viscosity Stokes flow with Dirichlet boundary conditions on Geometry1 (Fig. 4.3: left) and Geometry2 (Fig. 4.3: right). We show convergence in the relative L^∞ norm of the error as we increase the multipole order m, decrease tolerance ϵ_{tree} for mesh refinement and decrease tolerance ϵ_{GMRES} for the GMRES solver. For both shapes, we also report the number of GMRES iterations N_ϕ for the VIE solve, the average number of GMRES iterations $\overline{N}_\sigma$ for the boundary solves and the total solve time on 16-CPU cores.*

we show convergence to 9-digits of accuracy. The number of GMRES iterations N_ϕ for the VIE solve are not excessively large even for the highly elongated Geometry2 (Fig. 4.3). We also report the total time to solution T_{solve}. Of this, most of the time ($70\% - 89\%$) is spent in the BIE solver since we have to solve the BIE N_ϕ times, each requiring $\overline{N}_\sigma$ iterations on average.

We show similar results in Fig. 4.4 and Table 4.6 for Stokes flow in a curved duct with prescribed inlet and outlet velocity fields. Since we do not know the analytical velocity field, we compute errors by comparing with a reference solution. We observe convergence

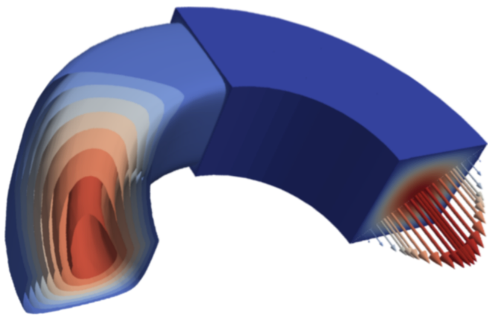


Figure 4.4 *Stokes flow in a curved duct with square cross section for given inlet and outlet velocity fields. We also visualize the isosurfaces with the same velocity magnitude.*

q	m	ϵ_{GMRES}	N_{oct}	N_ϕ	$\overline{N}_\sigma$	L^∞	T_{solve}
10	6	2E-2	8	35	14	5.7E-2	14
10	6	7E-4	8	47	21	1.1E-2	26
10	8	2E-5	64	67	36	2.5E-3	158
10	10	8E-7	64	86	45	5.4E-4	433
10	10	8E-7	456	88	55	5.3E-5	1574
10	12	3E-8	2528	110	84	5.8E-7	15299

Table 4.6 *For the flow in Fig. 4.4, we show convergence in the relative L^∞ norm as we increase the multipole order m, reduce GMRES tolerance ϵ_{GMRES} and increase mesh refinement. The error is computed by comparing with a reference solution computed using $m = 14$, $\epsilon_{\text{GMRES}} = 1\text{E-}9$ and 4-levels of mesh refinement.*

to about 6-digits for $m = 12$, $\epsilon_{\text{GMRES}} = 3\text{E-}8$ and 4-levels of mesh refinement.

4.5 Conclusions

We have presented new VIE formulations for Poisson and Stokes equations. These formulations map constant and variable coefficient elliptic PDEs in certain complex geometries to cubic domains. This allows us to use our volume fast multipole method, presented in Chapter 2, to compute volume integrals on the cubic domain and boundary integrals on the boundary of the cube. The discretized integral equation is then solved using GMRES.

We have also discussed the direct boundary integral equation formulation for Dirichlet boundary conditions on a cubic domain. Standard double-layer formulations converge slowly and require excessive refinement due to the presence of edges and corners in the domain. We have showed that our formulation does not have this drawback. Furthermore, we have presented a block-diagonal preconditioner to accelerate solution of the BIE using GMRES.

We have showed convergence results for constant viscosity Stokes flow in complex geometries using our VIE formulation. Our formulation works well and the number of GMRES iterations remain small even for highly elongated geometries.

While we have presented results only for Stokes flow with constant viscosity, our formulation also applies to the case with variable viscosity. In addition, more sophisticated preconditioners can be constructed for both boundary integral equations and volume integral equations. In particular, for the boundary integral solver, we plan to develop multigrid preconditioners.

5 Parallel Simulation of Concentrated Vesicle Suspensions

In this chapter, we describe a parallel boundary integral method for the numerical simulation of highly concentrated vesicle suspensions in a Stokesian fluid. We develop efficient parallel algorithms for singular and near-singular integration and accelerate far-field computation using our parallel FMM discussed in Chapter 2. We also discuss an adaptive time-stepping scheme and algorithms for collision handling, mesh reparameterization, area-volume correction.

5.1 Introduction

Vesicles are fluid filled membranes. The membrane is phospholipid bilayer with the hydrophobic tail of the molecules pointing towards the membrane and the hydrophilic head facing away from the membrane. The dynamics of vesicle flows are used to study the rheology of blood and other complex biofluids.

The method described here is an extension of previous work of [105, 88]. A boundary integral method for simulation of vesicle flows in three-dimensions was developed in [105]. In [88], this was extended to allow viscosity contrast between the fluid inside and outside of the vesicles.

We extend this method to allow long-timescale simulations of concentrated vesicle suspensions in parallel. The simulation of high volume fraction vesicle suspensions, which are representative of real biological systems (such as blood with $35\% \sim 50\%$ volume fraction for RBCs), presents several challenges. It requires computing accurate vesicle-vesicle interactions at length scales where standard quadratures are too expensive. The inter-vesicle separation can become arbitrarily small leading to vesicle collisions. Numerical errors can accumulate over time, making long-timescale simulations inaccurate.

Overcoming these challenges requires a scalable boundary integration framework which can efficiently handle interactions between vesicles at different length scales (singular, near-singular and far-field interactions). The complex dynamics of vesicle flows require time-adaptivity and algorithms to detect and handle collisions between vesicles.

5.1.1 Contributions

Our main contributions are summarized as follows:

- We present a singular integration scheme which has $\mathcal{O}\left(p^5\right)$ setup cost and $\mathcal{O}\left(p^4\right)$ cost for each subsequent evaluation for p^{th} order discretization. With the original $\mathcal{O}\left(p^5\right)$ evaluation scheme, singular integration accounted for over 90% of the total runtime; however, with the new scheme, the setup and the singular integration stages together account for less than 20% of the total time.

- We present a new near-singular integration scheme based on the method of [110]. This allows us to efficiently simulate high volume-fraction vesicle flows without the need for expensive upsampling required in the original scheme. The new scheme is implemented in parallel and has good scalability.

- We accelerate the computation of far-field interactions using our PVFMM library discussed in Chapter 2. The library is highly optimized using AVX vectorization and uses MPI for distributed memory parallelism. In addition, the library supports periodic boundary conditions and this allows us to simulate vesicles in periodic flows.

- We present an inexpensive method for estimating the error in each of the singular, near-singular and far-field integration schemes. We use this to adaptively adjust the order of the quadrature scheme and achieve the desired accuracy using the least amount of work.

- We introduce an algorithm for detecting collisions between vesicles. Such collision happen due to the discretization errors in our numerical scheme and ideally should not happen. These collisions can break simulations and make it impossible to simulate flows with high vesicle concentrations. To avoid vesicle collisions, we have introduced a short range repulsion term in our model and developed an efficient algorithm for evaluating this repulsion force.

- In long-timescale simulations, errors can gradually change the surface area and volume of the vesicles. We need correct for this drift by adjusting the area and volume of the vesicles in each time-step. We have developed an efficient algorithm to do this.

- The absence of in-plane shear resistance in vesicles necessitates a reparameterization scheme. The basic algorithm was presented in [105]. In this chapter we introduce some modifications to this algorithm and also analyze the scheme for different parameter values. This has significantly improved the quality for our surface meshes.
- We implement an adaptive time-stepping scheme which is based on the work of [85, 84]. We also present a new variation of this scheme which uses a more robust error estimate. This significantly reduces the solve time over a uniform time-stepping scheme.
- We present numerical results to show convergence of our method and study the dependence of the solution error on different parameter values when compared to a reference solution.
- We visualize long-timescale simulations for periodic Taylor-Green vortex flow and sedimentation of polydisperse vesicle suspensions with thousands of vesicles. We present scalability results for these simulations up to several thousands CPU cores on the Stampede system at Texas Advanced Computing Center.

5.1.2 Limitations

We restrict our attention to suspensions of vesicles in unbounded or periodic domains. We have ignored inertial terms, so the overall method is restricted to low Reynolds numbers. Only vesicles with spherical topology are considered and topological changes are not allowed. For general topologies one could, for example, use the boundary representation and singular integral quadrature introduced in [110]. We do not have any in-plane shear resistance in our formulation and this is a reasonable assumption for vesicles. However, for red blood cells (RBCs) and other cells, the shear resistance cannot be ignored.

The spatial discretization order is fixed and while we select the quadrature order for boundary integration adaptively, it is the same for all vesicles. This has the advantage that the algorithm can be applied to a large number of vesicles at once, resulting in better data locality and higher performance. However, for polydisperse simulations, each vesicle may require a different discretization order depending on the size and properties of the vesicle. In the current method, we are forced to use the highest discretization order and quadrature order for all vesicles.

We use a first-order adaptive time-stepping scheme. A spectral deferred correction (SDC) method, presented in [85, 84] for 2D vesicles, is well suited for high-order, adaptive time-stepping and leverages the Globally-Implicit first-order scheme presented here.

The number of GMRES iterations in each time step is relatively large for high volume fraction flows, particularly with periodic boundary conditions. Our current scheme uses an analytical preconditioner constructed for a sphere [105]. We believe that the inverse of the block diagonal part of the linear system would be a more effective preconditioner.

5.1.3 Related Work

A detailed review of related work on three-dimensional simulation of vesicle flows and flows with viscosity contrast can be found in [105] and [88] respectively. Work on simulating flow of concentrated vesicle suspensions includes [71, 72, 114, 115, 113, 83].

An $\mathcal{O}\left(p^4 \log p\right)$ scheme for singular integration is based on fast rotation of spherical harmonic expansions is discussed in [45]. The near-singular integration scheme used in the current work was first presented in [110] and applied to 2D vesicle flows in [83]. Other near-singular quadratures include the use of partition of unity along with polar coordinate transform [113]; the use of regularized kernel with corrections discussed in [100]; and the quadrature by expansion (QBX) scheme of [62] applied to simulation of rigid bodies in [2]. In [25], near interactions are computed through simple upsampling.

The fast multipole method (FMM) for gravitational N-body problems is discussed in [26]. FMM for the Stokes kernel includes the work of [101] and the STKFMMLIB3D library [42]. In the current work, we have used the kernel independent FMM (KIFMM) of [109] implemented in the PVFMM library [75]. A discussion of fast multipole accelerated boundary element methods can be found in [70].

Collision handling in 2D using repulsion is discussed in [37]. In 3D, a repulsion based scheme similar to ours is discussed in [71]. In [113], collisions are handled by moving the mesh points away when surfaces are closer than 2% of the cell radius.

A novel adaptive time-stepping scheme which we use in this work was introduced in [85, 84]. They also discuss a high-order spectral deferred correction (SDC) time-stepping.

5.1.4 Organization of the Chapter

In Section 5.2, we introduce the mathematical model and the integral equation formulation for vesicles embedded in a Stokesian fluid. Then, in Section 5.3, we discuss the discretization and the algorithms for numerically solving the discretized equations. Finally, in Section 5.4, we present convergence and scalability results. In Table 5.1, we list some frequently used symbols for easy reference.

Symbol	Definition
$\mathbb{S}^2$	Unit sphere
(θ, ϕ)	Spherical angles
Y_{nm}	Spherical harmonic function of degree n and order m
p	Degree of spherical harmonic expansion
q	Order of quadrature scheme
N_γ	Number of vesicles
γ_i	Boundary of i^{th} vesicle
W	Area element
S_i	The single-layer Stokes operator over i^{th} surface
D_i	The double-layer Stokes operator over i^{th} surface
μ	Viscosity of ambient fluid
μ_i	Viscosity of fluid in i^{th} vesicle
ρ	Density of ambient fluid
ρ_i	Density of fluid in i^{th} vesicle
σ	Tension
$\boldsymbol{f}_b$	Bending force
$\boldsymbol{f}_\sigma$	Tension force
$\boldsymbol{f}_r$	Repulsion force

Symbol	Definition
R_{repul}	Repulsion parameter: determines the range of repulsion force
$\boldsymbol{u}$	Velocity
$\boldsymbol{u}^\infty$	Background velocity
T	Time-horizon of a simulation
Δt	Time-step size
$\mathcal{E}$	Error tolerance for time-adaptivity
E	Reparameterization energy function
a_n	Attenuation coefficients which define reparameterization energy function
$\Delta\tau_{max}$	Maximum reparameterization step size
n_p	Number of MPI processes
ϵ_{GMRES}	GMRES tolerance
N_{iter}	Average GMRES iterations per solve
N_{Tstep}	Number of time steps
T_{solve}	Time to solution
T_{setup}	Setup time
T_{self}	Singular integration time
T_{near}	Near-singular integration time
T_{far}	Far-field integration time
T_{repar}	Reparameterization time

Table 5.1 *Index of frequently used symbols.*

5.2 Formulation

We have used the formulation of [88] and we briefly summarize it in this section.

Differential Formulation. We assume that the vesicle membrane γ has zero thickness, is locally inextensible and offers resistance to bending. In the length scales of vesicles, we can safely neglect the inertia term and assume the surrounding fluid to be Stokesian. The ambient fluid has viscosity μ and the fluid inside the i^{th} vesicle has viscosity μ_i. The dynamics of the ambient fluid are governed by the incompressible Stokes equation,

$$-\mu\Delta\boldsymbol{u}(\boldsymbol{x}) + \nabla p(\boldsymbol{x}) = 0 \quad \text{and} \quad \operatorname{div}\boldsymbol{u}(\boldsymbol{x}) = 0 \quad \text{for all } \boldsymbol{x} \in \mathbf{R}^3\backslash\omega, \tag{5.1}$$

where $\mathbf{R}^3\backslash\omega$ is the exterior region occupied by the suspending fluid, $\boldsymbol{u}(\boldsymbol{x})$ denotes the fluid velocity, and $p(\boldsymbol{x})$ denotes the pressure. Replacing μ with μ_i in Eq. (5.1) gives the governing equation for $\boldsymbol{x} \in \omega_i$ (the fluid inside the i^{th} vesicle). Due to no-slip at the vesicle boundary, we have the relation

$$\frac{\partial\boldsymbol{x}}{\partial t} = \boldsymbol{u}(\boldsymbol{x}) \quad \text{for all } \boldsymbol{x} \in \gamma, \quad \boldsymbol{u}(\boldsymbol{x}) \to \boldsymbol{u}^\infty(\boldsymbol{x}) \quad \text{as } \|\boldsymbol{x}\| \to \infty, \tag{5.2}$$

where $\boldsymbol{u}^\infty$ is the imposed far field velocity field. The inextensibility of the vesicle membrane implies that the surface divergence of the velocity field should be zero

$$\operatorname{div}_\gamma \boldsymbol{u}(\boldsymbol{x}) = 0 \quad \text{for all } \boldsymbol{x} \in \gamma. \tag{5.3}$$

Due to balance of momentum, the traction jump at the vesicle surface is balanced by the interfacial forces,

$$[\![T\boldsymbol{n}]\!] = \boldsymbol{f}_b(\boldsymbol{x}) + \boldsymbol{f}_\sigma(\boldsymbol{x}) \quad \text{for all } \boldsymbol{x} \in \gamma, \tag{5.4}$$

where $T = -pI + \mu(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T)$ is the Cauchy stress tensor, $\boldsymbol{n}$ is the surface normal at point $\boldsymbol{x}$, $[\![\cdot]\!]$ denotes the jump across the interface, $\boldsymbol{f}_b$ and $\boldsymbol{f}_\sigma$ are the bending and tensile forces exerted by the membrane on the fluid.

Boundary Integral Formulation. Eqs. (5.1–5.4) can be reformulated as,

$$\boldsymbol{u}(\boldsymbol{x}) = \frac{1}{\alpha_i}\left(\boldsymbol{u}^\infty(\boldsymbol{x}) + \sum_{j=1}^{N_\gamma} \mathcal{S}_j[\boldsymbol{f}_b + \boldsymbol{f}_\sigma](\boldsymbol{x}) + \mathcal{D}_j[\boldsymbol{u}](\boldsymbol{x})\right), \tag{5.5}$$

$$\operatorname{div}_{\gamma_i} \boldsymbol{u}(\boldsymbol{x}) = 0, \tag{5.6}$$

$$\frac{\partial \boldsymbol{x}}{\partial t} = \boldsymbol{u}(\boldsymbol{x}) \quad \text{for all } \boldsymbol{x} \in \gamma. \tag{5.7}$$

where $\alpha_i = (1 + \lambda_i)/2$. The notation $\mathcal{S}_i[\cdot](\boldsymbol{x})$ denotes the single-layer integral over the i^{th} surface and is defined as,

$$\mathcal{S}_i[\boldsymbol{f}](\boldsymbol{x}) := \int_{\gamma_i} S(\boldsymbol{x}, \boldsymbol{y})\boldsymbol{f}(\boldsymbol{y})\,\mathrm{d}\gamma(\boldsymbol{y}), \quad S(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{8\pi\mu}\frac{1}{\|\boldsymbol{r}\|}\left(I + \frac{\boldsymbol{r} \otimes \boldsymbol{r}}{\|\boldsymbol{r}\|^2}\right), \tag{5.8}$$

where, $\boldsymbol{r} := \boldsymbol{x} - \boldsymbol{y}$. $\mathcal{D}_i[\cdot](\boldsymbol{x})$ is the double-layer integral over the i^{th} surface,

$$\mathcal{D}_i[\boldsymbol{u}](\boldsymbol{x}) := \int_{\gamma_i} D_i(\boldsymbol{x}, \boldsymbol{y})\boldsymbol{u}(\boldsymbol{y})\,\mathrm{d}\gamma(\boldsymbol{y}), \quad D_i(\boldsymbol{x}, \boldsymbol{y}) = -\frac{3(1 - \lambda_i)}{4\pi}\frac{(\boldsymbol{r} \cdot \boldsymbol{n})(\boldsymbol{r} \otimes \boldsymbol{r})}{\|\boldsymbol{r}\|^5}. \tag{5.9}$$

Galerkin Formulation. The spherical harmonic function of degree n ($n = 0, 1, 2, \cdots$) and order m ($|m| \leq n$) is defined as

$$Y_{nm}(\theta, \phi) = \frac{1}{\sqrt{2\pi}} P_{nm}(\cos\theta)e^{im\phi} \tag{5.10}$$

where, P_{nm} is the normalized associated Legendre polynomial of degree n and order m. The normalization is such that $\int_{-1}^{1} P_{lm}(x)P_{nm}(x)\,\mathrm{d}x = \delta_{lm}$ for any fixed m. The Galerkin formulation is then derived from Eqs. (5.5–5.7) by computing the inner product $(\cdot, \cdot)$ with the spherical harmonic functions Y_{nm},

$$\alpha_i\left(\boldsymbol{u}, Y_{nm}\right) = \left(\boldsymbol{u}^{\infty}, Y_{nm}\right) + \sum_{j=1}^{N_\gamma}\left(\mathcal{S}_j[\boldsymbol{f}_b + \boldsymbol{f}_\sigma], Y_{nm}\right) + \left(\mathcal{D}_j[\boldsymbol{u}], Y_{nm}\right), \tag{5.11}$$

$$\left(\operatorname{div}_{\gamma_i} \boldsymbol{u}, Y_{nm}\right) = 0, \quad \text{for all } i = 1, \ldots, N_\gamma, \tag{5.12}$$

$$\left(\frac{\partial \boldsymbol{x}}{\partial t}, Y_{nm}\right) = \left(\boldsymbol{u}, Y_{nm}\right) \tag{5.13}$$

for all $n = 0, 1, \ldots$ and $|m| \leq n$.

5.3 Numerical Algorithms

We discretize the Galerkin formulation discussed in the previous section. We discuss the spatial discretization in Section 5.3.1 and algorithms for computing the Stokes single-layer and double-layer potentials from the vesicle surface in Section 5.3.2. In Section 5.3.3 we present an algorithm for detecting collisions between vesicles and introduce a repulsion term in our formulation to avoid such collisions. We present an algorithm for correcting the drift in area and volume of the vesicles in Section 5.3.4. In Section 5.3.5, we discuss the reparameterization algorithm and analyze the scheme for different choices of the attenuation coefficient. Then, we present a first order semi-implicit time-stepping scheme in Section 5.3.6 and discuss an algorithm for selecting the optimal time step size in Section 5.3.7. Finally, we summarize the overall algorithm in Section 5.3.8.

5.3.1 Spatial Discretization

We assume that each surface γ is smooth and homeomorphic to a sphere. Therefore, we can construct a C^∞ map from points on γ to points on the surface of the unit sphere $\mathbb{S}^2$. This mapping is not unique and can affect the magnitude of truncation errors when the surface is discretized. We will discuss these issues later in Section 5.3.5. The surface of the unit sphere can be parameterized by the spherical angles $(\theta, \phi) \in [0, \pi] \times [0, 2\pi)$. We approximate a function f on γ (mapped to $\mathbb{S}^2$) using the spherical harmonic basis Y_{nm} up to degree p

$$f(\theta, \phi) \approx \sum_{n=0}^{p} \sum_{m=-n}^{n} \widehat{f}_{nm} Y_{nm}(\theta, \phi) \tag{5.14}$$

where, $\widehat{f}_{nm}$ are the coefficients in the spherical harmonic expansion. Using orthonormality of the spherical harmonic basis, we can determine these coefficients using the relation

$\widehat{f}_{nm} = (f, Y_{nm})$. We construct such spherical harmonic approximations for the surface position, the tension and bending forces and the velocity of surface points.

Evaluation on Nodal Basis. When computing integrals over a surface γ, we require a nodal basis representation of the surface instead of the spherical harmonic basis discussed above. For q^{th} order quadratures, we discretize the spherical angles using a $(q+1) \times 2q$ grid of points (θ_i, ϕ_j) given by

$$\begin{aligned} \theta_i &= \cos^{-1} x_i && \text{for } i = 0, \cdots, q \\ \phi_j &= \pi/q\, j && \text{for } j = 0, \cdots, 2q-1 \end{aligned}$$

where, x_i are the roots of the Legendre polynomial of degree $q+1$. We refer to this as the q-grid. We evaluate the spherical harmonic representation $\widehat{f}$ at points on the q-grid as follows

$$f_{ij} = \sum_{n=0}^{p} \sum_{m=-n}^{n} \widehat{f}_{nm}\, Y_{nm}(\theta_i, \phi_j) \quad \text{for all} \quad i = 0, \cdots, q \quad \text{and} \quad j = 0, \cdots, 2q-1. \tag{5.15}$$

We define a linear operator Y^q which implements the above computation, so that $f = Y^q\, \widehat{f}$. Since the spherical harmonic basis functions Y_{nm} are products of the associated Legendre polynomials $P_{nm}(\cos\theta)$ and the Fourier basis functions $e^{im\phi}$, the above transform can be computed efficiently using a tensor product rule. We first evaluate the associated Legendre polynomials at $x_i = \cos\theta_i$ for $i = 0, \cdots, q$. This is followed by computing $q+1$ discrete Fourier transforms in ϕ_i. The method requires $\mathcal{O}\left(p^2 q + pq^2\right)$ work; with $\mathcal{O}\left(p^2 q\right)$ work for computing the associated Legendre polynomials and $\mathcal{O}\left(pq^2\right)$ work for the Fourier transform. We could use FFT for computing the Fourier transform and FLT (Fast Legendre Transform) for evaluating the associated Legendre polynomials to reduce the complexity to $\mathcal{O}\left(q^2 \log^2 q\right)$ work (when $q \geq p$ and using zero padding); however, this does not provide any noticeable improvement in performance for the small discretization orders ($p, q \leq 32$) used in this work.

We also use Eq. (5.15) to compute derivatives in θ and ϕ by replacing Y_{nm} with $\partial Y_{nm}/\partial\theta$ and $\partial Y_{nm}/\partial\phi$ respectively. Evaluating the derivatives on the q-grid from a spherical harmonic approximation of degree p requires $\mathcal{O}\left(p^2 q + pq^2\right)$ work.

Projection to Spherical Harmonic Basis. We often have to compute a projection to the spherical harmonic space $\widehat{f}$ from function values f_{ij} on the q-grid where, $q \geq p$. To do this, we use the relation $\widehat{f}_{nm} = (f, Y_{nm})$ and compute the inner product using a quadrature rule

$$\widehat{f}_{nm} = \sum_{i=0}^{q} \sum_{j=0}^{2q-1} f_{ij}\, Y_{nm}(\theta_i, \phi_j)\, \frac{\pi}{q}\, w_i \qquad \text{for all} \quad n = 0, \cdots, p \quad \text{and} \quad |m| \le n. \tag{5.16}$$

Here, we have used the Gauss-Legendre quadrature rule (with weights w_i) to integrate in ϕ and trapezoidal rule (with weights π/q) to integrate in θ. As before for Eq. (5.15), this sum can be evaluated efficiently using a tensor product rule and this requires $\mathcal{O}\left(p^2 q + pq^2\right)$ work. We define a linear operator Y^p_{proj} which computes the above projection, so that $\widehat{f} = Y^p_{proj}\, f$.

5.3.2 Stokes Layer Potentials

In our boundary integral formulation, we evaluate single- and double-layer potential from N_γ surfaces, with position $\boldsymbol{x}$, single-layer density $\boldsymbol{f}_s$ and double-layer density $\boldsymbol{f}_d$

$$\boldsymbol{u}(\boldsymbol{x}) = \sum_{k=1}^{N_\gamma} \mathcal{S}_k[\boldsymbol{f}_s](\boldsymbol{x}) + \mathcal{D}_k[\boldsymbol{f}_d](\boldsymbol{x}). \tag{5.17}$$

We evaluate this potential numerically on the p-grid for each surface. For the Galerkin formulation, we then compute a projection to the spherical harmonic space $\widehat{\boldsymbol{u}}_{nm} = (\boldsymbol{u}, Y_{nm})$. In the remainder of this section, we will only discuss computation of the single-layer potential; however, the algorithms are also applicable to computing the double-layer potential.

We consider computation of the Stokes single-layer potential $\mathcal{S}_\gamma[\boldsymbol{f}](\boldsymbol{y})$ from a single surface γ at a target point $\boldsymbol{y}$. We can express the integral over the surface γ as an integral over the spherical angles θ and ϕ as follows

$$\mathcal{S}_\gamma[\boldsymbol{f}](\boldsymbol{y}) = \int_\gamma S(\boldsymbol{y}, \boldsymbol{x})\, \boldsymbol{f}\ \mathrm{d}\gamma \ = \ \int_{\theta=0}^{\pi} \int_{\phi=0}^{2\pi} S(\boldsymbol{y}, \boldsymbol{x}(\theta, \phi))\, \boldsymbol{f}(\theta, \phi)\, W(\theta, \phi)\ \mathrm{d}\phi\ \mathrm{d}\theta \tag{5.18}$$

where, $W(\theta, \phi) = \sqrt{EG - F^2}$ is the area element of the surface (with E, F and G denoting the coefficients of the first fundamental form of γ). When $\boldsymbol{y}$ is not on the surface, then the integrand is smooth and standard quadratures (Gauss-Legendre quadrature for θ and trapezoidal quadrature for ϕ directions) are sufficient. We discretize the integral in Eq. (5.18) by a quadrature rule over the q-grid

$$\mathcal{S}_\gamma[\boldsymbol{f}](\boldsymbol{y}) \approx \sum_{i=0}^{q} \sum_{j=0}^{2q-1} S(\boldsymbol{y}, \boldsymbol{x}_{ij})\, \boldsymbol{f}_{ij}\, W_{ij}\, \frac{\pi}{q}\, w_i \tag{5.19}$$

where, w_i are the Gauss-Legendre quadrature weights and $\boldsymbol{x}_{ij}$, $\boldsymbol{f}_{ij}$ and W_{ij} are the surface position, the density function and the area element evaluated on the q-grid. This computation requires $\mathcal{O}\left(q^2\right)$ work for each evaluation point. The method is spectrally convergent in q; however, to be accurate, it requires that the distance between the points on the q-grid should be smaller than the distance between the surface and the evaluation points. Therefore, for $h = \min|\boldsymbol{x}(\theta, \phi) - \boldsymbol{y}|$, we must have $q = \mathcal{O}\left(h^{-1}\right)$. The method becomes prohibitively expensive as h becomes smaller and does not converge for $h = 0$. Next, we discuss special quadratures for computing these singular and near-singular integrals efficiently.

Singular Integration. We use the algorithm discussed in [105] for the Stokes single-layer potential and extended to the Stokes double-layer potential in [88]. This singular integration scheme is spectrally convergent for both single- and double-layer potentials. Below, we briefly summarize this algorithm for computing $\widehat{\boldsymbol{u}}_{nm} = (\mathcal{S}_\gamma[\boldsymbol{f}], Y_{nm})$ for a single surface γ.

We define the linear operators $\mathrm{R}(\theta, \phi)$ which transforms the coefficients in a spherical harmonic expansion to the coefficients in a rotated coordinate space with the north pole at (θ, ϕ). The construction of these operators is discussed in [43]. For spherical harmonic discretization of degree p, the application of the operator requires $\mathcal{O}\left(p^3\right)$ work for each surface. We use these operators to compute the spherical harmonic expansions of the surface position in rotated coordinate space: $\widehat{\boldsymbol{x}}^{ij} = \mathrm{R}(\theta_i, \phi_j)\, \widehat{\boldsymbol{x}}$ for each point (θ_i, ϕ_j) on the p-grid. Then, we evaluate the spherical harmonic expansions on the q-grid: $\boldsymbol{x}^{ij} = Y^q\, \widehat{\boldsymbol{x}}^{ij}$. Similarly, we compute the single-layer density on the q-grid in rotated coordinate space: $\boldsymbol{f}^{ij} = Y^q\, \mathrm{R}(\theta_i, \phi_j)\, \widehat{\boldsymbol{f}}$. We also compute the area elements W^{ij} from $\widehat{\boldsymbol{x}}^{ij}$. This requires evaluating the derivatives $\partial \boldsymbol{x}/\partial\theta$ and $\partial \boldsymbol{x}/\partial\phi$ on the q-grid. Now, we compute the potential $\boldsymbol{u}_{ij} = \boldsymbol{u}(\theta_i, \phi_j)$ on the p-grid

$$\boldsymbol{u}_{ij} = \left(\Lambda_q \,\circ\, W^{ij} \,\circ\, S(\boldsymbol{x}_{ij}, \boldsymbol{x}^{ij})\right) \cdot \boldsymbol{f}^{ij} \quad \text{for all} \quad i = 0, \cdots, p \quad \text{and} \quad j = 0, \cdots, 2p-1 \tag{5.20}$$

where, Λ_q are the quadrature weights from the scheme of Graham-Sloan [47] for evaluating singular integrals at the pole. Here, we are computing the Stokes single-layer potential at the north pole $\boldsymbol{x}_{ij}$ from each point on the q-grid with position $\boldsymbol{x}^{ij}$ and density $\boldsymbol{f}^{ij}$ scaled by the area elements and the quadrature weights at each grid point. Finally, we compute the projection from $\boldsymbol{u}$ on the p-grid to the spherical harmonic space $\widehat{\boldsymbol{u}} = Y^p_{proj}\, \boldsymbol{u}$.

The above algorithm uses a q-grid for the singular quadrature rule instead of the p-grid used in [105]. We observed that depending on the desired accuracy of the result, we often need to use an upsampled grid such that $q > p$. Later in this section, we will discuss how we select the optimal value for q. The algorithm requires $\mathcal{O}\left(p^5\right)$ work for computing all the rotations, $\mathcal{O}\left(p^2(p^2q + pq^2)\right)$ work for evaluating spherical harmonic expansions on q-grid and $\mathcal{O}\left(p^2q^2\right)$ work for evaluating the Stokes operator and computing the weighted inner-product with the density. Overall, this requires $\mathcal{O}\left(p^5 + p^3q^2\right)$ work each time we compute $\widehat{\boldsymbol{u}}$.

In our semi-implicit time-stepping scheme, we use GMRES to solve for the new surface position. Each GMRES iteration involves computing $\boldsymbol{u}$ for the same surface position $\boldsymbol{x}$ but different densities $\boldsymbol{f}$. Therefore, we modify the above algorithm to instead precompute the operator matrix

$$\mathbf{S}_{ij} = \left(\Lambda_q \circ W^{ij} \circ S(\boldsymbol{x}_{ij}, \boldsymbol{x}^{ij})\right) \cdot Y^q \, \mathrm{R}(\theta_i, \phi_j) \quad \text{for all} \quad i = 0, \cdots, p \quad \text{and} \quad j = 0, \cdots, 2p-1$$
$$\widehat{\mathbf{S}} = Y^p_{proj} \, \mathbf{S}$$

where, $\widehat{\mathbf{S}}$ is a $(p+1)^2 \times (p+1)^2$ matrix. In each GMRES iteration, we can now compute $\widehat{\boldsymbol{u}} = \widehat{\mathbf{S}} \, \widehat{\boldsymbol{f}}$. Computing $\widehat{\mathbf{S}}$ still requires $\mathcal{O}\left(p^5 + p^3q^2\right)$ work per surface; however, each subsequent application of $\widehat{\mathbf{S}}$ only requires $\mathcal{O}\left(p^4\right)$ work. This results in a significant improvement in performance over the original scheme.

Near-Singular Integration. For computing interactions between a surface γ and a target point $\boldsymbol{y}$, such that $\boldsymbol{y}$ is not on the surface but at a distance smaller than h_n from the surface, we use a near-singular integration scheme. The scheme is adapted from the method of [110]. The value of h_n depends on the order q of the Nyström scheme used in Eq. (5.19). For a given q, the optimal choice for h_n has to be determined empirically; however, we observed that the optimal value can be estimated by the relation $h_n = \sqrt{A/q}$ where, A is the maximum surface area of any vesicle. For this choice of h_n, we still get spectral convergence with the Nyström scheme (for target points at a distance h_n or greater from the surface), since distance between points on the q-grid $\mathcal{O}\left(\sqrt{A}/q\right)$ decreases faster than h_n as we increase q. For points closer than h_n to the surface, the near-singular integration algorithm has the following sequence of steps.

(a) *Identifying Near Points:* The first step in the near-singular integration scheme is to identify, all pairs of surface and target points $(\gamma_i, \boldsymbol{y})$ that are separated by a distance smaller than h_n. We can do this by comparing the distance between every target point and every point on the discretized surface (q-grid). For N_{trg} target points and N_γ surfaces, this requires $\mathcal{O}\left(N_{trg}N_\gamma\right)$ work and can be very expensive when N_γ and N_{trg} are large. A more efficient method is to sort all the surface discretization points,

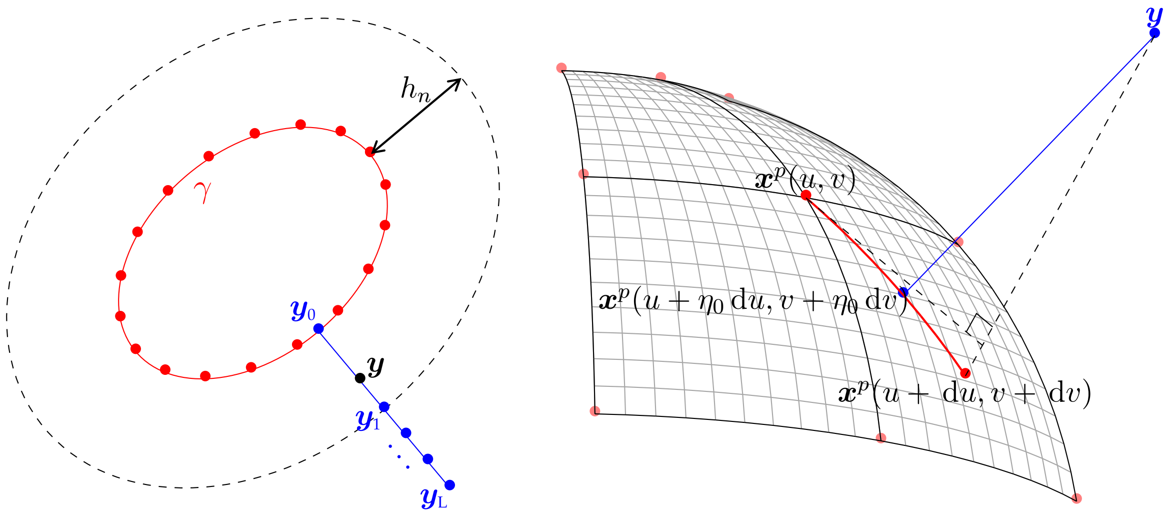


Figure 5.1 *Left: 2D schematic of vesicle surface (γ) and the surface discretization points. The Nyström scheme is accurate for target points at a distance h_n or greater from the surface. To compute velocity at the point $\boldsymbol{y}$, we determine the nearest point $\boldsymbol{y}_0$ on the surface. We compute the velocity at $\boldsymbol{y}_0$ using a singular quadrature scheme and interpolation on the surface. Using the Nyström scheme, we compute the velocity at a sequence of points $\boldsymbol{y}_1, \cdots, \boldsymbol{y}_L$ on the line through $\boldsymbol{y}_0$ and $\boldsymbol{y}$. We compute the velocity at $\boldsymbol{y}$ by interpolating the velocity at $\boldsymbol{y}_0, \cdots, \boldsymbol{y}_L$. Right: Quadratic patch created by interpolating a 3×3 grid of surface points. We show the first iteration for computing the projection of the target point $\boldsymbol{y}$ on this patch. We start at the center of the patch with $(u, v) = (0, 0)$ and compute the update $(\mathrm{d}u, \mathrm{d}v)$ by approximating the surface by the tangent plane passing through $\boldsymbol{x}^p(u, v)$. Then we search for the surface point nearest to $\boldsymbol{y}$ along the line $(u, v) - (u + \mathrm{d}u, v + \mathrm{d}v)$ in parameter space. We iterate until we reach the edge of the patch or the updates are small enough ($|(\mathrm{d}u, \mathrm{d}v)| \leq$ 1E-6).*

then for each target point we can find the surface points close to it by searching in a sorted array. This can be implemented in a number of ways, such as: using radix sort to bin points in $h_n \times h_n \times h_n$ size boxes; using an octree of depth $\log_2 h_n^{-1}$; or sorting points on a space-filling curve. In our implementation, we use the last approach since it is easy to parallelize. We compute the Morton Id with depth $\log_2 h_n^{-1}$ for each surface point and sort these using a parallel sorting algorithm [96]. The parallel sorting algorithm requires $\mathcal{O}\left(N/n_p \log N/n_p + N/n_p \log n_p\right)$ time for N points on n_p processes. For each target point, we compute its Morton Id and also the neighboring 26 Morton Ids. We find all the surface points with one of these 27 Morton Ids using binary search and compare the distance between these surface points and the target point. We make a pair of the target point and the surface for any surface with a discretization point closer than h_n from the target point.

When we do this in parallel (distributed memory systems), the target points must be on the same MPI process as the surface points when doing the local binary search. Therefore, we also sort the target points by their Morton Id and partition them across MPI processes using the same partitioning as for the surface points. Another advantage of doing this is that we can now process all the target points with the same Morton Id together and avoid repeated binary searches in the array of surface points. For our parallel implementation, we also have to add ghost Morton Ids to the array of surface points so that for each target point we have all the 27 adjacent Morton Ids available locally on the MPI process. We identify these ghost Morton Ids and communicate them using point-to-point communication. In addition, once we form pairs of surface and target points, we have to send these to the process where the surface originated before the parallel sort. The rest of the near-singular integration algorithm can then proceed independently on each MPI process.

(b) *Projection on the Surface:* For each pair of surface γ and near target point $\boldsymbol{y}$, we determine the projection $\boldsymbol{y}_0$ of the target point on the surface (see Fig. 5.1). To do this, we determine the surface discretization point which is closest to $\boldsymbol{y}$ and select the 3×3 grid of points in the surface mesh around this point. If one of the poles is the nearest point, then we select the pole and eight other points adjacent to the pole to form the 3×3 grid. We create a quadratic surface interpolant $\boldsymbol{x}^p(u, v)$ from this grid, such that $\boldsymbol{x}^p(0, 0)$ is the nearest grid point to $\boldsymbol{y}$. Now, we use an iterative scheme to find the point on this interpolant which is closest to the target point $\boldsymbol{y}$. We start from the center of the patch ($u = 0$ and $v = 0$). In each iteration, we linearize the interpolant $\boldsymbol{x}^p$ around u and v so that

$$\boldsymbol{x}^p(u + \mathrm{d}u, v + \mathrm{d}v) \approx \boldsymbol{x}^p(u, v) + \nabla \boldsymbol{x}^p(u, v)\, [\,\mathrm{d}u \;\; \mathrm{d}v]^T \tag{5.21}$$

We want to find the least squares solution to $\boldsymbol{x}^p(u + \mathrm{d}u, v + \mathrm{d}v) = \boldsymbol{y}$. Substituting in the above equation and solving for $[\,\mathrm{d}u \;\; \mathrm{d}v]$, we have

$$[\,\mathrm{d}u \;\; \mathrm{d}v]^T = (\nabla \boldsymbol{x}^p)^+ \, (\boldsymbol{y} - \boldsymbol{x}^p) \tag{5.22}$$

where, $(\nabla \boldsymbol{x}^p)^+$ is the pseudo-inverse of $\nabla \boldsymbol{x}^p(u, v)$. To avoid overshooting (due to the curvature of the surface) we now search along the line $(u, v) - (u + \mathrm{d}u, v + \mathrm{d}v)$ in parameter space; i.e. we try to find $\eta_0 \in [0, 1]$ such that $\boldsymbol{x}^p(u + \eta_0\, \mathrm{d}u, v + \eta_0\, \mathrm{d}v)$ is nearest to $\boldsymbol{y}$. We build a quadratic interpolant $p(\eta)$ such that $p(\eta) = |\boldsymbol{x}^p(u + \eta\, \mathrm{d}u, v + \eta\, \mathrm{d}v) - \boldsymbol{y}|$ for $\eta = 0, 0.5, 1$ and compute $\eta_0 = \arg\min_{\eta \in [0,1]} p(\eta)$. We update $(u, v) {+}{=} \eta_0(\,\mathrm{d}u,\, \mathrm{d}v)$ and repeat the above process until we reach the edge of the patch or the updates

$|(\mathrm{d}u, \mathrm{d}v)|$ are smaller than a given tolerance (about 1E-6). When the method converges, we obtain the projection $\boldsymbol{y}_0 = \boldsymbol{x}^p(u, v)$ of $\boldsymbol{y}$ on the surface γ. In our experiments, the method converged to single-precision accuracy in about 10 iterations.

(c) *Interpolation:* For each surface γ, we compute the Stokes singular integral $\widehat{\boldsymbol{u}}$ as discussed in the previous section and evaluate it on the q-grid. Now, for each near point $\boldsymbol{y}$ of γ, we interpolate the singular potential at the projection $\boldsymbol{y}_0$ using the quadratic surface interpolant described above to obtain $\boldsymbol{u}_0 = \boldsymbol{u}(\boldsymbol{y}_0)$. We construct a set of points $\boldsymbol{y}_1, \cdots, \boldsymbol{y}_L$ on the line through $\boldsymbol{y}_0$ and $\boldsymbol{y}$; distributed evenly between distances h_n and $2h_n$ from point $\boldsymbol{y}_0$. Since these points are at a distance greater than or equal to h_n, we can use the Nyström scheme to compute the potential $\boldsymbol{u}_1, \cdots, \boldsymbol{u}_L$ at the points $\boldsymbol{y}_1, \cdots, \boldsymbol{y}_L$ respectively. We now construct a Lagrange polynomial interpolant of degree L for the potential $\{\boldsymbol{u}_0, \cdots, \boldsymbol{u}_L\}$ at points $\{\boldsymbol{y}_0, \cdots, \boldsymbol{y}_L\}$. Finally, we evaluate this interpolant at $\boldsymbol{y}$ to obtain the potential $\boldsymbol{u}_{\boldsymbol{y}} = \boldsymbol{u}(\boldsymbol{y})$. In our implementation, we have used the interpolation order $L = 8$.

The algorithm assumes that the interpolation points $\boldsymbol{y}_1, \cdots, \boldsymbol{y}_L$ are at a distance greater than h_n from γ so that the potential at these points is smooth. However, this may not be true when the surface has large deformations. In this case, we need to use a larger q so that h_n is smaller and therefore, the interpolation points are separated from the surface by a distance greater than h_n. This requires adaptively choosing the appropriate q and is discussed later in this section.

The steps (a) and (b) are part of the setup phase and are compute once per time step; however, step (c) must be evaluated each time the layer potential is computed. For N_γ surfaces, N_{trg} near target points and q^{th} order quadratures (where $q > p$), step (a) requires $\mathcal{O}\left(N_\gamma q^2/n_p \log\left(N_\gamma q^2/n_p\right) + N_\gamma q^2/n_p \log n_p\right)$ time, step (b) requires $\mathcal{O}\left(N_{trg}/n_p\right)$ time and step (c) requires $\mathcal{O}\left(N_{trg} L q^2/n_p\right)$ time on n_p processors. Here, $L = 8$ is the order of the Lagrange interpolation discussed above. In most cases, we can approximate $N_{trg} = \mathcal{O}\left(N_\gamma p^2\right)$.

Far-Field Integration. Computing the summation in Eq. (5.17), using the Nyström integration scheme in Eq. (5.19), requires $\mathcal{O}\left(N_\gamma^2 p^2 q^2\right)$ work for N_γ surfaces with spherical harmonic discretization of degree p and q^{th} order quadrature scheme. This is an N-body problem with $\mathcal{O}\left(N_\gamma p^2\right)$ target points and $\mathcal{O}\left(N_\gamma q^2\right)$ source points; with the source densities scaled by the area element and the quadrature weights.

We can accelerate the above computation by using the Fast Multipole Method (FMM) [26] and compute solutions in $\mathcal{O}\left(N_\gamma(q^2+p^2)\right)$ work. We use our PVFMM library [75] which is an optimized, parallel implementation of the Kernel Independent FMM scheme of [109]. For $N = N_\gamma(p^2+q^2)$ source and target points, our algorithm requires $\mathcal{O}\left(N/n_p \log(N/n_p) + N/n_p \log n_p\right)$ setup time for tree construction and $\mathcal{O}\left(N/n_p + (N/n_p)^{2/3} \log n_p\right)$ time for each subsequent evaluation on n_p processes. In addition to free-space boundary conditions, the library can also compute periodic solutions by creating an infinite periodic tiling of the source density. The far-field computation is by far the most expensive computation in our scheme and the use of the PVFMM library allows us to compute solutions efficiently and scale our scheme to a large number of compute nodes.

Using FMM requires that we compute interactions between all pairs of source and target points. Therefore, we have to use direct summation to compute interactions with the near target points for each surface, subtract it from the FMM solution and then add the contributions from the singular and near-singular integration schemes discussed above. Computing these summations requires an additional $\mathcal{O}\left(N_{trg}q^2\right)$ work, where $N_{trg} = \mathcal{O}\left(N_\gamma p^2\right)$; therefore, we still retain linear work complexity in N_γ. While it may be more efficient to exclude these self- and near-interactions when computing the FMM sum, it is very complicated and would require implementing an entirely new FMM library.

Integration Error. We can estimate the error in the integration scheme discussed above by checking with a known eigenvalue of the double-layer operator. We evaluate the following double-layer integral

$$\boldsymbol{u}(\boldsymbol{y}) = \sum_{k=1}^{N_\gamma} \mathcal{D}_k[\boldsymbol{f}](\boldsymbol{y}) \tag{5.23}$$

where, the density $\boldsymbol{f} = 1$ on each surface. For closed and non-overlapping surfaces, this integral has an analytical solution: $\boldsymbol{u}(\boldsymbol{y}) = 1$ when $\boldsymbol{y}$ is enclosed by any surface; $\boldsymbol{u}(\boldsymbol{y}) = 0.5$ when $\boldsymbol{y}$ is on any surface; and $\boldsymbol{u}(\boldsymbol{y}) = 0$ when $\boldsymbol{y}$ is in the exterior of all surfaces. We use this analytical solution to estimate the error for each of the singular, near-singular and far-field integration schemes in each time-step. We then adjust the order of the quadrature schemes, by incrementing the order by one ($q \leftarrow q+1$) if the error is larger than the required accuracy and decrease the order by one ($q \leftarrow q-1$) if the error is smaller than the required accuracy. While we use the same quadrature order for the near-singular and far-field integration schemes, the order for the singular quadrature scheme can be different.

5.3.3 Collision Handling

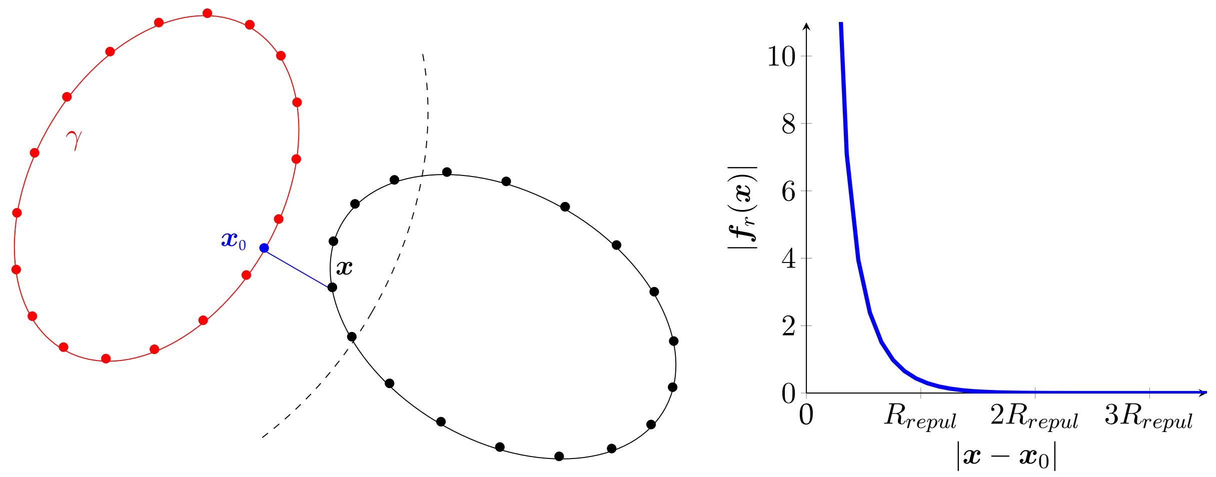


Figure 5.2 *Left: To determine if two vesicles intersect, for each point $\boldsymbol{x}$, we compute its normal projection $\boldsymbol{x}_0$ on the other vesicle. If the vesicles intersect, then $\boldsymbol{n}_{\boldsymbol{x}_0} \cdot (\boldsymbol{x} - \boldsymbol{x}_0) \leq 0$. We can also add a repulsion force given by $\boldsymbol{f}_r(\boldsymbol{x}) = \int_{\boldsymbol{y} \in \gamma} K(\boldsymbol{x} - \boldsymbol{y})$. Right: Plot of the repulsion force $\boldsymbol{f}_r(\boldsymbol{x})$ as a function of the distance $|\boldsymbol{x} - \boldsymbol{x}_0|$ for the repulsion function given in Eq. (5.26). The parameter R_{repul} controls the range of the repulsion force. The repulsion becomes infinitely large as the surfaces approach each other and decays rapidly as the distance between them increase. At $|\boldsymbol{x} - \boldsymbol{x}_0| = 3R_{repul}$, the repulsion force $|\boldsymbol{f}_r(\boldsymbol{x})| \sim$ 1E-5.*

During the course of a numerical simulation, two vesicles may intersect. This happens due to various discretization errors introduced in the simulation. When this happens, it leads to non-physical behavior and the simulation may even break (GMRES may fail to converge).

In every time step, after computing the updated vesicle position, we check if the vesicles intersect. To do this, for each surface mesh point $\boldsymbol{x}$, we find the surfaces near this point and for every such surface γ, we determine the normal projection $\boldsymbol{x}_0$ of $\boldsymbol{x}$ on γ. This is already done as part of the near-singular integration discussed in Section 5.3.2 and therefore, we need to do this only once per time step. We compute the dot-product of the outward surface normal vector $\boldsymbol{n}_{\boldsymbol{x}_0}$ at $\boldsymbol{x}_0$ with the vector $\boldsymbol{x} - \boldsymbol{x}_0$. If the dot-product $\boldsymbol{n}_{\boldsymbol{x}_0} \cdot (\boldsymbol{x} - \boldsymbol{x}_0) \leq 0$, then we know that the vesicles either touch or intersect. Note that this can sometimes fail since we only check for intersection at the mesh points on the vesicles and also because the quadratic patch is only an approximation of the actual vesicle surface. If we determine that the vesicles intersect, we reject the solution for that time step, reduce the time step by half and recompute the solution.

Despite the above adaptive time-stepping, there are still cases where it is not possible to avoid collision between surfaces. To address this issue, we introduce a repulsion force between the vesicles. The repulsion term is added along with the bending and tensile forces in our Galerkin formulation in Eq. (5.11) as follows

$$\alpha_i\left(\boldsymbol{u}, Y_{nm}\right) = \left(\boldsymbol{u}^{\infty}, Y_{nm}\right) + \sum_{j=1}^{N_\gamma} \left(\mathcal{S}_j[\boldsymbol{f}_b + \boldsymbol{f}_\sigma + \boldsymbol{f}_r], Y_{nm}\right) + \left(\mathcal{D}_j[\boldsymbol{u}], Y_{nm}\right) \tag{5.24}$$

where, the force $\boldsymbol{f}_r$ can be any highly localized repulsive force. We define $\boldsymbol{f}_r(\boldsymbol{x})$ by the convolution of the surface with a kernel function,

$$\boldsymbol{f}_r(\boldsymbol{x}) = \int_{\boldsymbol{y}\in\gamma} K(\boldsymbol{x}-\boldsymbol{y}), \qquad \text{where} \quad K(\boldsymbol{x}) = \left(\frac{3{R_{repul}}^4}{2|\boldsymbol{x}|^5} + \frac{{R_{repul}}^2}{|\boldsymbol{x}|^3}\right) \exp\left(\frac{-|\boldsymbol{x}|^2}{{R_{repul}}^2}\right) \boldsymbol{x} \tag{5.25}$$

where, K is the repulsion kernel function and the constant R_{repul} is related to the range of the repulsion force. We choose the repulsion kernel in such a way that the repulsion force becomes infinite as two surfaces approach each other. This allows the surfaces to come arbitrarily close but guarantees that they will not touch. The kernel function also decays quickly as the distance between the surfaces increases to ensure that the repulsion only comes into play when the surfaces are very close together.

Since the evaluation points are very close to the vesicle surface, a Nyström discretization of the integral in Eq. (5.25) will converge very slowly. Instead, we compute this integral analytically by assuming that the vesicle surface is nearly flat in the vicinity of the target point. The repulsion force can then be approximated as,

$$\boldsymbol{f}_r(\boldsymbol{x}) \;\approx\; \frac{{R_{repul}}^2}{|\boldsymbol{x}-\boldsymbol{x}_0|^3} \exp\left(\frac{-|\boldsymbol{x}-\boldsymbol{x}_0|^2}{{R_{repul}}^2}\right) (\boldsymbol{x}-\boldsymbol{x}_0) \tag{5.26}$$

When R_{repul} is smaller, the repulsion force is more localized but the equations become stiffer and we need to use smaller time step size. In our simulations we choose $R_{repul} =$ 2E-2.

5.3.4 Area and Volume Correction

The area and volume of a vesicle determine their dynamical behavior through reduced volume [1]. To avoid drift in these values we correct the area A and volume V at each time step by solving the following constraint minimization problem for each vesicle γ

$$\underset{\text{s.t. } A(\boldsymbol{x})=A_0, V(\boldsymbol{x})=V_0}{\arg\min} \frac{1}{2}\|\boldsymbol{x}-\boldsymbol{x}^\star\|^2_{L^2(\gamma)}, \tag{5.27}$$

where $\boldsymbol{x}^\star$ is the candidate position obtained through time-stepping and reparametrization, whose area and volume may have drifted from A_0 and V_0. One approach in solving this minimization problem is to linearize the constraints and solve the equality constrained quadratic program directly. The variations of the area and volume of a surface with displacement $\delta\boldsymbol{x}$ are given by [28, Section 9.4]

$$dA(\boldsymbol{x}, \delta\boldsymbol{x}) = -2 \int_{\gamma(\boldsymbol{x})} H \; \delta\boldsymbol{x} \cdot \boldsymbol{n} \, \mathrm{d}\gamma = -2 \, (H, \psi) \, , \tag{5.28}$$

$$dV(\boldsymbol{x}, \delta\boldsymbol{x}) = \int_{\gamma(\boldsymbol{x})} \delta\boldsymbol{x} \cdot \boldsymbol{n} \, \mathrm{d}\gamma = (1, \psi) \, . \tag{5.29}$$

In the last terms above, we restricted $\delta\boldsymbol{x}$ to the normal direction $\delta\boldsymbol{x} = \psi\boldsymbol{n}$ because these first variations only depend on the normal perturbation to the surface. The linearized minimization problem is then

$$\underset{\text{s.t.} \; \substack{dA(\boldsymbol{x},\psi)=\delta A \\ dV(\boldsymbol{x},\psi)=\delta V}}{\arg\min} \; \frac{1}{2} \, (\psi, \psi) \tag{5.30}$$

where $\delta A = A_0 - A(\boldsymbol{x})$ and $\delta V = V_0 - V(\boldsymbol{x})$. The Lagrangian for Eq. (5.30) is

$$\mathcal{L}(\psi, \alpha, \beta) = (\psi, \psi) + \alpha \, (dA(\boldsymbol{x}, \psi) - \delta A) + \beta \, (dV(\boldsymbol{x}, \psi) - \delta V) \, . \tag{5.31}$$

The derivative of $\mathcal{L}$ with respect to the Lagrange multipliers recovers the linearized constraints and the derivative with respect to ψ is

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\boldsymbol{x}}(\phi) = (\psi - 2H\alpha + \beta, \phi) \, , \tag{5.32}$$

which gives a linear system for the KKT conditions. The linear system can be simplified to a small system for the Lagrange multipliers:

$$\begin{bmatrix} -4 \, (H, H) & 2 \, (H, 1) \\ 2 \, (1, H) & - \, (1, 1) \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \delta A \\ \delta V \end{bmatrix} . \tag{5.33}$$

After computing α and β, we evaluate $\psi = 2\alpha H - \beta$ and move the surface to $\boldsymbol{x} + \psi\boldsymbol{n}$. Due to the linearization step we may need to iterate a few times to satisfy the constraint up to the given accuracy. In our experiments, this step typically converges in two or three iterations.

5.3.5 Reparameterization

As a simulation progresses, due to the absence of in-plane shear resistance in vesicles, the quality of the mesh eventually deteriorates. The grid points may cluster in some regions and become sparse in other regions. When unchecked, this leads to unresolvable high frequencies in the spherical harmonic expansion of the vesicle shape and interfacial forces. The distortion of the mesh may also adversely affect the accuracy of the singular integra-

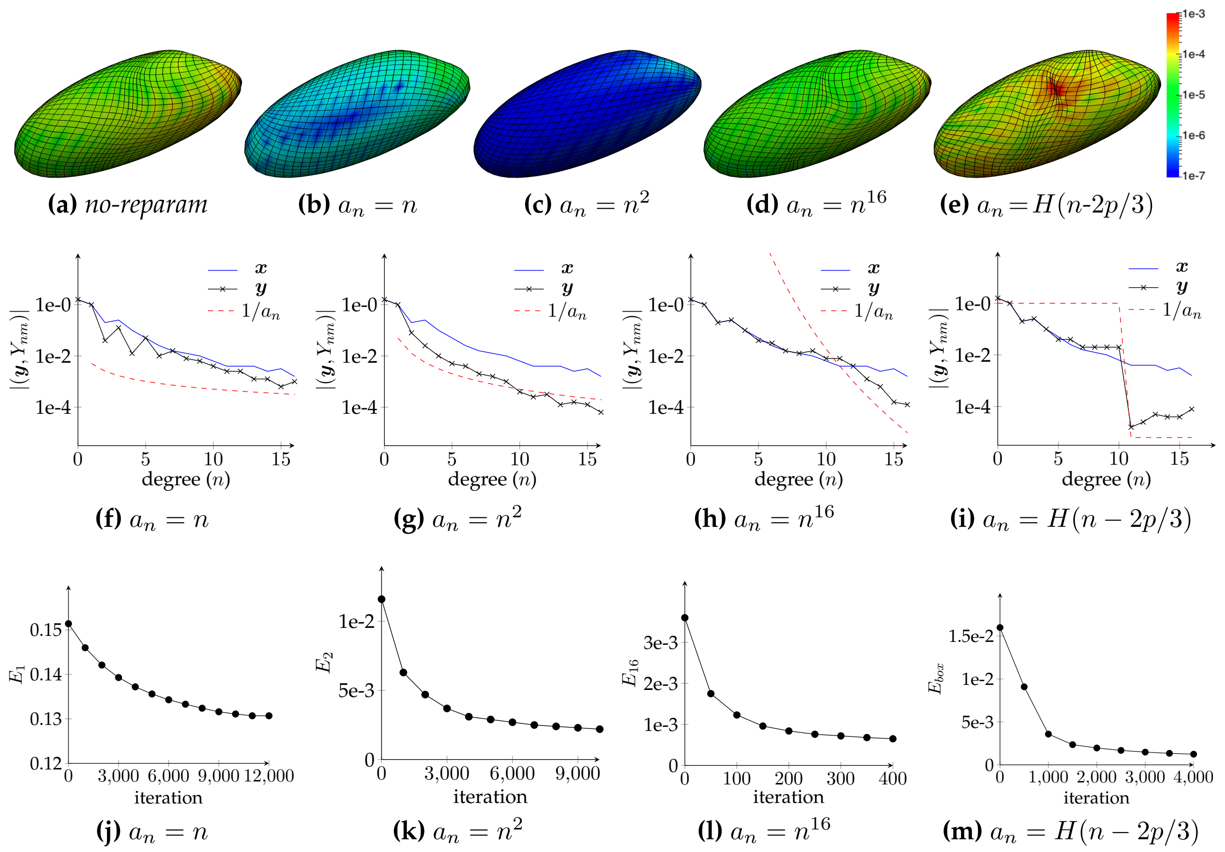


Figure 5.3 *Comparison of different reparameterization schemes. Fig. 5.3a shows the mesh and the singular integration error for a simulation without reparameterization. The reparameterized mesh for different schemes are shown in Figs. 5.3b to 5.3e. For each scheme, we also show the spectrum for the original mesh $\boldsymbol{x}$ and the reparameterized mesh $\boldsymbol{y}$ in Figs. 5.3f to 5.3i. The decay of the quality measure E with reparameterization iterations is shown in Figs. 5.3j to 5.3m. The reparameterization scheme with the attenuation coefficients $a_n = n^2$ works best as it has small truncation error (10^{-4}) for the spherical harmonic expansion and also has about 6-digits of accuracy for double-layer singular integration.*

tion scheme. In [105], this is solved by reparameterization of the vesicle surface after each time step. Below, we briefly outline this scheme.

The vesicle surface γ parameterized by spherical coordinates is given by the map $\boldsymbol{x}(s) : \mathbb{S}^2 \to \mathbf{R}^3$. Let $F : \mathbf{R}^3 \to \mathbf{R}$ denote the implicit representation of the surface such that $F(\gamma) = 0$ and ∇F does not vanish. Our goal is the to choose an alternate parameterization $\boldsymbol{y}(s) : \mathbb{S}^2 \to \mathbf{R}^3$ for the surface such that it minimizes the quality measure $E(\boldsymbol{y}) = (\boldsymbol{y}, \boldsymbol{y})_E$ with the inner product defined as $(\boldsymbol{x}, \boldsymbol{y})_E := \sum_{n=0}^{p} \sum_{m=-n}^{n} a_n^2 \, (\boldsymbol{x}, Y_{nm}) \, (\boldsymbol{y}, Y_{nm})$. Therefore, to obtain $\boldsymbol{y}$ we need to solve the following constrained minimization problem:

$$\underset{\boldsymbol{y}\in C^\infty(\mathbb{S}^2)}{\arg\min}\; E(\boldsymbol{y}) \qquad \text{subject to} \qquad F(\boldsymbol{y}(s)) = 0 \quad \text{for all} \quad s \in \mathbb{S}^2. \tag{5.34}$$

This is reformulated as a pseudo-transient continuation [61] problem and discretized using an explicit first order scheme (see [105] for details) as follows,

$$\boldsymbol{y}_{k+1} = \boldsymbol{y}_k - \boldsymbol{v}_k \Delta\tau \qquad \text{where,} \quad \boldsymbol{v}_k = \frac{(I - \boldsymbol{n}_k \otimes \boldsymbol{n}_k)\nabla E(\boldsymbol{y}_k)}{\|(I - \boldsymbol{n}_k \otimes \boldsymbol{n}_k)\nabla E(\boldsymbol{y}_k)\|_\infty} \tag{5.35}$$

We select the reparameterization step size $\Delta\tau$ to be the minimum of $\Delta\tau_{max}$ and $(\boldsymbol{y}_k, \boldsymbol{v}_k)_E \,/\, (\boldsymbol{v}_k, \boldsymbol{v}_k)_E$. This ensures that the first order reparameterization scheme does not introduce errors larger than $\mathcal{O}\left(\Delta\tau_{max}\right)$ and having $\Delta\tau \leq \Delta\tau_{max}$ and $(\boldsymbol{y}_k, \boldsymbol{v}_k)_E \,/\, (\boldsymbol{v}_k, \boldsymbol{v}_k)_E$ guarantees that in each iteration of the algorithm $E(\boldsymbol{y}_{k+1}) \leq E(\boldsymbol{y}_k)$, i.e. the quality measure $E(\boldsymbol{y})$ always decreases. We stop the algorithm when $\Delta\tau$ becomes smaller than a specified tolerance.

In Fig. 5.3, we analyze the performance of the reparameterization algorithm for different choices of the attenuation coefficients a_n in the definition of the quality measure E. Fig. 5.3a shows the mesh with degree $p = 16$ from a simulation without reparameterization. We reparameterize this mesh using different schemes in Figs. 5.3b to 5.3e. For each mesh we also visualize the singular integration error computed using the method discussed at the end of Section 5.3.2 for the singular-integration scheme of order $q = 32$. In Figs. 5.3f to 5.3i, we plot the spectrum of the spherical harmonic expansion of the reparameterized mesh $\boldsymbol{y}$ and compare it to that of the original mesh $\boldsymbol{x}$ and the coefficients $1/a_n$. For each reparameterization scheme, we show the decay for the quality measure E with the number of reparameterization iterations in Figs. 5.3j to 5.3m. We observe that for scheme $a_n = n$, the spectrum for the reparameterized surface does not decay fast enough and therefore, the surface is not resolved accurately in regions of high curvature. The schemes with $a_n = n^{16}$ and $a_n = H(n - 2p/3)$ affect only the high-order components in the spherical harmonic expansion. In this case, while the surface is resolved accurately, the scheme does not fix the clustering of mesh points and this affects the accuracy for the singular integration scheme. The scheme $a_n = H(n - 2p/3)$ was used in [105]. While it worked well for low-order discretizations, our present study shows that it does not work well for high-order discretizations. Finally, the scheme with $a_n = n^2$ has small truncation error for the spherical harmonic expansion and also achieves high accuracy for singular integration. In the remainder of this chapter, we always use this reparameterization scheme.

5.3.6 Semi-Implicit Time-Stepping

At time t_n, we denote the membrane position by $\boldsymbol{x}^n$. To compute the updated surface position $\boldsymbol{x}^{n+1}$ at time t^{n+1}, we use the semi-implicit time-stepping scheme of [88]. Next, we briefly summarize this scheme.

The interfacial forces $\boldsymbol{f}_b(\boldsymbol{x})$ and $\boldsymbol{f}_\sigma(\boldsymbol{x}, \sigma)$ are defined at each point on the membrane γ. The bending and tension operators are defined as $\mathcal{S}[\boldsymbol{f}_b(\boldsymbol{x})]$ and $\mathcal{S}[\boldsymbol{f}_\sigma(\boldsymbol{x}, \sigma)]$ respectively. We linearized the bending and tension operators around $\boldsymbol{x}^n$ as follows,

$$B\boldsymbol{u} = \mathcal{S}_{\boldsymbol{x}^n}[\boldsymbol{f}_b(\boldsymbol{x}^n) + \boldsymbol{f}_b{}'(\boldsymbol{x}^n)\boldsymbol{u}\Delta t], \tag{5.36}$$

$$T\sigma = \mathcal{S}_{\boldsymbol{x}^n}[\boldsymbol{f}_\sigma(\boldsymbol{x}^n, \sigma)]. \tag{5.37}$$

The discrete spectral version of these operator are given by: $\widehat{B}\ \widehat{\boldsymbol{u}} = Y^p_{proj}\ B\ Y^p\ \widehat{\boldsymbol{u}}$ and $\widehat{T}\ \widehat{\sigma} = Y^p_{proj}\ T\ Y^p\ \widehat{\sigma}$. Similarly, we also define the following discrete spectral operators: $\widehat{D}\ \widehat{\boldsymbol{u}} = Y^p_{proj}\ \mathcal{D}_{\boldsymbol{x}^n}[Y^p\ \widehat{\boldsymbol{u}}]$ and $\widehat{\mathrm{div}}_\gamma\,(\widehat{\boldsymbol{u}}) = Y^p_{proj}\ \mathrm{div}_\gamma\,(Y^p\ \widehat{\boldsymbol{u}})$. The operators B, T and D are implemented using the scheme described in Section 5.3.2. The Galerkin formulation in Eq. (5.11)e-ves:vel-gal-3 is now discretized to give the globally semi-implicit time-stepping scheme,

$$\alpha_i \widehat{\boldsymbol{u}}_i = \widehat{\boldsymbol{u}}_i^\infty + \sum_{j=1}^{N_\gamma} \left(\widehat{B}_{ij}\widehat{\boldsymbol{u}}_j + \widehat{T}_{ij}\widehat{\sigma}_j + \widehat{D}_{ij}\widehat{\boldsymbol{u}}_j \right) \quad \text{for all } i = 1, \ldots, N_\gamma, \tag{5.38}$$

$$\widehat{\mathrm{div}}_\gamma\,\widehat{\boldsymbol{u}} = 0, \tag{5.39}$$

$$\boldsymbol{x}^{n+1} = \boldsymbol{x}^n + \boldsymbol{u}\Delta t. \tag{5.40}$$

The subscripts i and j in operators $\widehat{B}_{ij}$, $\widehat{T}_{ij}$ and $\widehat{D}_{ij}$ denote that the operators are applied to the j^{th} surface and the target points are on the i^{th} surface. We solve this linear system of equations for the tension σ, velocity $\boldsymbol{u}$ and the updated position $\boldsymbol{x}^{n+1}$.

5.3.7 Adaptive Time-Stepping

Our adaptive time-stepping scheme is based on the work of [85, 84]. Even though we have a first order time-stepping scheme, we present the algorithm for choosing the time step size for a general k^{th} order scheme. In each step of the simulation, we use an estimate of the error e_n incurred in the current iteration with time step size Δt_n to determine the optimal time step size Δt_{n+1} for the next step. The error e_n is estimated in one of the following two ways:

- We determine the new position $\hat{\boldsymbol{x}}_n$ and $\boldsymbol{x}_n$ using different numerical schemes. From the vesicle position $\boldsymbol{x}_{n-1}$ at time t_{n-1}, we compute $\hat{\boldsymbol{x}}_n$ by taking one time step of size Δt_n and compute $\boldsymbol{x}_n$ by taking two time steps of size $\Delta t_n/2$. Then, we defined the error estimate as $e_n \coloneqq \|\boldsymbol{x}_n - \hat{\boldsymbol{x}}_n\|_2$.

- We measure the change in invariant quantities such as the surface area A_n and volume V_n. We define the error estimate as $e_n \coloneqq \max(|\Delta A|/A, |\Delta V|/V)$.

In the first case, we have to perform extra computation to determine two numerical solutions for the position and this can be expensive. In the second case, it requires very little computation to determine the change in area and volume between time steps. While the second error estimate worked well in 2D, it is not robust enough 3D and we observed several instances where the method underestimated the error. Therefore, in this work we always use the first error estimate. In both cases, the total error also depends on other factors such as truncation errors, accuracy of the Stokes operator (FMM and quadratures) and the GMRES tolerance used to enforce inextensibility. For a k^{th} order time-stepping scheme, the error is observed to scale with Δt_n as $e_n = \mathcal{O}\left(\Delta t_n^{k+1} + \epsilon_{other}\, \Delta t_n\right)$. Therefore, have to ensure that ϵ_{other} is also small enough.

For a long time scale simulation with time-horizon T and an overall error tolerance $\mathcal{E}$, we present an adaptive time-stepping algorithm for determining the optimal time step size Δt_n for each iteration. In $(n+1)^{\text{th}}$ time step, we choose the step-size Δt_{n+1} such that the error e_{n+1} is as close as possible to the maximum allowed error but does not exceed this error. Therefore, we want $e_{n+1}/\Delta t_{n+1} = \beta\mathcal{E}/T$, where $\beta < 1$ is a safety factor. In our experiments, we choose $\beta = 0.9$.

We assume that for a k^{th} order time-stepping scheme, the error scales with Δt_n as $e_n = \mathcal{O}\left(\Delta t_n^{k+1}\right)$. We also assume that the constants in this order estimate do not change significantly between consecutive time steps. Then, we have, $\Delta t_{n+1}^{k+1}/e_{n+1} = \Delta t_n^{k+1}/e_n$. We substitute $e_{n+1}/\Delta t_{n+1} = \beta\mathcal{E}/T$ to obtain the new time step size Δt_{n+1},

$$\Delta t_{n+1} = \Delta t_n \left(\beta\, \frac{\mathcal{E}}{T}\, \frac{\Delta t_n}{e_n}\right)^{1/k}$$

Then, we compute the solution $\boldsymbol{x}_{n+1}$ at time $t_{n+1} = t_n + \Delta t_{n+1}$. We measure the error e_{n+1} and check if it satisfies the condition $e_{n+1}/\Delta t_{n+1} \leq \mathcal{E}/T$. If the condition is satisfied, we accept the solution and proceed to the next time step. If the condition is not satisfied, we reject the solution and update the time step size as follows,

$$\Delta t_{n+1} = \Delta t_{n+1} \left(\beta\, \frac{\mathcal{E}}{T}\, \frac{\Delta t_{n+1}}{e_{n+1}}\right)^{1/k}$$

We repeat the above process with this updated time step size.

5.3.8 Algorithm Summary and Computational Cost

We start a simulation with N_γ vesicles. The position and shape of the vesicles is defined by the spherical harmonic discretization $\widehat{\boldsymbol{x}}$ of degree p. For each surfaces, we provide the bending modulus, the excess density and the viscosity contrast of the fluid inside the vesicle compared to the fluid outside. In addition, we provide the following simulation parameters: the time-horizon T, the initial time step size Δt, the error tolerance for time-adaptivity $\mathcal{E}$, the tolerance for each GMRES solve ϵ_{GMRES}, the maximum reparameterization step size $\Delta\tau_{max}$, the distance parameter for repulsion R_{repul}, the initial quadrature q order for singular, near-singular and far-field integration, the boundary conditions (periodic with period length or free space) and the background velocity $\boldsymbol{u}^\infty$.

To compute the new surface position $\widehat{\boldsymbol{x}}^+$ after time Δt, we solve the linear system discussed in Section 5.3.6. To do this, we setup the RHS for the linear system and initialize the linear operator. Constructing the linear operator requires setting up the operator for linearized interfacial forces and the Stokes single and double-layer potentials. Detailed discussion of linearized interfacial forces can be found in [88, 105]. For the Stokes layer potentials, we perform the setup for each of the singular, near-singular and far-field integration algorithms discussed in Section 5.3.2. This involves: computing the singular integration matrices; identifying the pairs of vesicles and their near target points, computing projection of these target points on the vesicle surface; and constructing the octree for FMM. During the setup for near-singular integration, we also check for vesicle collision, compute the repulsion force and add it to the RHS. We also check the accuracy of the quadratures using the scheme described at the end of Section 5.3.2 to update the order of the quadratures. Then, we solve this linear system using GMRES to obtain the new position $\widehat{\boldsymbol{x}}^+$ and the surface tension $\widehat{\sigma}^+$. In each GMRES iteration, the linearized interfacial forces are computed and then the Stokes layer potential is evaluated.

For adaptive time-stepping, we perform three GMRES solves as discussed in Section 5.3.7 to estimate the solution error. If the error is smaller than $\mathcal{E}\Delta t/T$, then we accept the new solution and advance the time by Δt; otherwise, we reject the solution. Then, we update the time step size to be used in the next iteration.

We now apply the area and volume correction algorithm discussed in Section 5.3.4. We reparameterize the surface discretization as described in Section 5.3.5. We repeat the above process until $t = T$.

We summarize the computational cost associated with different stages of our algorithm in Table 5.2.

Setup	$T_{setup} =$	$N_{\text{Tstep}} \times \frac{N_\gamma}{n_p} p^3 (p^2 + q^2)$	$+$	(singular)
		$N_{\text{Tstep}} \times \frac{N_\gamma}{n_p} q^2 \left(\log \frac{N_\gamma}{n_p} q^2 + \log n_p \right)$	$+$	(near-singular)
		$N_{\text{Tstep}} \times \frac{N_\gamma}{n_p} q^2 \left(\log \frac{N_\gamma}{n_p} q^2 + \log n_p \right)$		(far-field)
Singular Integration	$T_{self} =$	$N_{\text{Tstep}} \times N_{\text{iter}} \times \frac{N_\gamma}{n_p} p^4$		
Near-singular Integration	$T_{near} =$	$N_{\text{Tstep}} \times N_{\text{iter}} \times \frac{N_\gamma}{n_p} p^2 q^2$		
Far-field Integration	$T_{near} =$	$N_{\text{Tstep}} \times N_{\text{iter}} \times \frac{N_\gamma}{n_p} (p^2 + q^2)$	$+$	(computation)
		$N_{\text{Tstep}} \times N_{\text{iter}} \times \left(\frac{N_\gamma}{n_p} q^2 \right)^{2/3} \log n_p$		(communication)
Reparameterization	$T_{repar} =$	$N_{\text{Tstep}} \times N_{repar} \times \frac{N_\gamma}{n_p} p^3$		

Table 5.2 *Time complexity for N_γ vesicles with p^{th} order surface discretization, q^{th} order quadratures (such that $p < q$), N_{Tstep} GMRES solves of the implicit time-stepping scheme with N_{iter} GMRES iterations per solve on n_p processors. We denote the average number of reparameterization iterations in each time step by N_{repar}.*

5.4 Results

We present some numerical results to show the accuracy and time-to-solution on a single node in Section 5.4.1 and strong and weak scalability of our method in Sections 5.4.2 and 5.4.3 respectively. All results are presented for the Stampede system at the Texas Advanced Computing Center (TACC). It is a Linux cluster consisting of 6,400 compute nodes connected by 56Gb/s FDR Mellanox InfiniBand network in a fat tree configuration. Each compute node has dual eight-core Intel Xeon E5-2680 CPUs running at 2.7GHz and 32GB of memory. In addition, most nodes have an Intel Xeon Phi SE10P co-processor, while a few have an NVIDIA K20 GPU co-processor; however, our current implementation cannot utilize these accelerators.

5.4.1 Single Node Results

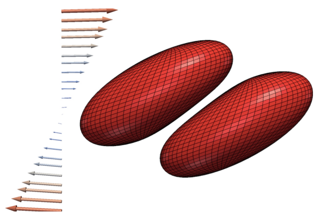
We present results for two vesicles in shear flow as shown in Figs. 5.4a to 5.4d. The fluid inside and outside the vesicles is identical, the vesicles have a reduced volume of 0.85 and bending modulus of 0.01 and the simulation has a time-horizon of $T = 160$. The experiment is designed to underline the significance of high-order spatial discretization, time-adaptivity, near-singular integration, reparameterization and collision handling. The simulation reveals significant inter-vesicle interactions and the vesicles undergo large deformations. The vesicles are closest at $t = 136$ (see Fig. 5.4c). Without repulsion, we required a discretization order of at least $p = 32$ to resolve this flow with sufficient accuracy to avoid collision between the vesicles. For lower orders, the spatial discretization errors cause the vesicles to intersect and this causes the simulation to break.

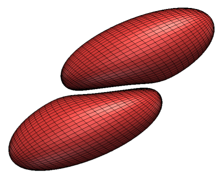
Convergence Analysis. We study dependence of the solution error on the discretization order (p), the tolerance for time-adaptivity ($\mathcal{E}$) and the repulsion distance (R_{repul}). All solutions are computed using the first order implicit time-stepping scheme and using the block-diagonal preconditioner. We use a fixed GMRES tolerance of $\epsilon_{\text{GMRES}} = 1\text{E-}7$ for the implicit solver. All boundary integrals are computed using $2\times$ upsampling of the mesh; i.e. the quadrature order for singular, near-singular and far-field integration is $q = 2p$ for p^{th} order surface discretization. We reparameterize using the attenuation coefficients $a_n = n^2$ with reparameterization time step size $\Delta\tau_{\max} = 1\text{E-}4$ and reparameterization termination condition $\Delta\tau < 1\text{E-}5$.

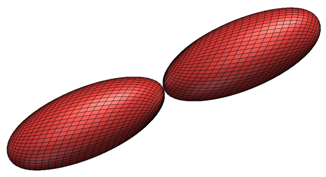
Reference Solution: We construct a reference solution without repulsion ($R_{repul} = 0$), using adaptive time-stepping with $\mathcal{E} = 0.1$ and with discretization order $p = 32$. Computing the reference solution required about two days of compute time on a single node of Stampede. Attaining similar accuracy without time-adaptivity would be $5\times$ more expensive.

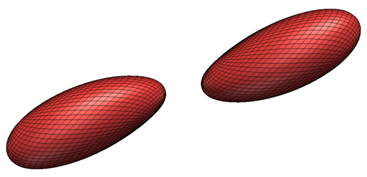
Time-Adaptivity and Relation Between $\mathcal{E}$ and Δt: In Fig. 5.4e, for fixed $p = 32$ and no repulsion, we plot the time step size Δt during the second half of the simulation ($80 \leq t \leq 160$). We vary the tolerance for time-adaptivity ($\mathcal{E}$) and observe an approximately linear relationship between Δt and $\mathcal{E}$ due to the first order semi-implicit time-stepping scheme.

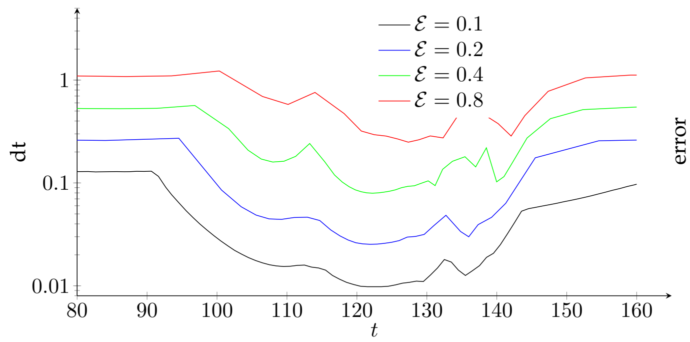
Effect of Repulsion Distance R_{repul}: In Fig. 5.4f, we introduce repulsion between the surfaces and plot the error in the vesicle position (compared with the reference solution) at the end of the simulation as a function of the repulsion distance (R_{repul}) for different discretization orders and time-adaptivity tolerance $\mathcal{E} = 0.2$. Due to reparameterization, we cannot directly compare the position of the surface grid points; therefore, we compare the surface center of mass $c_i = \int_{\gamma_i} \mathrm{d}\gamma_i$ for the i^{th} vesicle. We report the maximum error in the position for any vesicle at the end of the simulation and this error is normalized by the vesicle length. We observe that for $p = 32$, adding repulsion does not cause any noticeable increase in error for $R_{repul} < 0.04$. This is because the range of the repulsive force is smaller than the separation between the vesicles. We observe a steady increases in error with R_{repul}


(a) $t = 90$

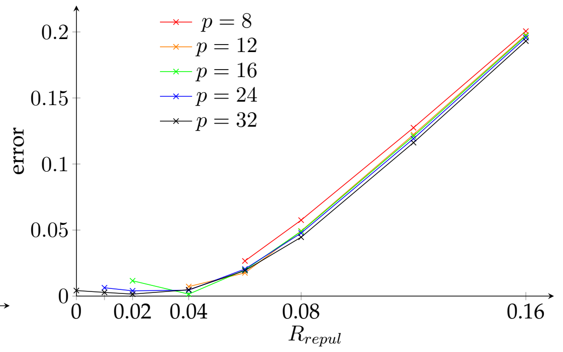

(b) $t = 120$


(c) $t = 136$

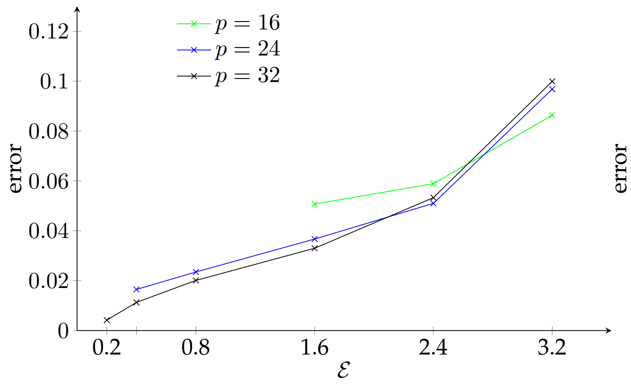

(d) $t = 145$



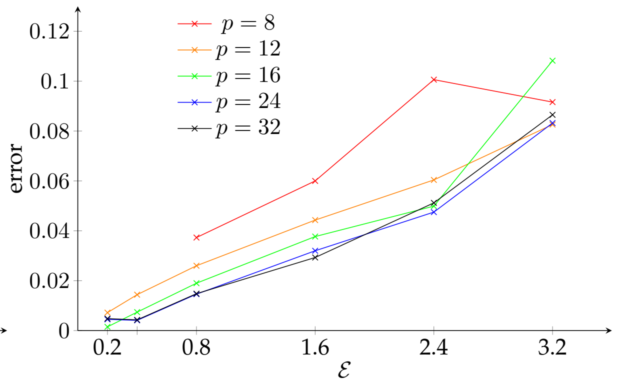
(e) $p = 32$, $R_{repul} = 0$



(f) $\mathcal{E} = 0.2$



(g) $R_{repul} = 0$



(h) $R_{repul} = 0.04$

Figure 5.4 *Convergence results for two vesicles in shear flow. The reference solution shown in figures Figs. 5.4a to 5.4d is computed using discretization order $p = 32$, time-adaptivity error tolerance $\mathcal{E} = 0.1$ and no repulsion force. In Fig. 5.4e, we show the time step size Δt at different points during the simulation for the reference solution ($\mathcal{E} = 0.1$) and also larger values of $\mathcal{E}$ while keeping $p = 32$ fixed. The first order behavior of the time-stepping scheme can be observed. We also observe a smaller time step size in regions where vesicle-vesicle interactions become significant. Fig. 5.4f shows the dependence of error in vesicle position (compared to the reference solution) when a short range repulsion force in added between the vesicles. The repulsion force allows us to use lower order discretizations and does not introduce significant errors for $R_{repul} \leq 0.04$. For different discretization orders, we show the linear dependence of the vesicle position error with the tolerance for time-adaptivity $\mathcal{E}$ in Figs. 5.4g and 5.4h for the cases without repulsion and with repulsion ($R_{repul} = 0.04$) respectively.*

for $R_{repul} \geq 0.04$. By adding repulsion, we are also able to compute solutions with lower order spatial discretizations. In general, the repulsion distance should be at least of the order of the distance between the grid points so that the repulsion force can be resolved on the vesicle surface. For $R_{repul} = 0.04$, we can compute solutions with $p \geq 12$ and this does not appear to significantly affect the solution accuracy.

Relation Between $\mathcal{E}$ and Position Error: In Figs. 5.4g and 5.4h, we plot the error in vesicle position as a function of the time-adaptivity tolerance $\mathcal{E}$ for $R_{repul} = 0$ and $R_{repul} = 0.04$ respectively. In both cases we observe an approximately linear relationship between the position error and $\mathcal{E}$.

p	$\mathcal{E}$	R_{repul}	*error*	N_{Tstep}	N_{iter}	T_{solve}	T_{setup}	T_{self}	T_{near}	T_{far}	T_{repar}
32	0.2	0.0	4.1E-3	3270	34	64321	23592	2724	6684	22925	4208
32	0.4	0.0	1.1E-2	1116	40	25864	8184	1104	2417	9228	3285
32	0.8	0.0	2.0E-2	441	50	13087	3316	558	1007	4500	2907
32	1.6	0.0	3.3E-2	216	67	8902	1670	363	572	2966	2802
32	2.4	0.0	5.4E-2	153	81	8828	1219	310	471	2522	3844
32	0.2	0.0	4.1E-3	3270	34	64321	23592	2724	6684	22925	4208
24	0.4	1E-2	1.3E-2	1191	39	8474	1933	262	954	3283	1157
16	0.8	2E-2	2.8E-2	513	36	1091	127	29	97	358	263
12	1.6	4E-2	4.3E-2	228	45	333	21	8	21	100	90
8	2.4	6E-2	4.3E-2	174	27	106	5	2	4	30	38

Table 5.3 *We present convergence results for the shear flow problem visualized in Figs. 5.4a to 5.4d. For different values of error tolerance $\mathcal{E}$, we report the error in vesicle position compared to a reference solution computed with $\mathcal{E} = 0.1$, $p = 32$ and no repulsion. We also report the number of solves of the implicit time-stepping scheme N_{Tstep}, the average number of GMRES iterations needed for each solve N_{iter} and the time-to-solution T_{solve}. In addition, we report a breakdown of the time spent in different stages of the algorithm: the cost for reparameterization T_{repar}, the cost for singular T_{self}, near-singular T_{near} and far-field T_{far} interactions and their setup cost T_{setup}. We compare results for high-order discretization and no repulsion with low order discretizations which are made possible due to introduction of a repulsion term. For about 5% error in vesicle position, using low order discretization ($p = 8$) is over $80\times$ faster when compared with a solution computed using $p = 32$.*

Timing Results. In Table 5.3, we present detailed results for the above shear flow test case. In addition to the error in vesicle position, we also report the number of solves of the implicit time-stepping scheme N_{Tstep}, the average number of GMRES iterations N_{iter}, the overall solve time T_{solve} and a detailed breakdown of the time spent in different stages of the algorithm. We report two sets of results.

Fixed Spatial Discretization Order $p = 32$: In the first set, we report results for a fixed discretization order $p = 32$, no repulsion and varying tolerance values for time-adaptivity $\mathcal{E}$. Due to the first order time-stepping scheme, we observe an approximately linear relationship between the number of time steps N_{Tstep} and inverse of the error tolerance $1/\mathcal{E}$. At the same time, we observe that with increasing time step size (due to increasing $\mathcal{E}$), the number of GMRES iterations per time step increase significantly. This happens because the semi-implicit scheme becomes more ill-conditioned. Therefore, we do not observe the expected speedup in T_{solve} with increasing $\mathcal{E}$. The interactions between vesicles computed through single-layer and double-layer Stokes kernel functions are evaluated at every GMRES iteration while the setup phase is execute once for each time step. Therefore, for a fixed discretization order p, we observe that T_{setup} scales as $\mathcal{O}(N_{\text{Tstep}})$ while T_{self}, T_{near} and T_{far} scale as $\mathcal{O}(N_{\text{Tstep}} N_{\text{iter}})$. We also note that the reparameterization time T_{repar} shows little variation with $\mathcal{E}$. This is because for larger $\mathcal{E}$ even though we reparameterize fewer times (since N_{Tstep} is smaller), a larger time step size means that we require many more reparameterization iterations.

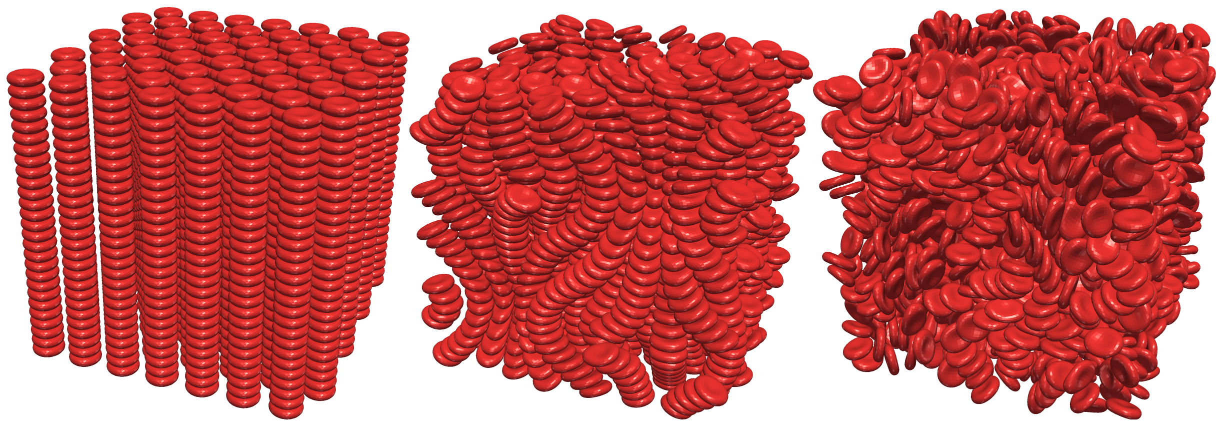
Optimal Spatial Discretization Order: In the second set of results in Table 5.3, we add a short range repulsion force between the surfaces to prevent collision between them. This allows us to use lower order discretizations and obtain a faster time to solution. Comparing these results with high-order discretizations, we observe that the error in vesicle position and number of time steps N_{Tstep} remain the same for the same value of $\mathcal{E}$. However, the number of GMRES iterations are much smaller for low order discretizations. We believe this is due to the smaller size of the linear system being solved. In addition, each GMRES iteration has drastically lower computational cost for smaller discretization orders p since we have $\mathcal{O}\left(N_\gamma p^4\right)$ cost for singular and near-singular integration and $\mathcal{O}\left(N_\gamma p^2\right)$ cost for far-field interactions. The setup stage (with $\mathcal{O}\left(N_\gamma p^5\right)$ cost for computing the singular integration operator) and the reparameterization algorithm (with $\mathcal{O}\left(N_\gamma p^3\right)$ cost per iteration) are also significantly less expensive for smaller p. For the same solution accuracy, we observe nearly two orders of magnitude speedup when using low order discretizations with repulsion compared to high-order discretizations without repulsion.

5.4.2 Strong Scaling

In this section, we present strong scaling results for the periodic Taylor-vortex flow simulation shown in Fig. 5.5. We used the following background velocity field

$$\begin{aligned} \boldsymbol{u}^{\infty}(x,y,z) \quad &= \quad \alpha \sin\left(\frac{2\pi x}{L}\right)\cos\left(\frac{2\pi y}{L}\right)\sin\left(\frac{2\pi z}{L}\right)\hat{i} \quad + \\ & \qquad \alpha \cos\left(\frac{2\pi x}{L}\right)\sin\left(\frac{2\pi y}{L}\right)\sin\left(\frac{2\pi z}{L}\right)\hat{j} \end{aligned} \tag{5.41}$$

where, $L = 17$ is the period length of the domain and $\alpha = 0.1$ is a scaling factor. The domain has 1408 biconcave shaped vesicles with 35% volume fraction. Each vesicle has an approximate diameter of 1.89, a height of 0.54, a reduced volume of 0.65 and a bending modulus of 0.1. For this simulation, we used 16th order spherical harmonic discretization with 50th order quadratures for singular integration and 24th order quadratures for near-singular and far-field integration for about 5-digits of accuracy. We used our adaptive time-stepping scheme with an error tolerance of $\mathcal{E} = 0.02$ for time-horizon $T = 2$. The linear system for the semi-implicit time-stepping scheme was solved using GMRES with a relative tolerance of 1E-5. For time-horizon $T = 2$, we needed 18 GMRES solves and each solve required an average of $N_{\text{iter}} = 139$ iterations for discretization order $p = 16$ and $N_{\text{iter}} = 311$ iterations for $p = 32$.



(a) $t = 0$ **(b)** $t = 125$ **(c)** $t = 500$

Figure 5.5 *A simulation of* 1408 *vesicles in a periodic Taylor-vortex flow. The vesicles have a volume fraction of* 35% *and each vesicle has a biconcave shape with a reduced volume of* 0.65. *For this simulation, we used 16th order discretization with 50th order quadratures for singular integration and 24th order quadratures for near-singular and far-field integration. We used adaptive time-stepping with error-factor* $\mathcal{E}/T = 0.01$ *and with a tolerance of* 1E-5 *for the GMRES solve.*

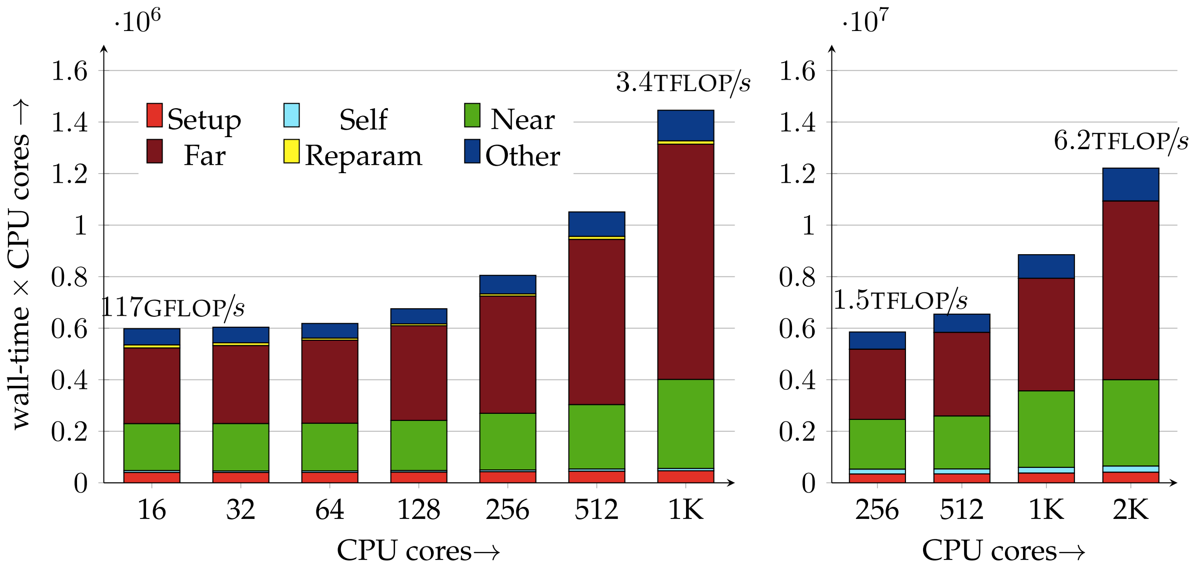
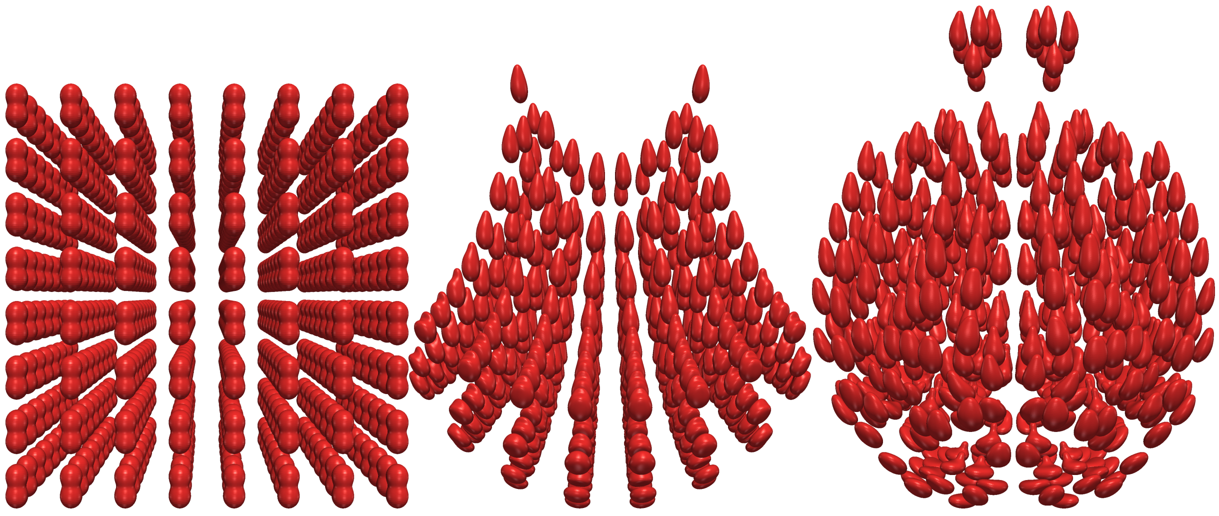


Figure 5.6 *Strong scalability results for the periodic Taylor-vortex flow in Fig. 5.5. We present results for discretization order $p = 16$ on the left and $p = 32$ on the right. In both cases, we used 50th order quadratures for singular integration and 24th order quadratures for near-singular and far-field integration. We solved the problem for a time-horizon $T = 2$ and the adaptive time-stepping scheme required* 18 *GMRES solves. On average, each GMRES solve requires $N_{\text{iter}} = 138$ iterations for $p = 16$ and $N_{\text{iter}} = 311$ iterations for $p = 32$.*

In Fig. 5.6, we report the total CPU time (wall-time×CPU cores). For $p = 16$ (figure on the left), we scale from 16 CPU cores (1 compute node) to 1024 cores (64 compute nodes). For $p = 32$ (figure on the right), $4\times$ more memory is required for storing the singular integration operators and therefore we start from 256 CPU cores (16 compute node) and scale up to 2048 cores (128 compute nodes). For $p = 16$, we achieve a $26.5\times$ speedup in the total wall-time or 41.4% strong scaling efficiency and for $p = 32$, we achieve a $3.8\times$ speedup in the total wall-time or 47.9% strong scaling efficiency. Overall, we observe that the case with $p = 32$ is about $10\times$ more expensive that $p = 16$.

We also provide a breakdown of the time spent in the different stages of our algorithm. The cost of the setup stage is dominated by the computation of singular integration matrix for each surface. Since the setup is performed only once for every N_{iter} evaluations, the setup cost is dwarfed by the evaluation cost (singular, near-singular and far-field integration) and requires just $3\% \sim 6\%$ of the runtime. The singular integration requires very little work (about 1% for $p = 16$ and 3% for $p = 32$ of runtime) due to our $\mathcal{O}\left(p^4\right)$ scheme. Due to the dense packing of the vesicles, we need to compute near-singular integration for a large number of target points. Therefore, near-singular integration is relatively expensive and requires about $23\% \sim 33\%$ of the CPU time; however, it scales well since it is compute bound and requires very little communication. The far-field computation is the most expensive stage in our scheme and requires $46\% \sim 63\%$ of the total CPU time. It is implemented using FMM and gives good performance up to 256 cores for $p = 16$ and up to 1024 cores for $p = 32$. As we increase the number of CPU cores further, the problem size per core is too small to remain efficient and we begin to lose performance. The reparameterization does not require any communication and is inexpensive compared to the overall solve time. The remaining time (about $8\% \sim 11\%$) is mostly spent inside the GMRES solve in PETSc.

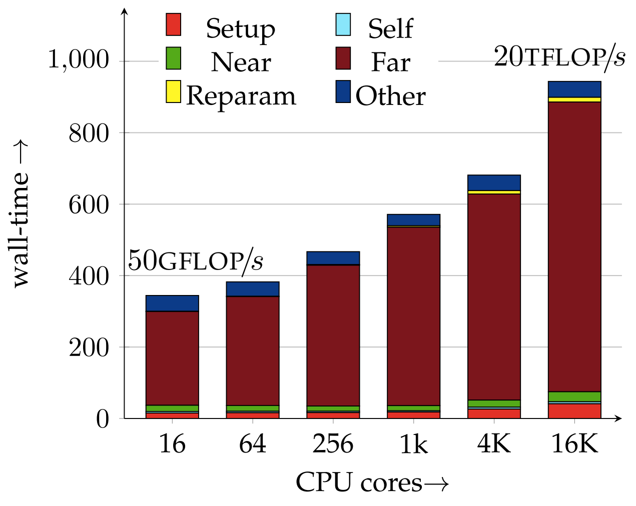


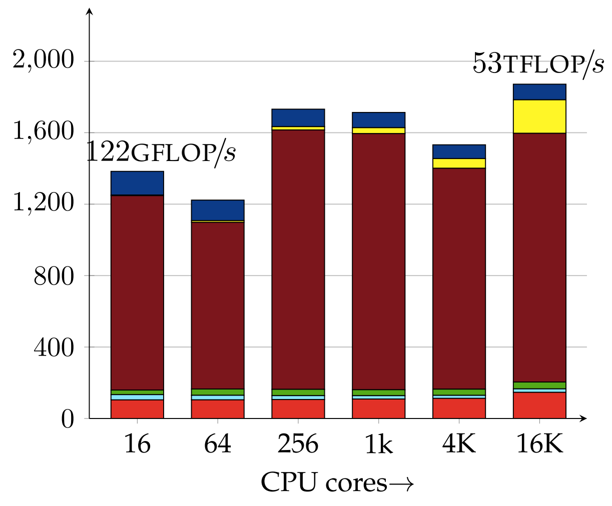
(a) $t = 0$ **(b)** $t = 4$ **(c)** $t = 30$

Figure 5.7 *A simulation showing sedimentation of vesicles under gravitational force. We start with* 512 *vesicles arranged in an* $8 \times 8 \times 8$ *lattice at where, each vesicle has a reduced volume of* 0.85, *bending modulus of* 0.05 *and has excess density* $\rho_i - \rho = 1.0$ *(and gravitational acceleration* $g = 1.0$*). We used our adaptive time-stepping scheme with error factor* $\mathcal{E}/T = 0.02$ *and a spatial discretization order* $p = 16$.

5.4.3 Weak Scaling

In Fig. 5.8, we present weak scaling results for a polydisperse sedimentation flow on $16K$ CPU cores. We used 16th order discretization, a fixed step-size of $\Delta t = 0.01$ and a time-horizon $T = 0.2$. For the semi-implicit scheme, we used a GMRES tolerance of 1E-5 and the average number of GMRES iterations varied from 64 iterations for the smallest problem size to 25 iterations for the largest problem size. We present two sets of results for different





cores	N_γ	N_{iter}	T_{solve}	GFLOP/s
16	16	64	344	50
64	64	59	382	204
256	256	52	466	766
1024	1024	44	571	2675
4096	4096	37	681	6246
16384	16384	31	943	20173

cores	N_γ	N_{iter}	T_{solve}	GFLOP/s
16	112	57	1384	122
64	448	48	1223	404
256	1792	41	1732	1350
1024	7168	34	1713	4832
4096	28672	30	1532	15206
16384	114688	25	1872	53350

Figure 5.8 *Weak scalability results for polydisperse sedimentation similar to the flow visualized in Fig. 5.7 on* $16K$ *CPU cores. We compute solutions for a time-horizon* $T = 0.2$ *using a fixed time step size* $\Delta t = 0.01$*. We used discretization order* $p = 16$ *and singular, near-singular and far-field quadratures of order* $q = 28$*. The vesicles have a reduced volume of* 0.85*, bending modulus in the range* $[0.05, 0.1]$*, viscosity contrast in the range* $[0.5, 5]$ *and an excess density of* 1*. We show results for two different problem sizes: on the left for* 1 *vesicle per core and on the right for* 7 *vesicles per core. For each case, we present a bar graph showing the breakdown of the solve time into each of the different stages of the algorithm. We also report the average number of GMRES iteration* N_{iter} *and the overall performance in* GFLOP/s *for each test case.*

grain sizes. In the first case (Fig. 5.8:left), we have one vesicle per CPU core. We achieve 50GFLOP/s on 16 cores and 20TFLOP/s on $16K$ cores; i.e. $400\times$ increase in performance for $1024\times$ increase in the number of processors. In the second case (Fig. 5.8:right), we have seven vesicle per CPU core. We achieve 122GFLOP/s on 16 cores and 53TFLOP/s on $16K$ cores; i.e. $437\times$ increase performance for $1024\times$ increase in the number of processors.

We have presented a detailed breakdown of the time in different stages of the algorithm as we scale from 16 cores to $16K$ cores. The solve time is dominated by the far-field integration stage which accounts for $74\% \sim 86\%$ of the total time. Unlike the Taylor-Green vortex flow discussed in Section 5.4.2, for sedimentation flow, the near-singular integration requires very little work (about $2\% \sim 5\%$ of T_{solve}). This is due to the relatively lower density of vesicles resulting in fewer near-singular interactions between vesicles. The setup and self interaction stages requires about $4\% \sim 8\%$ and $1\% \sim 2\%$ of the solve time respectively. The reparameterization time T_{repar} ranges from $0.3\% \sim 10\%$ of the total time.

5.5 Conclusions

We have presented new algorithms for fast simulation of vesicle in a Stokesian fluid. These include efficient singular, near-singular and far-field integration algorithms for computing single- and double-layer Stokes potentials. We have developed algorithms for detecting collision between vesicles and to handle such collisions in a robust way by adding a repulsion force between the surfaces. We have also developed an algorithm for correcting the area and volume of vesicles in long-time scale simulations. We have analyzed our surface reparameterization algorithm to determine the optimal choice of the quality measure function. We have implemented an adaptive time-stepping scheme which allows us to choose the optimal time step size. This has significantly improved time-to-solution while still achieving the desired error tolerance. These algorithms have enabled us to compute accurate, long-time scale simulations for flows with high volume fraction of vesicles.

We have presented convergence studies to show first order convergence in the time step size. We also analyzed the effect of our collision handling scheme (using repulsion force) on the solution accuracy. Our algorithms are extremely efficient and show good strong and weak scalability on distributed memory architectures.

6 Conclusions

6.1 Summary of Contributions

Particle and Volume Potentials. We have described our implementation of the PVFMM software library for computing particle and volume potentials. It can be used to compute pairwise interactions in N-body problems, accelerate boundary integral formulations and construct solutions of elliptic PDEs with a range of boundary conditions on the unit cube. We have developed novel performance optimizations and new parallel algorithms which allow us to achieve high performance and scalability on large high performance computing systems. Our library is many times faster compared to other particle N-body codes and PDE solvers on cubic domains.

Variable Coefficient PDEs. We have developed volume integral equation (VIE) formulations for variable coefficient elliptic PDEs. We developed an efficient solver for such formulations by using PVFMM to compute volume integrals and GMRES to solve the discretized system. We have applied this formulation to simulate incompressible Stokes flow in highly porous media geometries. We have demonstrated that our method achieves high performance and scalability up to 2K compute nodes on the Stampede system (TACC).

Furthermore, we have developed novel VIE formulations for Poisson and Stokes equations under coordinate transformations. This formulation maps problems on certain non-regular geometries to cubic domains which can then be solved using our VIE solver. We have demonstrated the efficacy of this scheme for incompressible Stokes flow on highly irregular geometries.

Concentrated Vesicle Flows. We have presented a boundary integral equation formulation for simulating the dynamics of concentrated vesicle suspensions in a Stokesian fluid. We have developed an efficient and scalable solver for this formulation. Our method uses special quadratures to compute singular and near-singular boundary integral efficiently and uses our particle FMM to accelerate the far-field interactions. We have also developed an adaptive time-stepping scheme, a repulsion based scheme for handling vesicle collisions and algorithms for surface re-meshing. We performed long time scale simulations of flows with high volume-fraction of vesicles in parallel. Such simulations are useful in studying the rheology of human blood and other complex biofluids.

6.2 Future Work

Here we have only scratched the surface of what is possible with such integral equation formulations. Our methods can be extended to many other problems in electrostatics, elasticity and electromagnetic and acoustic scattering. There are many applications in science and engineering which can benefit from these state-of-the-art solvers.

In the future, we plan to integrate our solver for vesicle flows with our VIE solver to simulate vesicle flow in confined geometries such as blood capillaries and microfluidic devices. Such simulations will lead to better understanding of hematological disorders and designing of new microfluidic devices for blood fractionation, which can help in early diagnosis of certain cancers by detecting circulating tumor cells.

An efficient semi-Lagrangian advection diffusion solve based on our PVFMM framework was developed in [7]. This can be coupled with our solver for vesicle flows for studying transport phenomenon in complex fluids.

One of the remaining challenges is the development of solvers for problems with large variations in coefficients. Fast direct solvers can be used for static problems where a compressed representation of the solution operator is precomputed and then applied in linear time. However, the precomputation is expensive and cannot be used for time-varying problems. A possible solution for such problems is to use our volume integral equation solver for computing the potential at each time step. This approach can be used for complex fluids and multiphase flows. We plan to work on approximate preconditioners to accelerate the solution of integral equations for problems with large variations in coefficients.

Bibliography

[1] David Abreu, Michael Levant, Victor Steinberg, and Udo Seifert. Fluid vesicles in flow. *Advances in Colloid and Interface Science*, 208:129–141, jun 2014.

[2] Ludvig af Klinteberg and Anna-Karin Tornberg. A fast integral equation method for solid particles in viscous flow using quadrature by expansion. *Journal of Computational Physics*, 326:420–445, dec 2016.

[3] M B Amar and F C Farnoux. Numerical solution of the lippmann-schwinger equations in photoemission: application to xenon. *Journal of Physics B: Atomic and Molecular Physics*, 16(13):2339–2358, jul 1983.

[4] Philippe Angot, Charles-Henri Bruneau, and Pierre Fabrie. A penalization method to take into account obstacles in incompressible viscous flows. *Numerische Mathematik*, 81(4):497–520, feb 1999.

[5] T. Askham and A.J. Cerfon. An adaptive fast multipole accelerated poisson solver for complex geometries. *Journal of Computational Physics*, 344:1–22, sep 2017.

[6] K. E. Atkinson. The numerical solution of laplace's equation in three dimensions—II. In *Numerical Treatment of Integral Equations / Numerische Behandlung von Integralgleichungen*, pages 1–23. Birkhäuser Basel, 1980.

[7] Arash Bakhtiari, Dhairya Malhotra, Amir Raoofy, Miriam Mehl, Hans-Joachim Bungartz, and George Biros. A parallel arbitrary-order accurate AMR algorithm for the scalar advection-diffusion equation. In *SC16: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, nov 2016.

[8] Satish Balay, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Karl Rupp, Barry F. Smith, and Hong Zhang. PETSc Web page. http://www.mcs.anl.gov/petsc, 2014.

[9] W. Bangerth, R. Hartmann, and G. Kanschat. deal.II—a general-purpose object-oriented finite element library. *ACM Transactions on Mathematical Software*, 33(4):24–es, aug 2007.

[10] Wolfgang Bangerth, Denis Davydov, Timo Heister, Luca Heltai, Guido Kanschat, Martin Kronbichler, Matthias Maier, Bruno Turcksin, and David Wells. The deal.II library, version 8.4. *Journal of Numerical Mathematics*, 24(3), jan 2016.

[11] Josh Barnes and Piet Hut. A hierarchical o(n log n) force-calculation algorithm. *Nature*, 324(6096):446–449, dec 1986.

[12] Michele Benzi, Gene H. Golub, and JÃűrg Liesen. Numerical solution of saddle point problems. *Acta Numerica*, 14:1–137, may 2005.

[13] George Biros, Lexing Ying, and Denis Zorin. A fast solver for the stokes equations with distributed forces in complex geometries. *Journal of Computational Physics*, 193(1):317–348, jan 2004.

[14] J. R. Blake. A note on the image system for a stokeslet in a no-slip boundary. *Mathematical Proceedings of the Cambridge Philosophical Society*, 70(02):303, sep 1971.

[15] Jed Brown. Efficient nonlinear solvers for nodal high-order finite elements in 3d. *Journal of Scientific Computing*, 45(1-3):48–63, jul 2010.

[16] Carsten Burstedde, Omar Ghattas, Georg Stadler, Tiankai Tu, and Lucas C. Wilcox. Parallel scalable adjoint-based adaptive solution of variable-viscosity stokes flow problems. *Computer Methods in Applied Mechanics and Engineering*, 198(21-26):1691–1700, may 2009.

[17] Carsten Burstedde, Omar Ghattas, Michael Gurnis, Tobin Isaac, Georg Stadler, Tim Warburton, and Lucas Wilcox. Extreme-scale AMR. In *2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, nov 2010.

[18] C. Burstedde, G. Stadler, L. Alisic, L. C. Wilcox, E. Tan, M. Gurnis, and O. Ghattas. Large-scale adaptive mantle convection simulation. *Geophysical Journal International*, 192(3):889–906, jan 2013.

[19] Xiao-Chuan Cai and Olof B. Widlund. Domain decomposition algorithms for indefinite elliptic problems. *SIAM Journal on Scientific and Statistical Computing*, 13(1):243–258, jan 1992.

[20] Dominic D.J. Chandar, Jayanarayanan Sitaraman, and Dimitri J. Mavriplis. A GPU-based incompressible navier–stokes solver on moving overset grids. *International Journal of Computational Fluid Dynamics*, 27(6-7):268–282, jul 2013.

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

[22] Aparna Chandramowlishwaran, Samuel Williams, Leonid Oliker, Ilya Lashuk, George Biros, and Richard Vuduc. Optimizing and tuning the fast multipole method for state-of-the-art multicore architectures. In *2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*. IEEE, 2010.

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

[24] David Colton and Rainer Kress. *Inverse Acoustic and Electromagnetic Scattering Theory*. Springer Berlin Heidelberg, 1998.

[25] Eduardo Corona, Leslie Greengard, Manas Rachh, and Shravan Veerapaneni. An integral equation formulation for rigid bodies in stokes flow in three dimensions. *Journal of Computational Physics*, 332:504–519, mar 2017.

[26] G. H. Cottet and Leslie Greengard. The rapid evaluation of potential fields in particle systems. *Mathematics of Computation*, 52(186):716, apr 1989.

[27] O. Coulaud, P. Fortin, and J. Roman. High performance BLAS formulation of the multipole-to-local operator in the fast multipole method. *Journal of Computational Physics*, 227(3):1836–1862, jan 2008.

[28] M. C. Delfour and J. P. Zolésio. *Shapes and Geometries: metrics, analysis, differential calculus, and optimization*. Society for Industrial and Applied Mathematics, jan 2011.

[29] Michael G. Duffy. Quadrature over a pyramid or cube of integrands with a singularity at a vertex. *SIAM Journal on Numerical Analysis*, 19(6):1260–1262, dec 1982.

[30] J.W. Eastwood, R.W. Hockney, and D.N. Lawrence. P3m3dp—the three-dimensional periodic particle-particle/ particle-mesh program. *Computer Physics Communications*, 19(2):215–261, apr 1980.

[31] Jonas Englund and Johan Helsing. A comparison of splittings and integral equation solvers for a nonseparable elliptic equation. *BIT Numerical Mathematics*, 44(4):675–697, dec 2004.

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

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

[34] Lawrence Evans. *Partial Differential Equations*. American Mathematical Society, mar 2010.

[35] G I Fann, R J Harrison, G Beylkin, J Jia, R Hartman-Baker, W A Shelton, and S Sugiki. MADNESS applied to density functional theory in chemistry and nuclear physics. *Journal of Physics: Conference Series*, 78:012018, jul 2007.

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

[37] Jonathan B. Freund. Leukocyte margination in a model microvessel. *Physics of Fluids*, 19(2):023301, feb 2007.

[38] 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, feb 2000.

[39] Yuhong Fu, Kenneth J. Klimkowski, Gregory J. Rodin, Emery Berger, James C. Browne, JÃijrgen K. Singer, Robert A. Van De Geijn, and Kumar S. Vemaganti. A fast solution method for three-dimensional many-particle problems of linear elasticity. *International Journal for Numerical Methods in Engineering*, 42(7):1215–1229, aug 1998.

[40] Amir Gholami, Dhairya Malhotra, Hari Sundar, and George Biros. FFT, FMM, or multigrid? a comparative study of state-of-the-art poisson solvers for uniform and

nonuniform grids in the unit cube. *SIAM Journal on Scientific Computing*, 38(3):C280–C306, jan 2016.

[41] David Gilbarg and Neil S. Trudinger. *Elliptic Partial Differential Equations of Second Order*. Springer Berlin Heidelberg, 1983.

[42] Zydrunas Gimbutas and Leslie Greengard. FMMLIB3D 1.2, 2012. http://www.cims.nyu.edu/cmcl/fmm3dlib/fmm3dlib.html.

[43] Z. Gimbutas and L. Greengard. A fast and stable method for rotating spherical harmonic expansions. *Journal of Computational Physics*, 228(16):5621–5627, sep 2009.

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

[45] Zydrunas Gimbutas and Shravan Veerapaneni. A fast algorithm for spherical grid rotations and its application to singular quadrature. *SIAM Journal on Scientific Computing*, 35(6):A2738–A2751, jan 2013.

[46] Z. Gimbutas, L. Greengard, and S. Veerapaneni. Simple and efficient representations for the fundamental solutions of stokes flow in a half-space. *Journal of Fluid Mechanics*, 776, jul 2015.

[47] I.G. Graham and I.H. Sloan. Fully discrete spectral boundary integral methods for helmholtz problems on smooth closed surfaces in $\mathbb{R}^3$. *Numerische Mathematik*, 92(2):289–323, aug 2002.

[48] Ananth Grama, George Karypis, Vipin Kumar, and Anshul Gupta. *An Introduction to Parallel Computing: Design and Analysis of Algorithms*. Addison-Wesley, second edition, 2003.

[49] 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, aug 2002.

[50] Leslie Greengard and Mary Catherine Kropinski. An integral equation approach to the incompressible navier–stokes equations in two dimensions. *SIAM Journal on Scientific Computing*, 20(1):318–336, jan 1998.

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

[52] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 135(2):280–292, aug 1997.

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

[54] Pierre Grisvard. *Elliptic Problems in Nonsmooth Domains*. Society for Industrial and Applied Mathematics, jan 2011.

[55] Max D. Gunzburger. *Finite Element Methods for Viscous Incompressible Flows*. Elsevier, 1989.

[56] Tsuyoshi Hamada, Tetsu Narumi, Rio Yokota, Kenji Yasuoka, Keigo Nitadori, and Makoto Taiji. 42 TFlops hierarchicalN-body simulations on GPUs with applications in both astrophysics and turbulence. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis - SC '09*. ACM Press, 2009.

[57] Robert J. Harrison, George I. Fann, Takeshi Yanai, Zhengting Gan, and Gregory Beylkin. Multiresolution quantum chemistry: Basic theory and initial applications. *The Journal of Chemical Physics*, 121(23):11587–11598, dec 2004.

[58] T.Y. Hou, J.S. Lowengrub, and M.J. Shelley. Boundary integral methods for multicomponent fluids and multiphase materials. *Journal of Computational Physics*, 169(2):302–362, may 2001.

[59] 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 on - SC '11*. ACM Press, 2011.

[60] Pritish Jetley, Lukasz Wesolowski, Filippo Gioachin, Laxmikant V. Kalé, and Thomas R. Quinn. Scaling hierarchical n-body simulations on GPU clusters. In *2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, nov 2010.

[61] C. T. Kelley and David E. Keyes. Convergence analysis of pseudo-transient continuation. *SIAM Journal on Numerical Analysis*, 35(2):508–523, apr 1998.

[62] Andreas KlÃűckner, Alexander Barnett, Leslie Greengard, and Michael O'Neil. Quadrature by expansion: A new method for the evaluation of layer potentials. *Journal of Computational Physics*, 252:332–349, nov 2013.

[63] Rainer Kress. Newton s method for inverse obstacle scattering meets the method of least squares. *Inverse Problems*, 19(6):S91–S104, nov 2003.

[64] Martin Kronbichler, Timo Heister, and Wolfgang Bangerth. High accuracy mantle convection simulation through modern numerical methods. *Geophysical Journal International*, 191(1):12–29, aug 2012.

[65] 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, aug 2011.

[66] Matthew Harper Langston. *An Adaptive Fast Multipole Method-based Pde Solver in Three Dimensions*. PhD thesis, New York University, New York, NY, USA, 2012. AAI3502706.

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

[68] 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, sep 2001.

[69] N. Liron and S. Mochon. Stokes flow for a stokeslet between two parallel flat plates. *Journal of Engineering Mathematics*, 10(4):287–303, oct 1976.

[70] Yijun Liu. *Fast Multipole Boundary Element Method*. Cambridge University Press, 2009.

[71] M. Loewenberg and E. J. Hinch. Numerical simulation of a concentrated emulsion in shear flow. *Journal of Fluid Mechanics*, 321(-1):395, aug 1996.

[72] Michael Loewenberg. Numerical simulation of concentrated emulsion flows. *Journal of Fluids Engineering*, 120(4):824, 1998.

[73] James W. Lottes and Paul F. Fischer. Hybrid multigrid/schwarz algorithms for the spectral element method. *Journal of Scientific Computing*, 24(1):45–78, jul 2005.

[74] J. Makino, T. Fukushige, and M. Koga. A 1.349 tflops simulation of black holes in a galactic center on GRAPE-6. In *ACM/IEEE SC 2000 Conference (SC'00)*. IEEE, 2000.

[75] Dhairya Malhotra and George Biros. PVFMM: A parallel kernel independent FMM for particle and volume potentials. *Communications in Computational Physics*, 18(03):808–830, sep 2015.

[76] Dhairya Malhotra and George Biros. Algorithm 967: A distributed-memory fast multipole method for volume potentials. *ACM Transactions on Mathematical Software*, 43(2):1–27, aug 2016.

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

[78] Anita Mayo. The fast solution of poisson's and the biharmonic equations on irregular regions. *SIAM Journal on Numerical Analysis*, 21(2):285–299, apr 1984.

[79] Peter McCorquodale, Phillip Colella, Gregory Balls, and Scott Baden. A local corrections algorithm for solving poisson's equation in three dimensions. *Communications in Applied Mathematics and Computational Science*, 2(1):57–81, aug 2007.

[80] A. McKenney, L. Greengard, and A. Mayo. A fast poisson solver for complex geometries. *Journal of Computational Physics*, 118(2):348–355, may 1995.

[81] Matthias Messner, Bérenger Bramas, Olivier Coulaud, and Eric Darve. Optimized m2l kernels for the chebyshev interpolation based fast multipole method. *arXiv preprint arXiv:1210.7292*, 2012.

[82] C. Pozrikidis. *Boundary Integral and Singularity Methods for Linearized Viscous Flow*. Cambridge University Press, 1992.

[83] Bryan Quaife and George Biros. High-volume fraction simulations of two-dimensional vesicle suspensions. *Journal of Computational Physics*, 274:245–267, oct 2014.

[84] Bryan Quaife and George Biros. High-order adaptive time stepping for vesicle suspensions with viscosity contrast. *Procedia IUTAM*, 16:89–98, 2015.

[85] Bryan Quaife and George Biros. Adaptive time stepping for vesicle suspensions. *Journal of Computational Physics*, 306:478–499, feb 2016.

[86] Abtin Rahimian, Ilya Lashuk, Shravan Veerapaneni, Aparna Chandramowlishwaran, Dhairya Malhotra, Logan Moon, Rahul Sampath, Aashay Shringarpure, Jeffrey Vetter, Richard Vuduc, Denis Zorin, and George Biros. Petascale direct numerical simulation of blood flow on 200k cores and heterogeneous architectures. In *2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, nov 2010.

[87] Abtin Rahimian, Shravan Kumar Veerapaneni, and George Biros. Dynamic simulation of locally inextensible vesicles suspended in an arbitrary two-dimensional domain, a boundary integral method. *Journal of Computational Physics*, 229(18):6466–6484, sep 2010.

[88] Abtin Rahimian, Shravan K. Veerapaneni, Denis Zorin, and George Biros. Boundary integral method for the flow of vesicles with viscosity contrast in three dimensions. *Journal of Computational Physics*, 298:766–786, oct 2015.

[89] Vladimir Rokhlin. Application of volume integrals to the solution of partial differential equations. *Computers & Mathematics with Applications*, 11(7-8):667–679, jul 1985.

[90] Diego Rossinelli, George Karniadakis, Massimiliano Fatica, Igor Pivkin, Petros Koumoutsakos, Yu-Hang Tang, Kirill Lykov, Dmitry Alexeev, Massimo Bernaschi, Panagiotis Hadjidoukas, Mauro Bisson, Wayne Joubert, and Christian Conti. The in-silico lab-on-a-chip. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '15*. ACM Press, 2015.

[91] Kirill Serkh and Vladimir Rokhlin. On the solution of elliptic partial differential equations on regions with corners. *Journal of Computational Physics*, 305:150–171, jan 2016.

[92] Piotr K. Smolarkiewicz and C. Larrabee Winter. Pores resolving simulation of darcy flows. *Journal of Computational Physics*, 229(9):3121–3133, may 2010.

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

[94] Hari Sundar, Rahul S. Sampath, and George Biros. Bottom-up construction and 2:1 balance refinement of linear octrees in parallel. *SIAM Journal on Scientific Computing*, 30(5):2675–2708, jan 2008.

[95] Hari Sundar, George Biros, Carsten Burstedde, Johann Rudi, Omar Ghattas, and Georg Stadler. Parallel geometric-algebraic multigrid on unstructured forests of octrees. In *2012 International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, nov 2012.

[96] Hari Sundar, Dhairya Malhotra, and George Biros. HykSort. In *Proceedings of the 27th international ACM conference on International conference on supercomputing - ICS '13*. ACM Press, 2013.

[97] Hari Sundar, Dhairya Malhotra, and Karl W. Schulz. Algorithms for high-throughput disk-to-disk sorting. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '13*. ACM Press, 2013.

[98] 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, aug 2011.

[99] Toru Takahashi, Cris Cecka, and Eric Darve. Optimization of the parallel black-box fast multipole method on CUDA. In *2012 Innovative Parallel Computing (InPar)*. IEEE, may 2012.

[100] Svetlana Tlupova and J. Thomas Beale. Nearly singular integrals in 3d stokes flow. *Communications in Computational Physics*, 14(05):1207–1227, nov 2013.

[101] Anna-Karin Tornberg and Leslie Greengard. A fast multipole method for the three-dimensional stokes equations. *Journal of Computational Physics*, 227(3):1613–1619, jan 2008.

[102] Lloyd N. Trefethen. *Spectral Methods in MATLAB*. Society for Industrial and Applied Mathematics, jan 2000.

[103] Shravan K. Veerapaneni, Denis Gueyffier, George Biros, and Denis Zorin. A numerical method for simulating the dynamics of 3d axisymmetric vesicles suspended in viscous flows. *Journal of Computational Physics*, 228(19):7233–7249, oct 2009.

[104] Shravan K. Veerapaneni, Denis Gueyffier, Denis Zorin, and George Biros. A boundary integral method for simulating the dynamics of inextensible vesicles suspended in a viscous fluid in 2d. *Journal of Computational Physics*, 228(7):2334–2353, apr 2009.

[105] Shravan K. Veerapaneni, Abtin Rahimian, George Biros, and Denis Zorin. A fast algorithm for simulating vesicle flows in three dimensions. *Journal of Computational Physics*, 230(14):5610–5634, jun 2011.

[106] M.S. Warren and J.K. Salmon. Astrophysical n-body simulations using hierarchical tree data structures. In *Proceedings Supercomputing '92*. IEEE Comput. Soc. Press, 1992.

[107] M. S. Warren and J. K. Salmon. A parallel hashed oct-tree n-body algorithm. In *Proceedings of the 1993 ACM/IEEE conference on Supercomputing - Supercomputing '93*. ACM Press, 1993.

[108] Lexing Ying, George Biros, Denis Zorin, and Harper Langston. A new parallel kernel-independent fast multipole method. In *Proceedings of the 2003 ACM/IEEE conference on Supercomputing - SC '03*. ACM Press, 2003.

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

[110] Lexing Ying, George Biros, and Denis Zorin. A high-order 3d boundary integral equation solver for elliptic PDEs in smooth domains. *Journal of Computational Physics*, 219(1):247–275, nov 2006.

[111] R. Yokota and L. A. Barba. A tuned and scalable fast multipole method as a preeminent algorithm for exascale systems. *International Journal of High Performance Computing Applications*, 26(4):337–346, jan 2012.

[112] Rio Yokota, Jaydeep P. Bardhan, Matthew G. Knepley, L.A. Barba, and Tsuyoshi Hamada. Biomolecular electrostatics using a fast multipole BEM on up to 512 gpus and a billion unknowns. *Computer Physics Communications*, 182(6):1272–1283, jun 2011.

[113] Hong Zhao, Amir H.G. Isfahani, Luke N. Olson, and Jonathan B. Freund. A spectral boundary integral method for flowing blood cells. *Journal of Computational Physics*, 229(10):3726–3744, may 2010.

[114] ALEXANDER Z. ZINCHENKO and ROBERT H. DAVIS. Shear flow of highly concentrated emulsions of deformable drops by numerical simulations. *Journal of Fluid Mechanics*, 455, mar 2002.

[115] A. Z. Zinchenko and D. Robert H. Large-scale simulations of concentrated emulsion flows. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 361(1806):813–845, may 2003.