

**Fast Multipole-Based
Elliptic PDE Solver and Preconditioner**

Dissertation by

Huda Ibeid

In Partial Fulfillment of the Requirements

For the Degree of

Doctor of Philosophy

King Abdullah University of Science and Technology, Thuwal,
Kingdom of Saudi Arabia

October, 2016

EXAMINATION COMMITTEE PAGE

The dissertation of Huda Ibeid is approved by the examination committee

Committee Chairperson: Professor. David Keyes

Committee Members: Professor Hakan Bagci, Professor Xin Gao, Professor Rio Yokota, Professor William Gropp

©October 2016

Huda Ibeid

All Rights Reserved

ABSTRACT

Fast Multipole Method-Based
Elliptic PDE Solver and Preconditioner

Huda Ibeid

Exascale systems are predicted to have approximately one billion cores, assuming Gigahertz cores. Limitations on affordable network topologies for distributed memory systems of such massive scale bring new challenges to the currently dominant parallel programming model. Currently, there are many efforts to evaluate the hardware and software bottlenecks of exascale designs. It is therefore of interest to model application performance and to understand what changes need to be made to ensure extrapolated scalability. Fast multipole methods (FMM) were originally developed for accelerating N -body problems for particle-based methods in astrophysics and molecular dynamics. FMM is more than an N -body solver, however. Recent efforts to view the FMM as an elliptic PDE solver have opened the possibility to use it as a preconditioner for even a broader range of applications. In this thesis, we (i) discuss the challenges for FMM on current parallel computers and future exascale architectures, with a focus on inter-node communication, and develop a performance model that considers the communication patterns of the FMM for spatially quasi-uniform distributions, (ii) employ this performance model to guide performance and scaling improvement of FMM for all-atom molecular dynamics simulations of uniformly distributed particles, and (iii) demonstrate that, beyond its traditional use as a solver in problems for which

explicit free-space kernel representations are available, the FMM has applicability as a preconditioner in finite domain elliptic boundary value problems, by equipping it with boundary integral capability for satisfying conditions at finite boundaries and by wrapping it in a Krylov method for extensibility to more general operators. Compared with multilevel methods, FMM is capable of comparable algebraic convergence rates down to the truncation error of the discretized PDE, and it has superior multicore and distributed memory scalability properties on commodity architecture supercomputers. Compared with other methods exploiting the low rank character of off-diagonal blocks of the dense resolvent operator, FMM-preconditioned Krylov iteration may reduce the amount of communication because it is matrix-free and exploits the tree structure of FMM. Fast multipole-based solvers and preconditioners are demonstrably poised to play a leading role in exascale computing.

TABLE OF CONTENTS

EXAMINATION COMMITTEE PAGE	2
Copyright	3
Abstract	4
List of Figures	11
List of Tables	14
1 Introduction	16
1.1 Objectives and Contributions	20
1.2 Contents of the Thesis	21
2 Fast Multipole Method	23
2.1 Basic Components	24
2.2 Flow of Calculation	25
2.2.1 Dual tree traversal	27
2.3 Multipole Expansions	30
2.4 FMM Communication Scheme	32
3 A Performance Model for the Communication in FMMs for Spatially Uniform Distributions	35
3.1 Related Work	36
3.2 Performance Challenges	37
3.2.1 Trends in computer hardware	38
3.2.2 Communication	38
3.3 FMM Communication Phases	39
3.3.1 Global M2L	39

3.3.2	Global M2M	40
3.3.3	Local M2L	41
3.3.4	Local P2P	41
3.4	Modeling Performance	42
3.4.1	Baseline model $((\alpha, \beta)$ model)	43
3.4.2	Distance penalty $((\alpha, \beta, \gamma)$ model)	43
3.4.3	Bandwidth penalty on β	44
3.4.4	Multicore penalty on α or γ	44
3.5	Model Validation	44
3.5.1	Machine description	44
3.5.2	Experimental setup	46
3.5.3	Model validation	48
3.6	Conclusions	56
4	Molecular Dynamics Simulation of Uniformly Distributed Particles Using the FMM	58
4.1	Computational Methods	60
4.1.1	Basic MD functions	61
4.1.2	Non-bonded short range force	62
4.1.3	Non-bonded long-range force	63
4.1.4	Periodic fast multipole method	64
4.2	Performance Benchmarks	64
4.2.1	Efficiency of the force calculation kernel	65
4.2.2	Sustained performance of the large-scale simulations	66
4.2.3	Load balance	68
4.2.4	Performance of the FMM	69
4.2.5	Accuracy of periodic FMM	71
4.3	Macromolecular Crowding Simulations with All Atoms	73
4.3.1	Simulation methods	73
4.3.1.1	Simulated systems	73
4.3.1.2	MD Simulations	73
4.3.2	Simulations of macromolecular crowding	74
4.4	Conclusions	74
5	FMM-Based Preconditioners for Sparse Iterative Solvers	76
5.1	Model Problems	77
5.1.1	Poisson model problem	78

5.1.2	Stokes model problem	78
5.1.3	Helmholtz model problem	79
5.2	Iterative Solvers and Preconditioning	80
5.2.1	Krylov subspace methods	80
5.2.2	Preconditioning	82
5.2.3	The FMM-BEM preconditioner	85
5.3	Boundary Element Method	86
5.3.1	Formulation	86
5.3.2	Singular integrals	89
5.3.3	Discretization	89
5.3.4	Variable coefficient problems	91
5.4	Numerical Results	92
5.4.1	The Poisson equation	92
5.4.2	Variable-coefficient Poisson equation	95
5.4.3	Stokes problem	97
5.4.4	The Helmholtz equation	98
5.4.5	Variable-coefficient Helmholtz equation	103
5.4.6	Effect of FMM precision on convergence	104
5.5	Performance Analysis	109
5.5.1	The Poisson equation	109
5.5.1.1	Serial results	109
5.5.1.2	Parallel results	111
5.5.1.3	Extension to 3-D	112
5.5.2	The Helmholtz equation	114
5.6	Conclusions	115
6	FMM as a Matrix-Free Hierarchically Low Rank Approximation	117
6.1	Hierarchically Low Rank Approximation: Analytic or Algebraic? . . .	119
6.1.1	Analytic low-rank approximation	119
6.1.2	Semi-analytical FMM	120
6.1.3	Algebraic low-rank approximation	121
6.2	Experimental Results	122
6.2.1	Matrix-vector multiplication	123
6.3	Conclusions	125

7 Summary and Future Work	127
7.1 Summary	127
7.2 Future Research Work	128
7.2.1 Nonuniform distributions	128
7.2.2 Future vision of the macromolecular crowding effect	129
7.2.3 Practical applications of the FMM preconditioner	129
7.2.4 Benchmarking HLRA based methods	130
References	132
Appendices	151
Appendix A Machines Description	152
A.1 Titan	152
A.2 K computer	152
A.3 Mira	153
A.4 Shaheen II	153
A.5 Stampede	153
A.6 Piz Dora	154
A.7 Shaheen I	154
Appendix B Mathematical Supplements	155
B.1 Divergence Theorem	155
B.2 Dirac's Delta Function (δ)	155
B.3 Fundamental Solutions	156
B.3.1 Laplace Equation	156
B.3.1.1 Fundamental solution in 2-D	156
B.3.1.2 Fundamental solution in 3-D	158
B.3.2 Helmholtz Equation	160
B.3.2.1 Fundamental solution in 2-D	160
B.3.2.2 Fundamental solution in 3-D	161
Appendix C Boundary Element Method	163
C.1 Weighted Residual Methods	163
C.2 Weak Form	164
C.3 Inverse Form	165
C.4 BEM Matrices	166
C.5 Discretization	169

C.6 Matrix Construction	172
Appendix D Papers Accepted, Submitted, and Under Preparation	175

LIST OF FIGURES

2.1	Hierarchical decomposition	24
2.2	Schematic of Fast Multipole Method. (a) shows the interactions for a $\mathcal{O}(N^2)$ direct method. (b) shows the interactions for the $\mathcal{O}(N)$ FMM, describing the type of interaction between elements in the tree data structure. (c) shows the same FMM kernels as in (b), but from a geometric point of view of the hierarchical domain decomposition. . .	26
2.3	Decomposition of the distance vector $\mathbf{x}_{ij} = \mathbf{x}_i - \mathbf{x}_j$ into five parts, that correspond to the five stages P2M, M2M, M2L, L2L, and L2P in the FMM.	31
2.4	Splitting of the local and global tree in FMM.	33
2.5	Heat maps for level-by-level communication patterns for the M2L phase of an FMM with $N=62,500$ per process using 128 processes. Areas of black indicate zero messages between processes, the peak communication volume is represented in red. In this example, the switch between global and local trees is between Level 3 and Level 4.	34
3.1	Performance model prediction and actual time for M2L communication phase on Shaheen I.	49
3.2	Load balance of M2L communication phase on Shaheen I.	50
3.3	Performance model prediction and actual time for M2L communication phase on Mira.	53
3.4	Performance model prediction and actual time for M2L communication phase on Titan.	54
3.5	Performance model prediction and actual time for M2L communication phase on Piz Dora.	55
4.1	Vicinal area of TTHA for the <i>in vivo</i> system. The TTHA molecule is drawn using the ribbon model, and ovalbumin molecules are drawn using the space-filled model. Each ovalbumin molecule is colored differently.	60
4.2	Weak scaling for 6,542 atoms/node. Constant wall clock times/step show the perfect scaling. Ratios of force calculation/communication/others are also constant, and force calculation is dominant.	67
4.3	Strong scaling to number of atoms per node. The red curve shows Amdahl's law.	68

4.4	Computational workload (FLOP count) in each node.	70
4.5	Weak scaling of FMM part of the MD simulation for 6,542 atoms/node. “Local FMM” is the aggregate time of all FMM kernels in the local tree, “Communicate Cells” is the time of the M2L communication in the local trees, “global M2M” and “global M2L” include both the communication time and computation time of the M2M and M2L kernels in the global tree. The dashed line is a reference for confirming the $\log P$ behavior of global communications.	71
4.6	Error of periodic FMM with respect to the order of expansion P and number of periodic images $3^k \times 3^k \times 3^k$. The error is the relative L^2 norm of the difference between the force from the Ewald summation and periodic FMM. With this measure, 10^{-8} indicates that 8 significant digits are matching.	72
4.7	Conformational fluctuations of TTHA for <i>in vivo</i> (viv) and <i>in vitro</i> systems (vit). The red and green lines indicate the RMSFs of TTHA in viv (FMM: solid, PME: dashed) and vit (FMM: solid, PME: dashed), respectively. The abscissa axis is the residue number of TTHA and the ordinate axis is the RMSF value (in Angstrom). Red and white boxes indicate alpha-helices and beta-sheets, respectively.	75
5.1	Solution of the Laplace and Helmholtz equations with the same boundary conditions.	84
5.2	Evolution of the residual of the unpreconditioned GMRES method for the Laplace equation, $k = 0$, and the Helmholtz equation, $k = 15$. . .	85
5.3	Flow chart of the FMM-BEM preconditioner within the conjugate gradient method.	87
5.4	Eigenvalues of the the coefficient matrix A for the problem described in (5.19) with $k = 5, 10, 20$, and 40 , respectively, $h = 2^{-5}$	99
5.5	The right-hand term f and solution u of (5.22) for $\mu = 1, 4, 6$, and 8	102
5.6	Convergence rate of the FMM preconditioner for the Poisson equation with different precision, plotted along with algebraic multigrid, geometric multigrid, and incomplete Cholesky preconditioners. The ϵ represents the precision of the FMM, where $\epsilon = 10^{-6}$ corresponds to six significant digits of accuracy.	105
5.7	Convergence rate of the FMM preconditioner for the Helmholtz equation with different precision, plotted along with AMG, GMG, and IC preconditioners. The ϵ represents the precision of the FMM where $\epsilon = 10^{-6}$ corresponds to six significant digits of accuracy, $h = 2^{-5}$. . .	106
5.8	Convergence of spatial discretization error for the FEM and BEM. The relative L^2 norm of the difference between the analytical solution is plotted against the grid spacing Δx	107
5.9	Eigenvalues of the coefficient matrix A and the FMM-preconditioned matrix $M^{-1}A$ with different FMM precisions $\epsilon = 10^{-2}, 10^{-4}$, and 10^{-6} , $h = 2^{-5}$	108

5.10	Time-to-solution for different problem sizes of the FMM and AMG preconditioners on a single core of a Xeon E5-2680.	110
5.11	Strong scaling of the 2-D FMM and AMG preconditioners.	112
5.12	Calculation time of 2-D and 3-D FMM for the same problem size. . .	113
5.13	Strong scaling of the 3-D Poisson FMM and AMG preconditioners. .	114
5.14	Calculation time of 3-D Laplace and 3-D Helmholtz FMM for the same problem size.	115
6.1	The compute-memory trade-off between the analytic and algebraic hierarchically low rank approximation methods. Various techniques lie between the analytic and algebraic extremes.	119
6.2	Elapsed time for the matrix-vector multiplication using FMM and HSS for different problem sizes.	123
6.3	Percentage of the computation time of HSS for different problem sizes.	124
6.4	Peak memory usage of FMM and HSS for the 3-D Laplace equation. .	125
C.1	Discretization of global integral into a sum of piecewise local integrals	168
C.2	Local integral by superposition of basis functions	169
C.3	Element centric and node centric approaches for BEM discretization .	173

LIST OF TABLES

3.1	Amount of communication in FMM	39
3.2	Machine parameters for latency α , inverse bandwidth β , and distance penalty γ , on Shaheen I, Mira, Titan, and Piz Dora.	45
3.3	Statistics of the M2L communication.	47
4.1	Performance on the main loop and the kernel loop for 418,707 atom simulation by 64 process	65
4.2	Conditions of the simulation for the peak performance	66
5.1	Preconditioned CG iterations for the relative residual to reduce by six orders of magnitude for the problem with $-\nabla^2 u = 1$ and homogeneous boundary conditions.	93
5.2	Smallest (λ_{min}) and largest (λ_{max}) eigenvalues and condition number (κ) of the stiffness matrix A and FMM-preconditioned matrix $P^{-1}A$ for the problem with $-\nabla^2 u = 1$ and homogeneous boundary conditions.	93
5.3	Preconditioned CG iterations for the relative residual to reduce by six orders of magnitude for the problem with $-\nabla^2 u = 0$ and inhomogeneous boundary conditions.	95
5.4	Preconditioned CG iterations for the relative residual to reduce by six orders of magnitude for the problem with $-\nabla^2 u = -4$ and inhomogeneous boundary conditions.	95
5.5	Preconditioned CG iterations for the relative residual to reduce by six orders of magnitude ($h = 2^{-6}$, $m = 1$ and $n = 1$).	96
5.6	Preconditioned CG iterations for the relative residual to reduce by six orders of magnitude ($h = 2^{-6}$, $\mu = 2^{-4}$).	96
5.7	Preconditioned CG iterations for the relative residual to reduce by six orders of magnitude ($m = 4$, $h = 2^{-6}$, $\mu = 2^{-4}$).	97
5.8	Smallest (λ_{min}) and largest (λ_{max}) eigenvalues and condition number (k) of the stiffness matrix A and FMM-preconditioned matrix $P^{-1}A$ with $\mu = 2^{-4}$	97
5.9	Preconditioned MINRES iterations for the relative residual to reduce by six orders of magnitude for the Stokes problem.	98
5.10	Preconditioned GMRES iterations for the relative residual to reduce by six orders of magnitude, $kh = 0.3125$	100

5.11	Preconditioned GMRES iterations for the relative residual to reduce by six orders of magnitude, $k = 5$	100
5.12	Preconditioned GMRES iterations for the relative residual to reduce by six orders of magnitude, $h = 2^{-6}$	101
5.13	Preconditioned GMRES iterations for the relative residual to reduce by six orders of magnitude for $\mu = 1, 4, 6,$ and 8	103
5.14	Preconditioned GMRES iterations for the relative residual to reduce by six orders of magnitude ($h = 2^{-5}, m = 4, n = 16,$ and $k = 10$). . .	104
5.15	Preconditioned GMRES iterations for the relative residual to reduce by six orders of magnitude ($h = 2^{-6}, \mu = 2^{-4},$ and $k = 15$).	104
5.16	Preconditioned GMRES iterations for the relative residual to reduce by six orders of magnitude ($m = 4, n = 4, h = 2^{-6}, \mu = 2^{-4}$).	105
6.1	Categorization of algebraic low-rank approximation methods.	121
A.1	Machines Description (as of November, 2016[1])	154
C.1	Weighted Residual Methods	164
C.2	Different basis functions	171

Chapter 1

Introduction

Elliptic PDEs arise in a vast number of applications in scientific computing. A significant class of these involve the Laplace operator, which appears not only in potential calculations but also in, for example, Stokes and Navier-Stokes problems [2, Chapters 5 and 7], electron density computations [3, Part II] and reaction-convection-diffusion equations [4, Part IV]. Another important class of elliptic PDEs is the Helmholtz equation, which can be used to describe both wave propagation and scattering phenomena arising in many fields of science and technology. For example, Helmholtz equations are used to express acoustic phenomena in aeronautics [5] and underwater acoustics [6, 7]. They are also utilized in geophysical applications [8] and in electromagnetic applications, e.g., photolithography [9]. Consequently, the rapid solution of such PDEs is of wide interest.

The classification of a PDE is related to its characteristic curves, which are paths in the solution domain along which information propagates. Elliptic PDEs have no real characteristic curves. The solution at every point in the solution domain depends upon the solution at all other points, including the boundaries. This all-to-all mathematical dependence must be accommodated in any solution algorithm.

Although many successful numerical methods for such PDEs exist, changing computer architectures necessitate new paradigms for computing and the development of new algorithms. Computer architectures of the future will favor algorithms with

high concurrency, high data locality, high arithmetic intensity (Flop/Byte), and low synchronicity. This trend is manifested on GPUs and co-processors, where some algorithms are accelerated much less than others on the class of architectures that can be extended to extreme scale. There is always a balance between algorithmic efficiency in a convergence sense, and how well an algorithm scales on parallel architectures. This balance is shifting towards increased parallelism, even at the cost of increasing computation. Since the processor frequency has plateaued for the last decade, Moore's law holds continued promise only for those who are willing to make algorithmic changes.

Among the scientific applications ripe for reconsideration, those governed by elliptic PDEs will be among the most challenging. A common solution strategy for such systems is to discretize the partial differential equations by fairly low-order finite element, finite volume or finite difference methods and then solve the resulting large, sparse linear system. However, elliptic systems are global in nature, and this does not align well with the sweet spots of future architectures. The linear solver must enable the transfer of information from one end of the domain to the other, either through successive local communications (as in many iterative methods), or a direct global communication (as in direct solvers with global recurrences, multigrid methods with global coarse grids, or Krylov methods with global reductions). In either case, avoiding synchronization and reducing communication are the main challenges.

Scalable algorithms for solving elliptic PDEs tend to have a hierarchical structure, as in multigrid methods [10], fast multipole methods (FMM) [11], and \mathcal{H} -matrices [12]. This structure is crucial, not only for achieving optimal arithmetic complexity, but also for minimizing data movement. For example, the standard 3-D FFT with three all-to-all communications requires $\mathcal{O}(\sqrt{P})$ communication for the transpose between pencil-shaped subdomains on P processes [13] and a recently published algorithm [14] with five all-to-all communication phases achieves $\mathcal{O}(P^{1/3})$ communication, whereas

these hierarchical methods require $\mathcal{O}(\log P)$ communication [15]. This $\mathcal{O}(\log P)$ communication complexity is likely to be optimal for elliptic problems, since an appropriately coarsened representation of a local forcing must somehow arrive at all other parts of the domain for the elliptic equation to converge. In other words, an elliptic problem for which the solution is desired everywhere cannot have a communication complexity of $\mathcal{O}(1)$. However, the convergence of these hierarchical solvers can be fragile with respect to coefficient distribution in the second-order term, and, if present, with respect to the first-order and zeroth-order terms.

Krylov subspace methods provide another popular alternative to direct methods for general operators. We note that methods such as Chebyshev semi-iteration can require even less communication in the fortunate case when information about the spectrum of the coefficient matrix is known [16, Section 10.1.5], [17]. Among the best known Krylov methods are the conjugate gradient method [18], MINRES [19] and GMRES [20], although a multitude of Krylov solvers are available in popular scalable solver libraries. The great advantage of these solvers is their robustness – for any consistent linear system there exists a Krylov method that will converge, in exact arithmetic, for sufficiently many iterations. However, the convergence rate of Krylov methods typically deteriorates as the discretization of an elliptic PDE is refined.

Mesh-independent convergence for Krylov methods applied to systems from elliptic PDEs can often be recovered by preconditioning. Among the best performing preconditioners are the optimal hierarchical methods or, for multiphysics problems such as Stokes and Navier-Stokes equations, block preconditioners with these methods as components. By combining these hierarchical methods and Krylov subspace solvers we get the benefits of both approaches and obtain a linear solver that is fast but robust. These hierarchical methods all have multiple parameters for controlling the precision of the solution and are able to trade-off accuracy for speed, which is a useful feature for a preconditioner. Furthermore, in analogy to geometric multigrid

and algebraic multigrid, \mathcal{H}^2 -matrices [21] can be thought of as an algebraic generalization of what FMMs do geometrically. There are advantages and disadvantages to using algebraic and geometric methods, and both have their place as preconditioners.

There has been some recent work on algebraic multigrid methods (AMG) in anticipation of the future hardware constraints mentioned above. Gahvari *et al.* developed a performance model for AMG and tested it on various HPC systems – Intrepid, Jaguar, Hera, Zeus, and Atlas [22]. They found that network distance and contention were both substantial performance bottlenecks for AMG. Adams presents a low-memory matrix-free full multigrid (FMG) with a full approximation storage (FAS) [23]. He revives an idea from the 1970s [24], which processes the multigrid algorithm vertically, and improves data locality and asynchronicity. Baker *et al.* compared the scalability of different smoothers – hybrid Gauss-Seidel, l_1 Gauss-Seidel, and Chebyshev polynomial, and showed that l_1 Gauss-Seidel and Chebyshev smoothers scale much better [25]. Vassilevski and Yang [26] present additive variants of AMG that are significantly improved with respect to classical additive methods and show their scalable performance on up to 4,096 cores. Indeed, there is continuous progress to evolve multigrid to future hardware constraints, and it is likely that multigrid will remain competitive.

On the other hand, performing a hierarchically low rank approximation (HLRA) of the off-diagonal blocks of a matrix leads to a whole new variety of $\mathcal{O}(N)$ solvers or preconditioners. HLRA-based methods include FMM itself [11], \mathcal{H} -matrices [12], hierarchically semi-separable matrices [27], hierarchically off-diagonal low-rank technique [28], and recursive skeletonization [29]. These techniques can be applied to a dense matrix or the Schur complement during a sparse direct solve, thus enabling an $\mathcal{O}(N)$ matrix-vector multiplication of a $N \times N$ dense matrix or an $\mathcal{O}(N)$ direct solve of a $N \times N$ sparse matrix to within a specified accuracy. These HLRA-based methods require a smooth kernel in the far field which yields a block low-rank structure. The

distinguishing features of the variants come in the way the low-rank approximation is constructed – rank-revealing LU [30], rank-revealing QR [31], pivoted QR [32], truncated SVD [33], randomized SVD [34], adaptive cross approximation [35], hybrid cross approximation [36], and Chebychev interpolation [37] are all possibilities. Multipole/local expansions in the FMM constitute another way to construct the low-rank approximations. In fact, many of the original developers of FMM are now working on these algebraic variants [38].

Literature on the HLRA-based methods mentioned above mainly focuses on the error convergence of the low-rank approximation and there is little investigation of the parallel scalability or direct comparison against multigrid. An exception is the work by Grasedyck *et al.* [39], where their \mathcal{H} -LU preconditioner is compared with BoomerAMG, Pardiso, MUMPS, UMFPACK, SuperLU, and Spooles. However, their executions are serial, and show that their \mathcal{H} -matrix code is not yet competitive with these other highly optimized libraries. Another is the work by Gholami *et al.* [40] where they compare FFT, FMM, and multigrids methods for the Poisson problem with constant coefficients on the unit cube with periodic boundary conditions.

1.1 Objectives and Contributions

The contributions of this thesis folds in the following streams:

- We discuss the challenges for FMM on current parallel computers and future exascale architectures, with a focus on inter-node communication. We also develop a performance model that considers the communication patterns of the FMM for spatially uniform distributions, and observe a good match between our model and the actual communication time on four HPC systems, when latency, bandwidth, network topology, and multi-core penalties are all taken into account.

- We discuss the performance and scaling of FMMs for molecular dynamics simulations of uniformly distributed particles on the K computer. To measure the performance of the K computer, we performed all-atom classical molecular dynamics simulations of two systems: target proteins in a solvent, and target proteins in an environment of molecular crowders that mimic the conditions of a living cell. Using the full system, we achieved 4.4 PFLOPs during a 520 million-atom simulation with cutoff of 28 Å.
- We demonstrate that, beyond its traditional use as a solver in problems for which explicit free-space kernel representations are available, the FMM has applicability as a preconditioner in finite domain elliptic boundary value problems, by equipping it with boundary integral capability for satisfying conditions at finite boundaries and by wrapping it in a Krylov method for extensibility to more general operators.
- We provide a thorough review of the recent advancements in the field of HLRA from both analytical and algebraic perspectives, and present a comparative benchmark of highly optimized implementations of contrasting methods for some simple yet representative test cases.
- We describe all our tests in reproducible detail with freely available codes and outline directions for further extensibility.

1.2 Contents of the Thesis

The thesis is organized as follows. Chapter 2 describes the FMM, the essential kernel that makes our methods efficient and scalable. We develop a performance model for the communication in the FMM in Chapter 3. Our performance model is validated on four different architectures, Shaheen I (BG/P), Mira (BG/Q), Titan (Cray XK7), and Piz Dora (Cray XC40). Then in Chapter 4, we discuss the performance and scaling

of FMMs for molecular dynamics simulations of uniformly distributed particles. In Chapter 5, we combine the FMM with Krylov iteration as a scalable and highly performant preconditioner for traditional low-order finite discretizations of elliptic boundary value problems. The FMM-preconditioner is compared with multilevel and sparse direct solvers on a variety of elliptic problems. Chapter 6 shows the contrast between the analytical and algebraic hierarchically low rank approximations, by reviewing the contributions over the years and placing them along the analytical-algebraic spectrum. Our concluding remarks are given in Chapter 7.

Chapter 2

Fast Multipole Method

N -body problems arise in many areas of physics (e.g., astrophysics, molecular dynamics, acoustics, electrostatics). In these problems, the system is described by a set of N particles and the dynamics of the system arise from interactions that occur between every pair of particles, which has the form

$$f(\mathbf{x}_i) = \sum_{j=1}^N q_j K(\mathbf{x}_i, \mathbf{x}_j). \quad (2.1)$$

Here, $f(\mathbf{x}_i)$ represents a field value evaluated at a point \mathbf{x}_i which is generated by the influence of sources located at \mathbf{x}_j with weights q_j . $K(\mathbf{x}_i, \mathbf{x}_j)$ is the kernel that governs the interactions between evaluation and source particles. The direct approach to simulate the N -body problem is relatively simple; it evaluates all pairwise interactions among the particles. While this method is exact to within machine precision, the solution is $\mathcal{O}(N^2)$ in its computational complexity, which is prohibitively expensive for even modestly large data sets. However, its simplicity and ease of implementation make it an appropriate choice when simulating small particle sets ($N < 1000$) where high accuracy is desired [41]. For a larger number of particles, many faster algorithms have been invented, e.g., treecodes [42] and the fast multipole method [11]. The main idea behind these fast algorithms is to coarse grain the effect of sufficiently far particles as permitted by rigorous analysis. The most common way to achieve this approximation is to cluster the far particles into successively larger

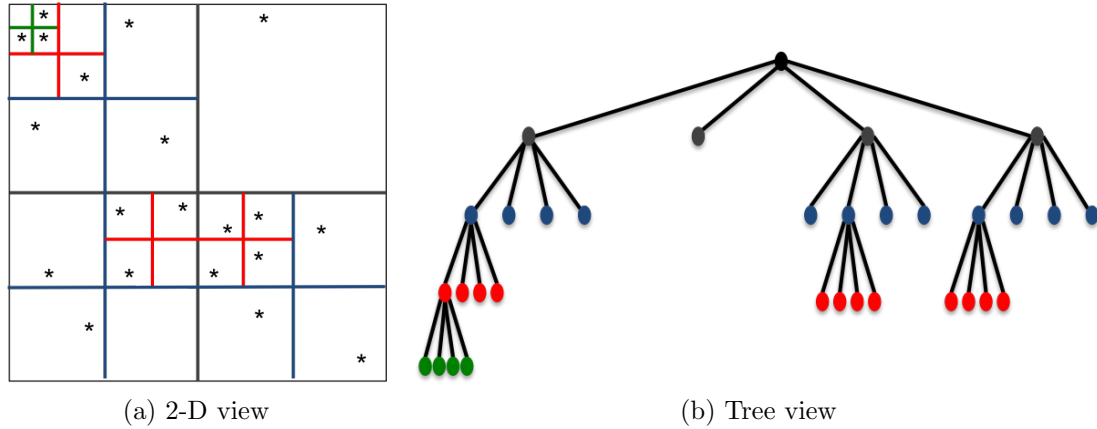


Figure 2.1: Hierarchical decomposition

groups by constructing a tree. The treecode clusters the far particles and achieves $\mathcal{O}(N \log N)$ complexity. The FMM further clusters the near particles in addition to the far particles to achieve $\mathcal{O}(N)$ complexity.

In this chapter, we present an overview of the FMM algorithm. First, the spatial hierarchy and fast approximate evaluation of the FMM are discussed in Section 2.1. The flow of calculation of the FMM algorithm is provided in Section 2.2. Section 2.3 presents the mathematics behind the specific FMM kernels. Then, in Section 2.4, a description of the communication pattern introduced by the domain partitioning scheme used in the FMM algorithm is described.

2.1 Basic Components

Both treecodes and FMM are based on two key ideas: the tree representation for the spatial hierarchy, and the fast approximate evaluation. The spatial hierarchy means that the computational domain is hierarchically decomposed into increasing levels of refinement, and then the near and far subdomains can be identified at each level. It is common to use an octree in 3-D and quad-tree in 2-D, where the space is recursively split until the finest level of refinement or “leaf level”. Figure 2.1 illustrates such hierarchical space decomposition for a 2-D domain (a), associated to a quad-

tree structure (b). The splitting is usually performed adaptively, so that the densely populated areas result in a deeper branching of the tree. The original FMM [43] is based on a series expansion of the Laplace Green's function ($1/r$) and therefore can be applied to the evaluation of related potentials and/or forces [44]. The approximation reduces the number of operations in exchange for accuracy.

2.2 Flow of Calculation

In Figure 2.2, we show by schematic how the fast multipole method is able to calculate (2.1) in $\mathcal{O}(N)$ operations. Figures 2.2a and 2.2b show how the source particles (red) interact with the target particles (blue) for the direct method and FMM, respectively. In the direct method, all source particles interact with all target particles directly. In the FMM, the source particles are first converted to multipole expansions using the P2M (particle to multipole) kernel. Figure 2.2c shows the corresponding geometric view of the hierarchical domain decomposition of the particle distribution. Then, multipole expansions are aggregated into larger groups using the M2M (multipole to multipole) kernel. Following this, the multipole expansions are translated to local expansions between well-separated cells using the M2L (multipole to local) kernel. Both Figures 2.2b and 2.2c show that the larger cells interact if they are significantly far away, and smaller cells may interact with slightly closer cells. The direct neighbors between the smallest cells are calculated using the P2P (particle to particle) kernel, which is equivalent to the direct method between a selected group of particles. Then, the local expansions of the larger cells are translated to smaller cells using the L2L (local to local) kernel. Finally, the local expansions at the smallest cells are translated into the potential on each particle using the L2P (local to particle) kernel. The mathematical formula for these kernels will be given in Section 2.3.

As discussed earlier, in order to perform the FMM calculation mentioned above, one must first decompose the domain in a hierarchical manner by splitting it by its

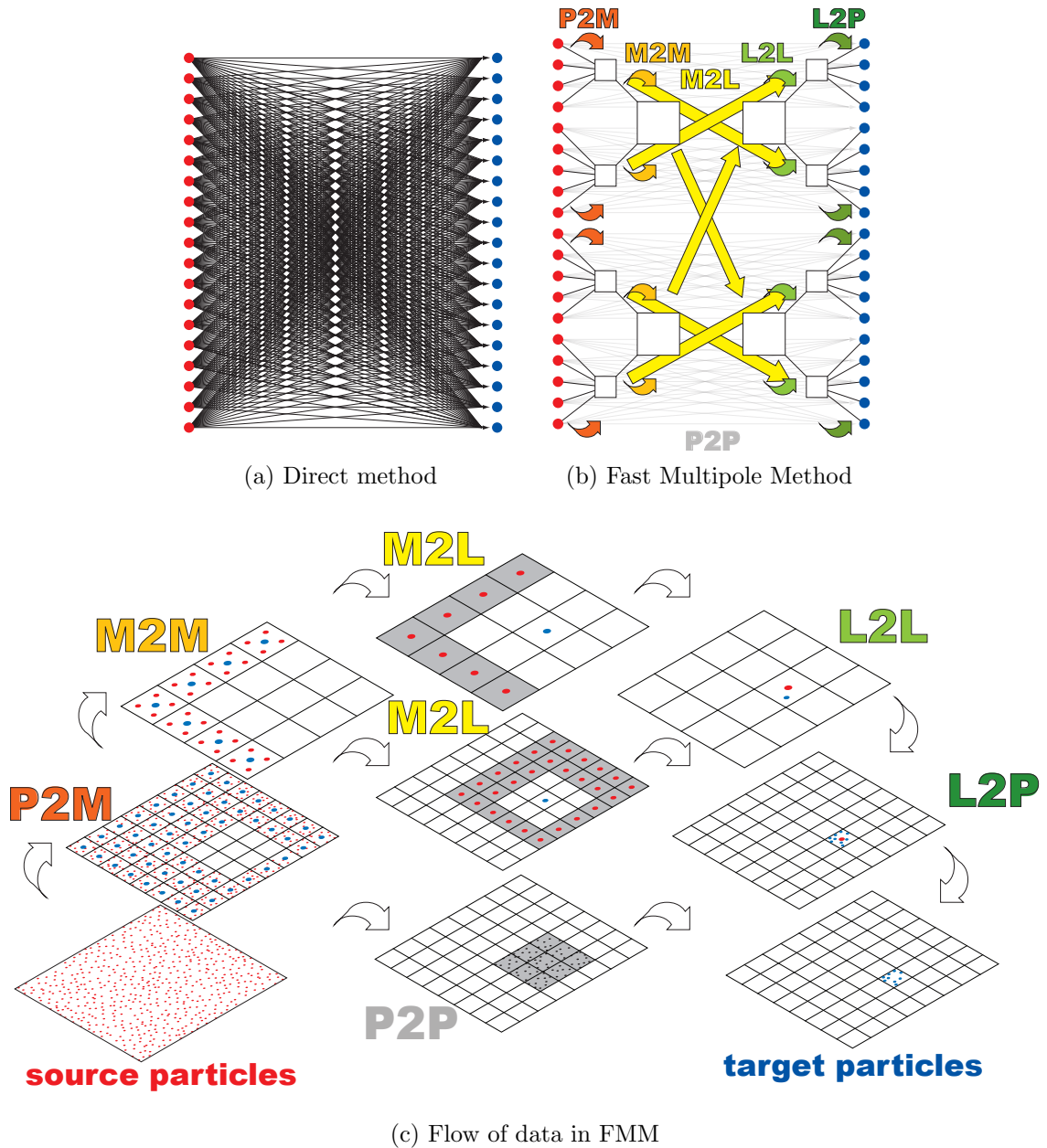


Figure 2.2: Schematic of Fast Multipole Method. (a) shows the interactions for a $\mathcal{O}(N^2)$ direct method. (b) shows the interactions for the $\mathcal{O}(N)$ FMM, describing the type of interaction between elements in the tree data structure. (c) shows the same FMM kernels as in (b), but from a geometric point of view of the hierarchical domain decomposition.

geometrical centerline. This splitting is performed recursively until the number of particles per cell reaches a prescribed threshold. A common requirement in FMMs is that these cells must be isotropic (cubes or squares and not rectangles), since they are used as units for measuring the well-separatedness, as shown in Figure 2.2c, during the M2L interaction. However, our FMM does not use the size of cells to measure the distance between them and allows the cells to be of any shape as long as they can be hierarchically grouped into a tree structure. Once the tree structure is constructed, it is trivial to find parent-child relationships between the cells/particles. This relation is all that is necessary for performing P2M, M2M, L2L, and L2P kernels. However, for the M2L and P2P kernels one must identify a group of well-separated and neighboring cells, respectively. We will describe an efficient method for finding well-separated cells in the following section.

2.2.1 Dual tree traversal

The simplest method for finding well-separated pairs of cells in the FMM is to “loop over all target cells and find their parent’s neighbor’s children that are non-neighbors,” as shown by Greengard and Rokhlin [11]. A scheme that permits the interaction of cells at different levels for an adaptive tree was introduced by Carrier *et al.* [45]. This scheme is used in many modern FMM codes, and is sometimes called the UVWX-list [15]. Another scheme to find well-separated pair of cells is to “simultaneously traverse the target and source tree while applying a multipole acceptance criterion (MAC),” as shown by Warren and Salmon [46]. Teng [47] showed that this dual tree traversal can produce interaction pairs that are almost identical to the adaptive interaction list by Carrier *et al.* [45]. A concise explanation and optimized implementation of the dual tree traversal is provided by Dehnen [48].

The dual tree traversal has many favorable properties compared to the explicit construction of interaction lists. First of all, the definition of well-separatedness can

be defined quite flexibly. For example, if one were to construct explicit interaction lists by extending the definition of neighbors from $3 \times 3 \times 3$ to $5 \times 5 \times 5$ using the traditional scheme, the M2L list size will increase rapidly from $6^3 - 3^3 = 189$ to $10^3 - 5^3 = 875$ in 3-D, which is never faster for any order of expansions. On the other hand, the dual tree traversal can adjust the definition of neighbors much more flexibly and the equivalent interaction list always has a spherical shape. (We say “equivalent interaction list” because there is no explicit interaction list construction in the dual tree traversal.) The cells no longer need to be cubic, since the cells themselves are not used to measure the proximity of cells. Of course, the explicit interaction list construction can be modified to include more flexibility, too [49]. However, the resulting code becomes much more complicated than the dual tree traversal, which is literally a few lines of code, see Pseudocode 1. This simplicity is a large advantage on its own. Furthermore, the parallel version of the dual tree traversal simply traverses the local tree for the target with the local essential tree (LET)¹ for the sources, so the serial dual tree traversal code can be used once the local essential tree is assembled.

A possible (but unlikely) limitation of dual tree traversals is the loss of explicit parallelism as it has no loops. It would not be possible to simply use an OpenMP “parallel for” directive to parallelize the dual tree traversal. In contrast, the traditional schemes always have an outer loop over the target cells, which can be easily parallelized and dynamically load balanced with OpenMP directives. However, this is not an issue since task based parallelization tools such as Intel Thread Building Blocks (TBB) can be used to parallelize the dual tree traversal. With the help of these tools, tasks are spawned as the tree is traversed and dispatched to idle threads dynamically. In doing so, we not only assure load balance but also data locality, so it may actually end up being a superior solution than parallelizing “for loops” with

¹The LET for any process is defined as the union of the interaction lists of all owned leaves and their ancestors, i.e., the data that will be required to compute the forces on every body in the local domain [50].

EvaluateFMMUsingDualTreeTraversal()

push pair of root cells (A, B) to stack;

```

while stack is not empty do
  | pop stack to get  $(A, B)$ ;
  | if target cell is larger than source cell then
  | | for all children a of target cell A do
  | | | Interact( $a, B$ );
  | | | end
  | | else
  | | | for all children b of target cell B do
  | | | | Interact( $A, b$ );
  | | | | end
  | | end
  | end
end

```

Interact(A, B)

```

if A and B are both leafs then
  | call P2P kernel;
else
  | if A and B satisfy MAC then
  | | call M2L kernel;
  | | else
  | | | push pair  $(A, B)$  to stack;
  | | | end
  | end
end

```

Pseudocode 1: Dual tree traversal

OpenMP, especially on NUMA architectures.

Considering the advantages mentioned above, we have decided to use the dual tree traversal in our current work. This allows us to perform low accuracy optimizations by adjusting the multipole acceptance criterion without increasing the order of expansions too much, which is the secret to our speed [51]. These low accuracy optimizations can give the FMM a performance boost when used as a preconditioner.

2.3 Multipole Expansions

For the 2-D Laplace equation, the free space Green's function has the form

$$G_{ij} = \frac{1}{2\pi} \log \left(\frac{1}{r_{ij}} \right), \quad (2.2)$$

where $r_{ij} = |\mathbf{x}_i - \mathbf{x}_j|$ is the distance between point i and point j . By using complex numbers to represent the two-dimensional coordinates $z = x + iy$, (2.1) can be written as

$$f(\mathbf{x}_i) = \sum_{j=1}^N \frac{q_j}{2\pi} \Re \{ -\log(z_{ij}) \}, \quad (2.3)$$

where $\Re(z)$ represents the real part of z . Figure 2.3 shows the decomposition of vector \mathbf{x}_{ij} into five parts, $\mathbf{x}_{ij} = \mathbf{x}_{i\lambda} + \mathbf{x}_{\lambda\Lambda} + \mathbf{x}_{\Lambda M} + \mathbf{x}_{M\mu} + \mathbf{x}_{\mu j}$, where λ and Λ are the center of local expansions and μ and M are the center of multipole expansions. The lower case is used for the smaller cells and upper case is used for the larger cells. We denote the n th order multipole expansion coefficient at \mathbf{x} as $M_n(\mathbf{x})$, and the n th order local expansion coefficient as $L_n(\mathbf{x})$, where $n = 0, 1, \dots, p - 1$ for a p th order truncation of the series. For well separated cells, we may assume the relation $|\mathbf{x}_{\Lambda M}| > |\mathbf{x}_{i\lambda} + \mathbf{x}_{\lambda\Lambda}| + |\mathbf{x}_{M\mu} + \mathbf{x}_{\mu j}|$. Given this relation, the following FMM approximations are valid [45]:

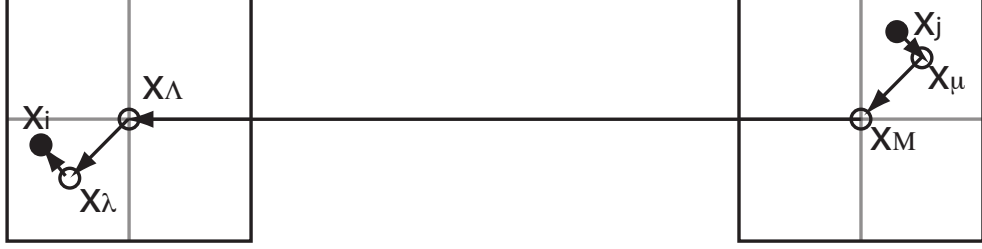


Figure 2.3: Decomposition of the distance vector $\mathbf{x}_{ij} = \mathbf{x}_i - \mathbf{x}_j$ into five parts, that correspond to the five stages P2M, M2M, M2L, L2L, and L2P in the FMM.

1. P2M from particle at \mathbf{x}_j to multipole expansion at \mathbf{x}_μ ,

$$M_0(\mathbf{x}_\mu) = \sum_{j=1}^{\hat{N}} q_j, \quad (2.4)$$

$$M_n(\mathbf{x}_\mu) = \sum_{j=1}^{\hat{N}} \frac{-q_j (-z_{\mu j})^n}{n}, \quad n = \{1, 2, \dots, p-1\}. \quad (2.5)$$

where \hat{N} is the total number of particles in the corresponding cell.

2. M2M from multipole expansion at \mathbf{x}_μ to multipole expansion at \mathbf{x}_M ,

$$M_0(\mathbf{x}_M) = M_0(\mathbf{x}_\mu), \quad (2.6)$$

$$M_n(\mathbf{x}_M) = -M_0(\mathbf{x}_\mu) \frac{(-z_{M\mu})^n}{n} + \sum_{k=1}^n M_k(\mathbf{x}_\mu) (-z_{M\mu})^{n-k} \binom{n-1}{k-1}. \quad (2.7)$$

3. M2L from multipole expansion at \mathbf{x}_M to local expansion at \mathbf{x}_Λ ,

$$L_0(\mathbf{x}_\Lambda) \approx M_0(\mathbf{x}_M) \log(z_{\Lambda M}) + \sum_{k=1}^{p-1} \frac{M_k(\mathbf{x}_M)}{z_{\Lambda M}^k}, \quad (2.8)$$

$$L_n(\mathbf{x}_\Lambda) \approx -\frac{M_0(\mathbf{x}_M)}{(-z_{\Lambda M})^n n} + \sum_{k=1}^{p-1} \frac{(-1)^n M_k(\mathbf{x}_M)}{z_{\Lambda M}^{n+k}} \binom{n+k-1}{k-1}. \quad (2.9)$$

4. L2L from local expansion at \mathbf{x}_Λ to local expansion at \mathbf{x}_λ ,

$$L_n(\mathbf{x}_\lambda) \approx \sum_{k=n}^{p-1} L_k(\mathbf{x}_\Lambda) z_{\lambda\Lambda}^{k-n} \binom{k}{n}. \quad (2.10)$$

5. L2P from local expansion at \mathbf{x}_λ to particle at \mathbf{x}_i ,

$$u_i \approx \Re \left(\sum_{n=0}^{p-1} L_n(\mathbf{x}_\lambda) z_{i\lambda}^n \right). \quad (2.11)$$

For the P2M, M2M, and M2L kernels, the first term requires special treatment. The expansions are truncated at order p , so the accuracy of the FMM can be controlled by adjusting p . In our implementation, we do not construct any matrices during the calculation of these kernels. The P2P kernel is vectorized with the use of SIMD intrinsics, and the $\log()$ function is calculated using a polynomial fit for $\log_2(x)/(x-1)$ using SIMD.

2.4 FMM Communication Scheme

Partitioning of the FMM global tree structure and communication stencils are shown in Figure 2.4. The binary tree on the left side is a simplification of what is actually an octree in a 3-D FMM. Likewise, the schematics on the right are a 2-D representation of what is actually a 3-D grid structure. Each leaf of the global tree is a root of a local tree in a particular MPI process, where the global tree has L_{global} levels, and the local tree has L_{local} levels. Each process stores only the local tree, and communicates the halo region at each level of the local and global tree as shown in the red hatched region in the four illustrations on the right. The blue, green, and black lines indicate global cell boundaries, process boundaries, and local cell boundaries, respectively. The switch between local and global trees produces a change in the communication pattern, as revealed in the heat map in Figure 2.5.

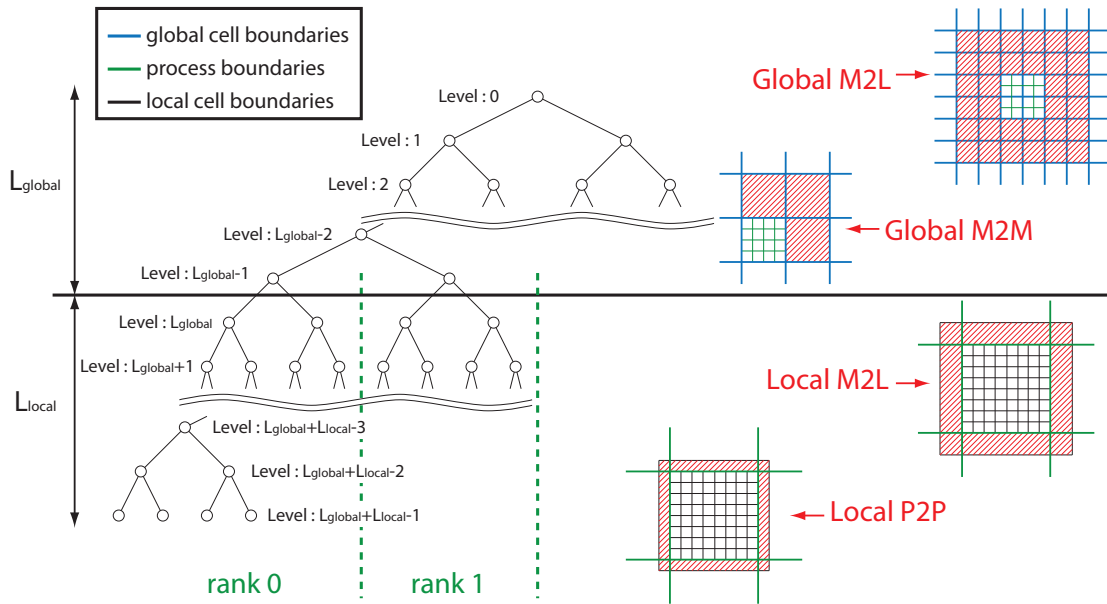


Figure 2.4: Splitting of the local and global tree in FMM.

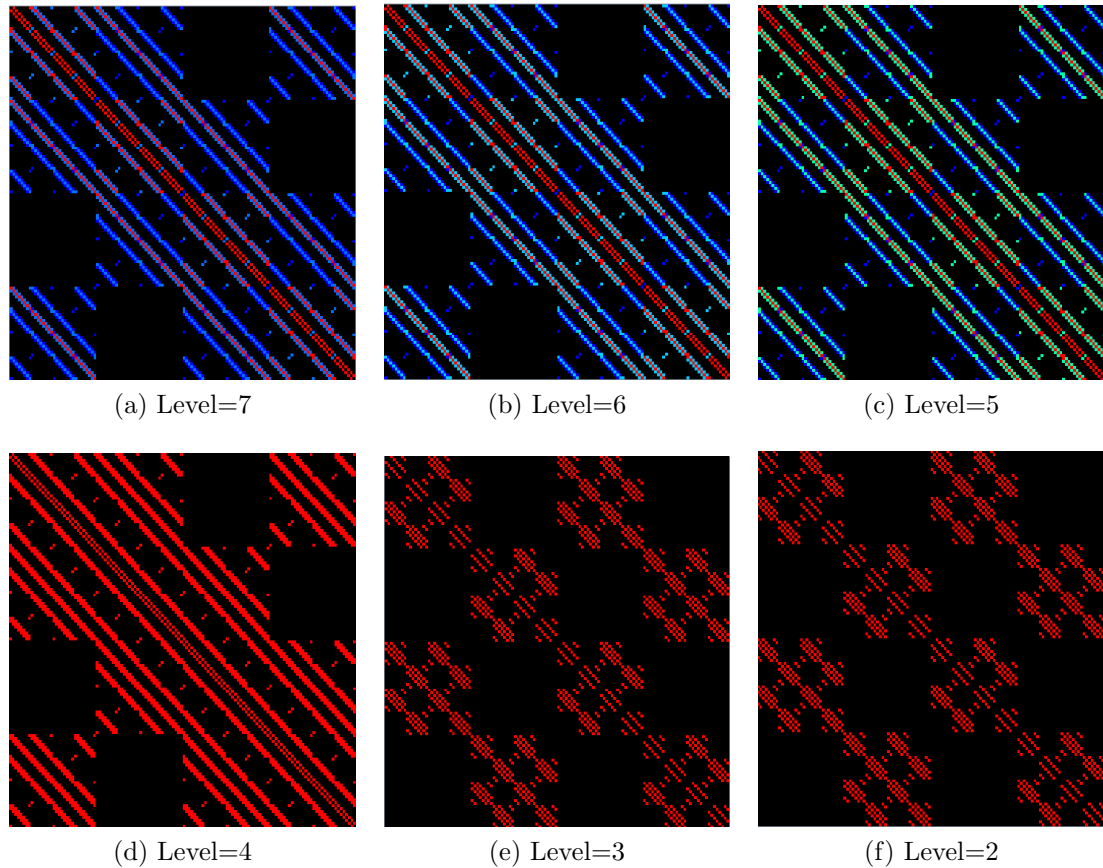


Figure 2.5: Heat maps for level-by-level communication patterns for the M2L phase of an FMM with $N=62,500$ per process using 128 processes. Areas of black indicate zero messages between processes, the peak communication volume is represented in red. In this example, the switch between global and local trees is between Level 3 and Level 4.

Chapter 3

A Performance Model for the Communication in FMMs for Spatially Uniform Distributions

Since the performance of a single-processor core has plateaued, future supercomputing performance will depend mainly on increases in system scale rather than improvements in single-processor performance. Processor counts are already over ten million for the top system. Modeling application performance at such scales is required to guide algorithmic choices and tunings on existing architectures and evaluate contemplated architectures. Since the performance of the FMM has a large impact on a wide variety of applications across a wide range of disciplines, it is important to understand the challenges that FMM implementations face on architectures with increased parallelism, as well as to predict and locate bottlenecks that might cause performance degradation. On future architectures where computation becomes relatively cheap compared to data movement, we anticipate that inter-node communication will become the bottleneck. The priority of the present chapter is the communication model of FMM.

To model the performance, we start with the baseline model, namely (α, β) model for communication, where α is the latency and β is the inverse bandwidth. Then, some penalties are added to the baseline model based on machine constraints. These

This chapter includes results from the previously published paper, “A performance model for the communication in fast multipole methods on high-performance computing platforms” by H. Ibeid, R. Yokota, and D. Keyes [52], ©2016 SAGE Publications, doi:10.1177/1094342016634819.

penalties include distance and reduced per-core bandwidth. Our performance model is related to universal communication features and can be applied regardless of local FMM implementation choices, core-scale machine characteristics that do not affect communication, and arithmetic workload associated with other aspects of the computation. Of course, the importance of communication as a bottleneck depends strongly on the cost of other tasks, but it is important to be able to evaluate communication costs as a component in an overall cost model. The Byte-count parameters in our model makes it adaptable to any of the various FMM implementations, while the penalties in our model are tunable to various architectures. We validate our performance model on four different architectures, Shaheen I (BG/P), Mira (BG/Q), Titan (Cray XK7), and Piz Dora (Cray XC40).

The chapter is organized as follows. Section 3.1 gives an overview of related work. Section 3.2 summarizes some performance challenges that face FMM on parallel machines. These challenges include massive parallelism and degradation due to inter-node communication. In Section 3.3, an exposition of the fast multipole method sufficiently detailed to expose communication properties is given. Section 3.4 describes our performance model. Experiments done to validate the performance models are provided in Section 3.5 and we conclude in Section 3.6.

3.1 Related Work

Performance modeling and characterization for understanding and predicting the performance of scientific applications on HPC platforms have been targeted by many related projects. For example, Clement and Quinn developed a performance prediction methodology through symbolic analysis of their source code [53]. Mendes and Reed focused on predicting scalability of an application program executing on a given parallel system [54]. Mendes proposed methodology to predict the performance scalability of data parallel applications on multi-computers based on information collected

at compile time [55]. The approach of combining computation and communication to obtain a general performance model is described by Snively *et al.* [56]. DeRose and Reed concentrate on tool development for performance analysis [57]. Performance models for implicit CFD codes have been considered in [58]. The efficiency of the spectral transform method on parallel computers has been evaluated by Foster [59]. Kerbyson *et al.* provide an analytical model for the application SAGE [60]. Performance models for AMG were developed by Gahvari *et al.* [22], who have also analysed the performance of AMG over a dragonfly network in [61]. Traditional evaluation of specific machines via benchmarking is presented by Worley [62].

Scaling FMM to higher and higher processor counts has been a popular topic [63, 64], while extensive study of single-node performance optimization, tuning, and analysis of FMM has also been of interest [65]. However, there has been little effort to model the inter-node communication of FMMs. Lashuk *et al.* derive the overall complexity of FMM on distributed memory heterogeneous architectures [66], but do not validate the model against the actual performance. The present work is based on the communication model for AMG [22], and extends their theory to FMM. To our knowledge, this is the first formal characterization of inter-node communication in FMM, which validates the model against actual measurements of communication time.

3.2 Performance Challenges

High performance computing systems have shown a sustained exponential growth with performance improvement of approximately 10x every 3.6 years as measured, for instance, by the Gordon Bell Prizes or the Top500 benchmark over the past 2.5 decades. This performance improvement comes at a cost in code complexity and introduces many challenges. Furthermore, the development of an exascale computing capability will cause significant and dramatic changes in computing hardware archi-

itecture relative to current petascale computers. In this section we present some of the challenges faced by FMMS to achieve good parallel performance on future exascale systems.

3.2.1 Trends in computer hardware

Computers consisting of nodes in the tens of thousands with cores per node in the hundreds have emerged as the most widely used high-performance computing platforms. These nodes communicate by sending messages through a network, which leads to lower scalability and less performance due to cores on a single node contending for access to the interconnect. We discuss multicore and manycore issues in more detail when presenting our performance models that take this into account.

3.2.2 Communication

Two types of costs in terms of time and energy are usually analyzed separately: computation (FLOPs) and communication (Bytes). Communication involves moving data between levels of a memory hierarchy in case of sequential algorithms and exchanging data between processors over a network in the case of parallel algorithms. Therefore, without considering overlap, the running time of an algorithm is the sum of three terms: the number of FLOPs times the time per flop, the number of words moved divided by the bandwidth (measured as words per unit time), and the number of messages times the latency. The last two terms determine the time consumed by communication. The time per flop is already an order of magnitude less than reciprocal bandwidth and latency and the gaps between computation and communication are growing exponentially with time. Communication performance models can guide development of algorithms to help reduce the communication.

3.3 FMM Communication Phases

As shown in Figure 2.4, our FMM uses a separate tree structure for the local and global tree. In order to construct a performance model for the communication in FMM, we estimate the amount of data that must be sent at each level of the hierarchy. Table 3.1 shows the number of cells that are sent, which correspond to the illustrations in Figure 2.4. L_{global} is the depth of the global tree, L_{local} is the depth of the local tree. We define N as the global number of particles, and P as the number of processes (MPI ranks). The global tree is constructed so that each MPI process is a leaf node in the global tree. Therefore, the depth of the global tree only depends on the number of processes P and not N . The depth of the global tree grows with $\log_8 P$, whereas the depth of the local tree grows with $\log_8(N/P)$. For the current calculations we are assuming a nearly uniform particle distribution (as in explicit solvent molecular dynamics; an application example is provided in Chapter 4) and therefore a full octree structure.

3.3.1 Global M2L

In Table 3.1 we show the number of cells to send per level and the total amount of communication for all levels. There are four types of communication in our FMM, which correspond to the four stages shown with the red hatching in Figure 2.4. The first is the ‘‘Global M2L’’ communication, which sends 26×8 cells at each level, as shown at the top right of Figure 2.4. The green lines are the process boundaries

Table 3.1: Amount of communication in FMM

	Cells to send / level	Total comm.
Global M2L	26×8	$\mathcal{O}(\log P)$
Global M2M	7	$\mathcal{O}(\log P)$
Local M2L	$(2^i + 4)^3 - 8^i$	$\mathcal{O}((N/P)^{2/3})$
Local P2P	$(2^i + 2)^3 - 8^i$	$\mathcal{O}((N/P)^{2/3})$

and the blue lines are the cell boundaries, which means one FMM cell belongs to many processes in the global tree. In order to avoid redundant communication, we index each process that shares a global cell and perform a one-to-one communication between the processes with matching indices only. In order to further reduce the communication, we select one process for a group of eight cells to do the communication. Therefore, the number of processes to communicate with (p_i) is always 26 and the number of cells to send is always 8 for every process and for every level in the global tree. In other words, for the “Global M2L” communication the message size and number of sends is constant regardless of N and P , and only the number of hops between the processes will increase depending on the network topology. On torus networks, we map the MPI ranks to the torus and synchronize the direction of the 26 one-to-one communications. The communication per level is $\mathcal{O}(1)$ and the number of levels in the global tree is $\mathcal{O}(\log P)$, so the total communication complexity for this stage is $\mathcal{O}(\log P)$ as shown in Table 3.1.

3.3.2 Global M2M

The second type of communication is the “Global M2M,” which sends 7 cells at each level, as shown in Figure 2.4. We use a similar technique to the “Global M2L” case to avoid redundant communication by pairing the MPI ranks for the one-to-one communication when many processes share the same global cell. The number of processes to communicate with is always seven and the number of cells to send is always one for every process and for every level in the global tree. Similar to the “Global M2L” case, only the number of hops during the one-to-one communication will increase, and the rate depends on the network topology. The communication per level is $\mathcal{O}(1)$ and the number of levels is $\mathcal{O}(\log P)$, so the total communication is $\mathcal{O}(\log P)$ for the “Global M2M” stage.

3.3.3 Local M2L

The third type of communication is the “Local M2L,” which is shown in the red hatching in the second picture from the bottom on the right side of Figure 2.4. The process boundaries shown in green are coarser than the local cell boundaries shown in black, which means that one process contains many cells, in contrast to the previous two communication types. In a full octree structure, we know that all cells are non-empty so we simply need to send two layers of halo cells for the M2L calculation at each level, as shown in Figure 2.4. Therefore, the number of processes to communicate with is always the 26 neighbors, and the number of cells to send depends on the level. At level i of the local tree, there are 2^i cells in each direction. Two layers of halo cells on each side will create a volume of $(2^i + 4)^3$ cells, and subtracting the center volume 8^i will give $(2^i + 4)^3 - 8^i$ as shown in Table 3.1. The leading term is $\mathcal{O}(4^i)$ since the 8^i term cancels out. Since the number of levels in the local tree grow as $\log_8(N/P)$ the communication complexity for the “Local M2L” is $\mathcal{O}(4^{\log_8(N/P)}) = \mathcal{O}((N/P)^{2/3})$. This can also be understood as the surface to volume ratio of the bottom two illustrations in Figure 2.4. Since N/P is constant for weak scaling and decreases for strong scaling, this part does not affect the asymptotic weak/strong scalability of the FMM.

3.3.4 Local P2P

The fourth type of communication in the FMM is the “Local P2P,” which is shown in the bottom picture on the right side of Figure 2.4. This communication only happens at the bottom level of the local tree. Similar analysis to the “Local M2L” stage shows that $(2^i + 2)^3 - 8^i$ cells must be sent, as shown in Table 3.1. In this case, i is exactly $\log_8(N/P)$ and we obtain the same asymptotic amount of communication of $\mathcal{O}((N/P)^{2/3})$. Similar to the “Local M2L,” this part does not affect the asymptotic weak/strong scalability of the FMM. However, the content of the data is different

from the previous three cases where the multipole expansion coefficients were being sent. In the P2P communication the coordinates and the charges of every particle that belongs to the cell must be sent. Therefore, the asymptotic constant of $\mathcal{O}(N/P)^{2/3}$ is typically much larger than that of the “Local M2L,” and this could be the dominant part of the communication time depending on the number of particles per leaf cell.

3.4 Modeling Performance

Performance modeling is a key ingredient in high performance computing. It has a great importance in the design, development and optimization of applications, architectures and communication systems. It also plays a crucial role in understanding important performance bottlenecks of complex systems. For this reason, performance models are used to analyze, predict, and calibrate performance for systems of interest. The tree-based communication of FMM is increasingly important in HPC applications, both of FMM itself and, for instance, of hierarchically low-rank (or “rank-structured”) matrices, which are under active development in theory and software. The application of a model of demonstrated relevance to one application to an entirely different application makes a statement about the value and general applicability of the model. In this section we develop a performance model to understand the performance of the communication in FMM through a phase-by-phase analysis based on four principal phases.

We start with a baseline model that is a combination of the latency and inverse bandwidth. We subsequently refine this baseline model to reach a more realistic model that is able to cover the relevant system architecture properties, with the exception that overlapping communication with computation is not considered in this work.

3.4.1 Baseline model $((\alpha, \beta)$ model)

To model interprocess communication, we begin with the basic (α, β) model, where α represents communication latency, where β is the send time per-Byte (inverse bandwidth). Using the basic model, a message send cost can be represented as

$$T_{\alpha-\beta} = \alpha + n\beta, \quad (3.1)$$

where n is the number of Bytes in the message.

This basic model describes the communication over an ideal architecture where the communication cost does not depend on processor locations or network traffic caused by many processors communicating at the same time [67]. For a more realistic architecture, a more detailed model is needed. For this reason, we add penalties to this basic model to take into account machine-specific performance issues. In particular, we consider communication distance, interconnection switching delay, limited bandwidth, and the effect of multiple cores on a single node contending for available resources.

3.4.2 Distance penalty $((\alpha, \beta, \gamma)$ model)

Following [22], we refine the assumption that distance between processors in interconnected networks does not have effect on communication time. To take into account the effect of distance we refine the baseline model according to the number of extra hops a message travels

$$T_{\alpha-\beta-\gamma} = \alpha + n\beta + (h - h_m)\gamma, \quad (3.2)$$

where h is the number of hops a message travels, h_m is the smallest possible number of hops a message can travel in the network, and γ is the delay per extra hop. If there

is no network contention and all messages travel with minimum number of hops, this distance penalty should have no effect.

3.4.3 Bandwidth penalty on β

The peak hardware bandwidth is rarely achieved in message passing. Therefore, we multiply β by B_{max}/B to incorporate the ratio between the peak hardware per-node bandwidth B_{max} and the effective bandwidth from the benchmark B .

$$T_{\beta-Penalty} = \alpha + n\beta\frac{B_{max}}{B} + (h - h_m)\gamma. \quad (3.3)$$

3.4.4 Multicore penalty on α or γ

Increasing the number of cores per node increases the data traffic between nodes, and could potentially result in congestion. Furthermore, larger number of cores per node introduces more noise caused by access to resources shared by multiple cores. To model these effects, we multiply α and/or γ by the number of active cores per node c . This model focuses on the worst case behavior where a machine's aggregate bandwidth could be exceeded by all cores communicating simultaneously. The resulting models are

$$T_{\alpha-Penalty} = c\alpha + n\beta + (h - h_m)\gamma, \quad (3.4)$$

$$T_{\gamma-Penalty} = \alpha + n\beta + c(h - h_m)\gamma. \quad (3.5)$$

3.5 Model Validation

3.5.1 Machine description

To validate our performance models we benchmark our FMM code on four different architectures; Shaheen I, Mira, Titan, and Piz Dora. Machine descriptions are

Table 3.2: Machine parameters for latency α , inverse bandwidth β , and distance penalty γ , on Shaheen I, Mira, Titan, and Piz Dora.

	Shaheen I	Mira	Titan	Piz Dora
α	4.12 μs	5.33 μs	1.67 μs	0.46 μs
β	2.14 ns	1.32 ns	1.62 ns	0.41 ns
γ	29.9 ns	134 ns	284 ns	484 ns

provided in Appendix A.

In order to obtain the machine parameters, the “b_eff” benchmark in the HPC Challenge suite [68] was used to determine the parameters α and β . We report the best-case latency and bandwidth measurements. To find the parameter γ , we followed the same procedure as Gahvari *et al.* [22]. Starting with the formulation of α as a function of the number of hops h

$$\alpha(h) = \alpha(h_m) + \gamma(h - h_m), \quad (3.6)$$

we set $\alpha(h_m)$ to be the measured value of α . If D is the diameter of the network, the maximum latency possible is

$$\alpha(D) = \alpha(h_m) + \gamma(D - h_m). \quad (3.7)$$

We use the maximum latency reported by the same benchmark we used to measure α as a value for $\alpha(D)$. Then

$$\gamma = \frac{\alpha(D) - \alpha(h_m)}{D - h_m}. \quad (3.8)$$

The machine parameters for Shaheen I, Mira, Titan, and Piz Dora are shown in Table 3.2. Note that our definition of β is defined as send time per Byte, whereas Gahvari *et al.* define their β as send time per element (8 Bytes).

3.5.2 Experimental setup

We run the FMM code for ten steps and measured the time spent on the communication for the “Global M2L” and “Local M2L” phases. The results are then divided by ten to get the average time spent at each level. The “Global M2M” phase is negligible and the “Local P2P” phase only occurs at the bottom level and is irrelevant to the scalability of the FMM, so we do not consider these two phases in the current analysis. We use the Laplace kernel in three dimensions with random distribution of particles in a cube. We use periodic boundary conditions so that there is no load imbalance at the edges of the domain. The number of MPI processes is varied between $P = \{128, 1024, 8192\}$, while the number of particles per process is kept constant at $N/P = 62,500$. On all machines we use the maximum number of cores on each node before increasing the number of nodes. Timings are measured with “gettimeofday()” after a “MPI_Barrier()” call. We use the default rank mapping to the nodes that the system provides.

Table 3.3 shows communication information and statistics when running the FMM on 128, 1024, and 8192 processes. “Level” is the level within the tree structure and goes from 0 to $L_{global} + L_{local} - 1$, where $L_{local} = 4$ for $N/P = 62,500$. Therefore, the bottom four levels in Table 3.3 (a), (b), and (c) belong to the local tree. The depth of the global tree L_{global} is 4, 5, and 6 for 128, 1024, and 8192 processes, respectively. “Cells” is the total number of cells at that level of the tree structure, which is simply 8^{Level} for a full octree. “Sends” is the number of processes to which sends. As mentioned in Section 3.3 we have developed a communication scheme that limits the number of sends to 26 regardless of the problem size, number of processes, or the level. “Bytes” is the aggregate data size that is sent by a given process at each level of the tree. As shown in Table 3.1, the number of cells for the “Global M2L” communication is 26×8 . For each cell we are sending 56 multipole expansion

Table 3.3: Statistics of the M2L communication.

(a) 128 Processes

Level	Cells	Sends	Bytes
0	1	0	0
1	8	0	0
2	64	26	46,592
3	512	26	46,592
4	4,096	26	46,592
5	32,768	26	100,352
6	262,144	26	272,384
7	2,097,152	26	874,496

(b) 1024 Processes

Level	Cells	Sends	Bytes
0	1	0	0
1	8	0	0
2	64	26	46,592
3	512	26	46,592
4	4,096	26	46,592
5	32,768	26	46,592
6	262,144	26	100,352
7	2,097,152	26	272,384
8	16,777,216	26	874,496

(c) 8192 Processes

Level	Cells	Sends	Bytes
0	1	0	0
1	8	0	0
2	64	26	46,592
3	512	26	46,592
4	4,096	26	46,592
5	32,768	26	46,592
6	262,144	26	46,592
7	2,097,152	26	100,352
8	16,777,216	26	272,384
9	134,217,728	26	874,496

coefficients in single precision (4 Bytes). Therefore, the total number of Bytes for the “Global M2L” phase is $26 \times 8 \times 56 \times 4 = 46,592$. We can see from Table 3.1 that the amount of cells involved in the “Local M2L” communication can be calculated by $(2^i + 4)^3 - 8^i$, where i is the level in the local tree (not the “Level” shown in Table 3.3).

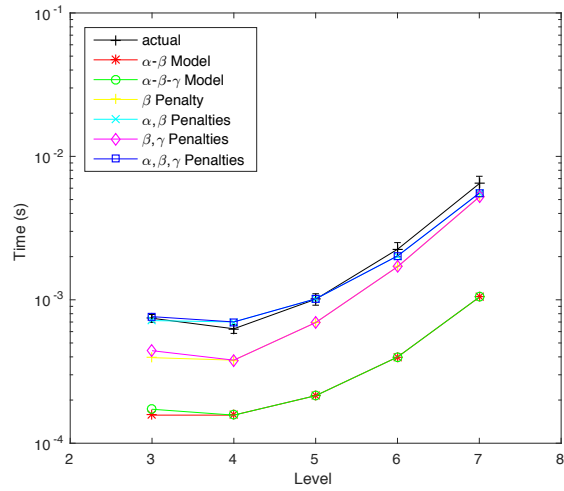
For example, for level one in the local tree, the amount of cells will be $(2^1 + 4)^3 - 8^1$ which is equivalent to 26×8 . This is why the “Bytes” is the same for the “Global M2L” and the first level of the “Local M2L” in Table 3.3.

3.5.3 Model validation

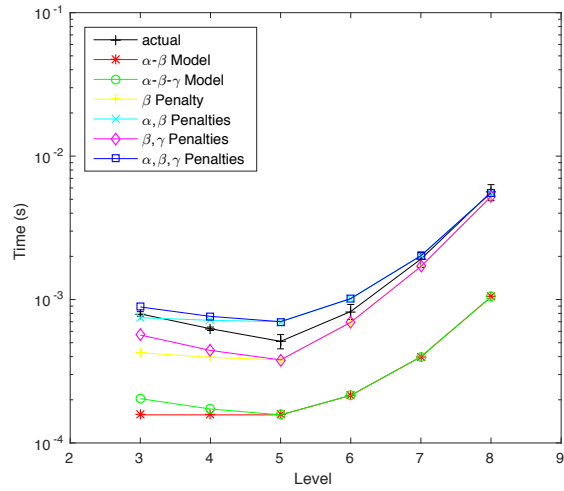
We compare the actual communication time for the M2L communication with our performance model on Shaheen I, Mira, Titan, and Piz Dora. We compare against same combination of models as in the multigrid study [22]. The combinations are:

1. Baseline model ($\alpha - \beta$ model)
2. With distance penalty ($\alpha - \beta - \gamma$ model)
3. With distance and bandwidth penalty (β penalty)
4. With distance and bandwidth penalty, plus multicore penalty on latency (α, β penalty)
5. With distance and bandwidth penalty, plus multicore penalty on distance (β, γ penalty)
6. With distance and bandwidth penalty, plus multicore penalty on latency and distance (α, β, γ penalty)

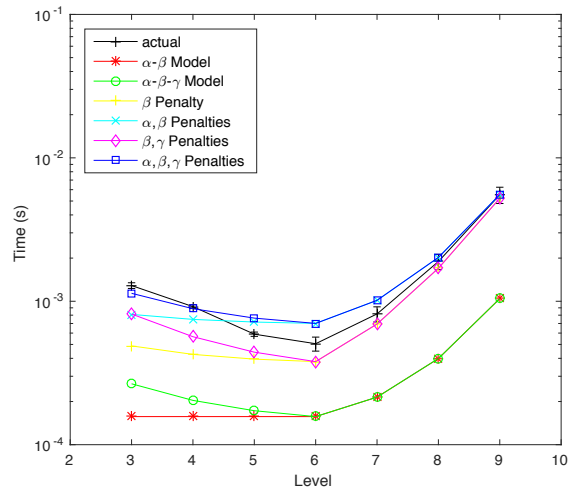
The results on Shaheen I are shown in Figure 3.1. The actual measured performance is shown as a black line, where an error bar is drawn according to the standard deviation in communication time among the different MPI ranks. By comparing the Bytes in Table 3.3 with the communication time in Figure 3.1, we see that the deepest four levels that belong to the “Local M2L” phase have a communication time that is proportional to the data size being sent. The main discrepancy in the models is



(a) 128 processes

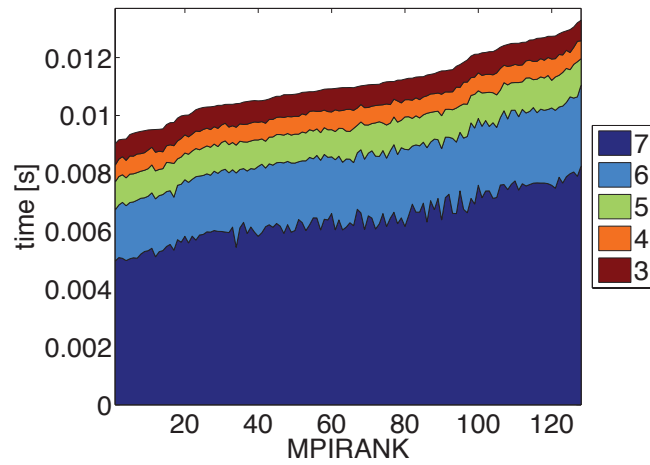


(b) 1024 processes

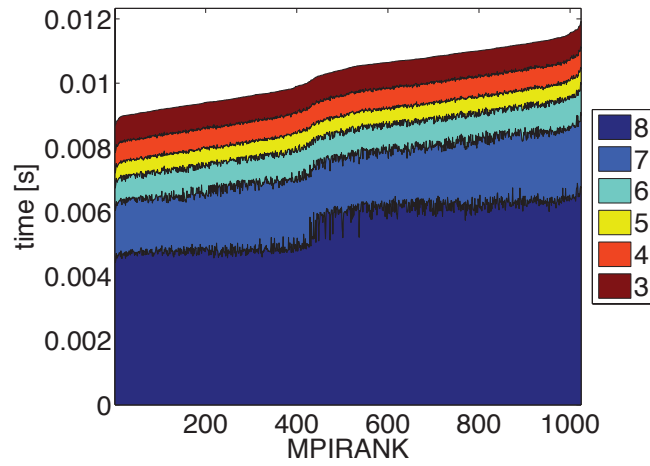


(c) 8192 processes

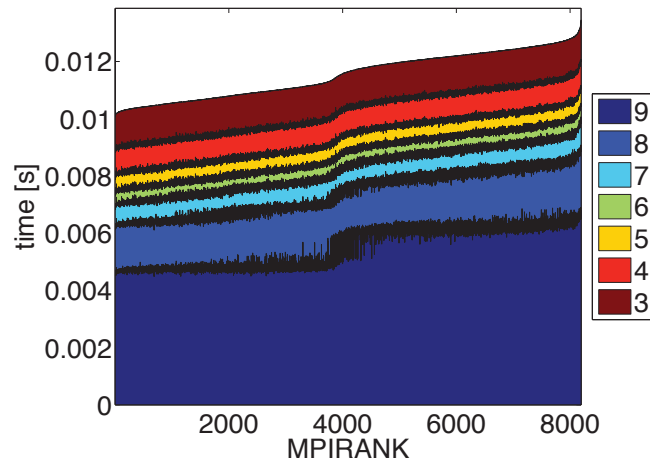
Figure 3.1: Performance model prediction and actual time for M2L communication phase on Shaheen I.



(a) 128 processes



(b) 1024 processes



(c) 8192 processes

Figure 3.2: Load balance of M2L communication phase on Shaheen I.

caused by the β penalty, for which the ratio between the theoretical injection bandwidth and the “b_eff” benchmark results is accounted for. The actual communication time agrees well with the models with α , β , and γ penalties.

For the shallow levels that belong to the “Global M2L” phase, the communication time increases as the level decreases/coarsens. Here, and in Figures 7, 8, and 9 to follow, the “Global M2L” levels are 3 in part (a), 3 and 4 in part (b), and 3, 4, and 5 in part (c). The reason for the increase can be understood by looking back at Figure 2.4, where the “Global M2L” is communicating with farther processes at coarser levels of the tree. Since we are mapping the geometric partitioning of the octree to the 3-D torus network of Shaheen I, the proximity in the octree directly translates to the proximity in the network. Therefore, even though the data size is constant for all levels in the “Global M2L” phase, the number of hops is larger, which accounts for switching delays and also network contention to some extent. This increases the communication time at coarser levels and the models that incorporate γ are able to predict this behavior.

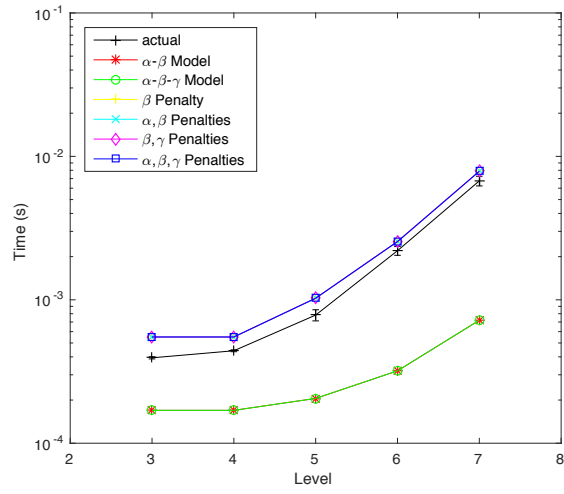
In Figure 3.2, the M2L communication time on Shaheen I is plotted against the MPI rank to show the load balance between the processes. Each color shows M2L communication at a different level of the tree structure, and the numbers in the legend represent the levels. The communication time of each level is stacked on top of the others so that the total height of the area plot represents the total M2L communication time shown in Figure 3.1. The MPI ranks are sorted according to the total M2L communication time for better visibility in the small differences between processes. As can be seen from the figure, the load balance is quite good. The imbalance seems to come from the finest levels, which are 7, 8, and 9 for 128, 1024, and 8192 processes, respectively.

The M2L communication time on Mira is plotted along with the six model predictions in Figure 3.3. Similarly to the runs on Shaheen I, the main difference in

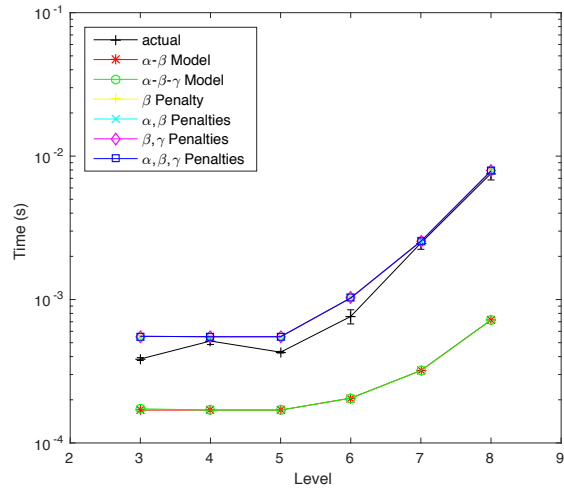
the model predictions is caused by the β penalty. We also see a discrepancy between the model predictions with and without the α penalty for the “Global M2L” phase (coarser levels). The multicore penalty is very small on the Bluegene/Q. This lack of multicore penalty has been observed in other applications where the use of hybrid OpenMP+MPI approach did not improve the performance over a flat MPI approach [69]. In contrast to the runs on Shaheen I, the communication time has a nearly flat profile for the “Global M2L” phase. This is because the 5-D torus network minimizes the number of hops and network contention so the degradation at coarse levels of the tree is minimal. Far nodes in the octree are not so far in the Bluegene/Q network topology.

Figure 3.4 shows the M2L communication time on Titan along with the six model predictions. Similarly to the previous two cases, the difference between the model predictions is mainly due to the correction for the inverse bandwidth. This difference in the theoretical injection bandwidth and measured effective bandwidth seems to have the largest effect on all three architectures. What is different from the previous two cases is the large jump in the actual communication time for the “Global M2L” phase. For example, for the 8192 process run level 5 is taking about 10 times more than level 6 even though the message size is 46,592 Bytes for both cases. The γ term in the current performance models anticipates such behavior. The error bars in the actual timings are quite large, which indicates that there is a large load imbalance compared to the previous two systems. The concave-convex switch at level 5 in 8(b) is not well predicted by the models, but the more refined models do pick it up at level 6 in 8(c). Though a good match between the measurements and simple models is not realized for M2L at all granularities on Titan, performance trends are generally well predicted.

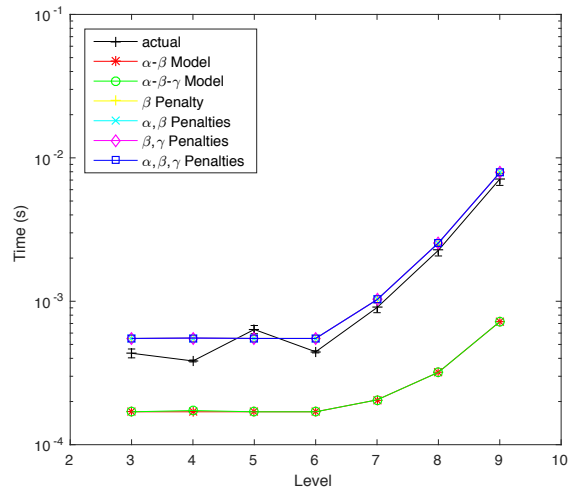
The M2L communication time on Piz Dora is plotted along with the six model predictions in Figure 3.5. In the case of 128 processes, the best fitting model is the



(a) 128 processes

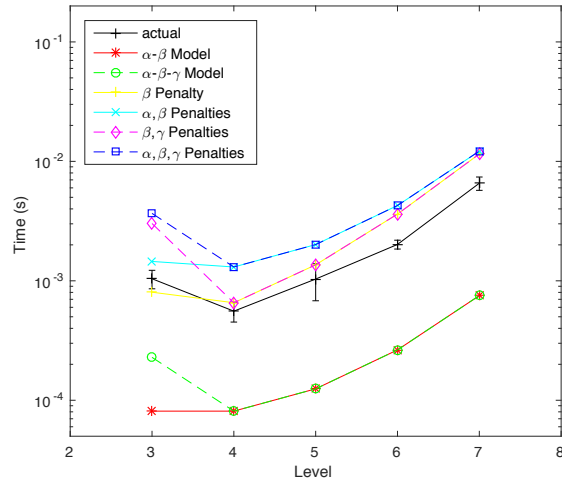


(b) 1024 processes

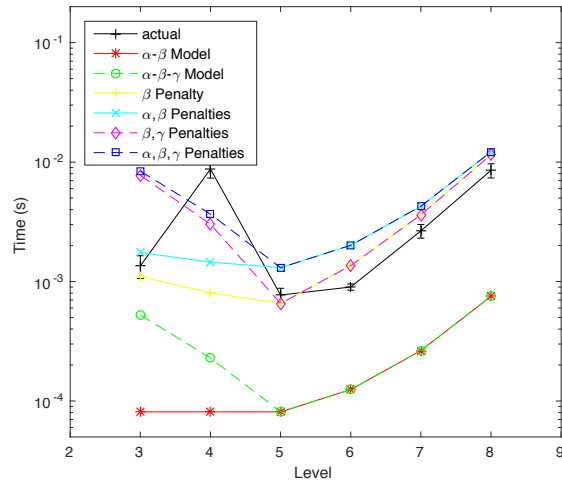


(c) 8192 processes

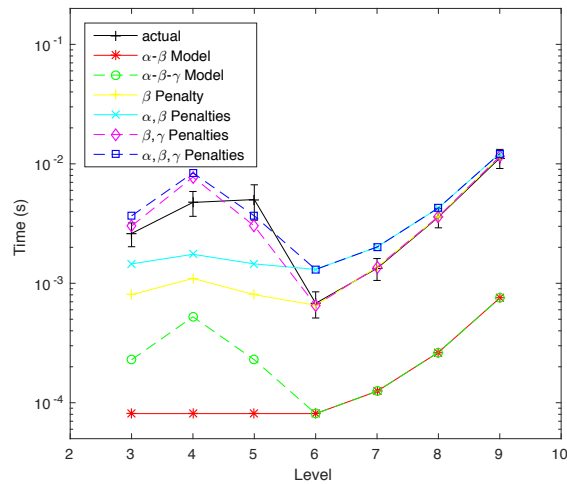
Figure 3.3: Performance model prediction and actual time for M2L communication phase on Mira.



(a) 128 processes

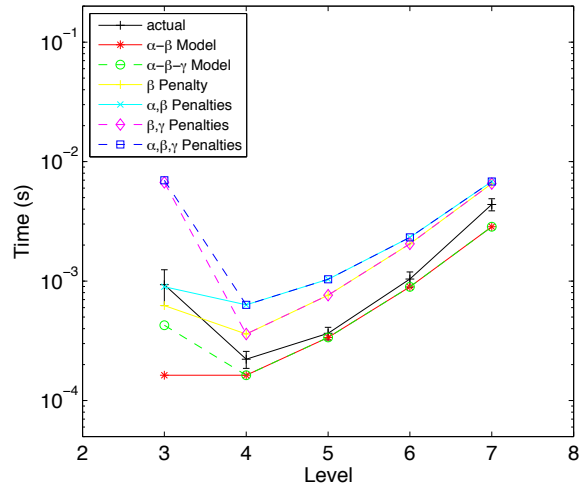


(b) 1024 processes

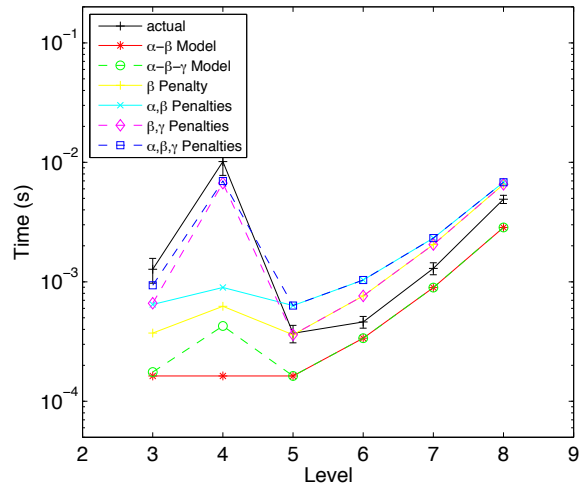


(c) 8192 processes

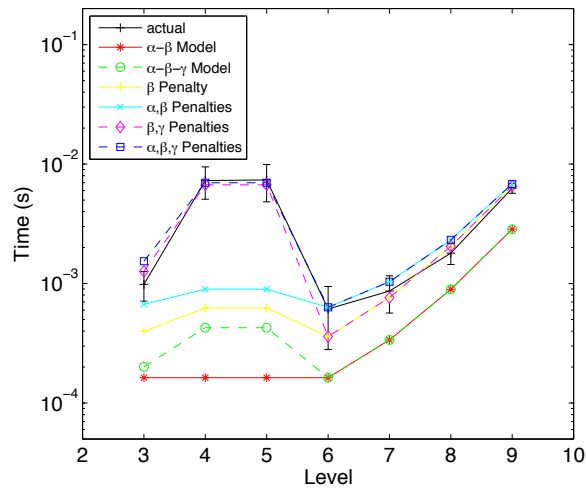
Figure 3.4: Performance model prediction and actual time for M2L communication phase on Titan.



(a) 128 processes



(b) 1024 processes



(c) 8192 processes

Figure 3.5: Performance model prediction and actual time for M2L communication phase on Piz Dora.

baseline model plus only the distance penalty. Increasing the number of processes increases the possibility of contention and makes the model with all penalties the best fitting model. Similar to the runs on Titan, there is a large jump in the actual communication time for the “Global M2L” phase with even worse load balancing suggested by the large error bars. The performance model is able to predict the poor performance at the coarse levels.

3.6 Conclusions

The goal of this chapter is to model the global communication of the FMM, to be able to anticipate challenges on future exascale machines. To improve model fidelity, we consider penalties based on machine constraints including distance effects, reduced per core bandwidth, and the number of cores per node. We observe a good match between the (α, β, γ) model with multicore penalties and the actual communication time. The discrepancy between the other models means that all components of the model; latency (α), bandwidth (β), hops (γ), and multicore penalty must be taken into account when predicting the communication performance of FMM.

In our benchmark tests, we compare the performance models with measurements for the M2L communication, since this is the dominant part of the FMM communication. Our observations are consistent with those of the studies by Gahvari *et al.* [22], where the performance of an algebraic multigrid method is analyzed using the same model. The measurements fall within the bounds of the performance models, and match best with the model where latency, bandwidth, hops, and multicore penalty are all taken into account.

The ultimate communication model is predictive in an absolute sense; however, on complex systems, this objective is often out of reach, or of a difficulty out of proportion to its benefit when there exists a simpler model that is inexpensive and sufficient to guide coding decisions leading to improved scaling. The current model

provides such guidance.

Looking into the future, we will most likely be seeing more network topologies with larger diameter (more hops). Large radix networks seem to be the current trend, but with the exponential increase in the core count the increase of the network diameter is unavoidable. Our communication model with the distance penalty is able to capture the increase in communication time at the coarse levels of the FMM communication on Titan's torus network. This should allow predicting the communication bottlenecks on future networks with larger diameter.

The performance model herein is applicable to evolving heterogeneous systems, such as GPUs or Xeon Phi. This is because the accelerators and coprocessors affect the per-node computation but not the inter-node communication. Nor is the model affected by the on-node computational performance of FMM, as long as the accelerators and coprocessors are not using more than one MPI process, which is the optimal way to use the current generation of such hardware.

Chapter 4

Molecular Dynamics Simulation of Uniformly Distributed Particles Using the FMM

The human body is made up of an estimated 60 trillion cells (with estimates varying by counting conventions) that consist of various biomolecules. Each biomolecule plays an important role in biological activities; the functions of biomolecules are realized by their inherent structures and dynamics. Recent experimental techniques such as X-ray crystallographic analysis and nuclear magnetic resonance (NMR) spectroscopy enable us to measure both the structure and fluctuation of biomolecules. Computational approaches have also been attempted to study the dynamic and thermodynamic properties of biomolecules. Among these methods, molecular dynamics (MD) simulation is the most prevalent. In MD simulations, atoms are treated as a point mass, and their orbits are calculated using Newton's equation of motion. A typical time step size Δt of the simulation is in the order of femtoseconds, while sampling over microseconds or milliseconds is sometimes necessary to observe events of interest. Therefore, MD simulations require large computational power, and they are often used as HPC benchmarks. In this section, we report large-scale MD simulations of proteins in a crowded environment on the massively parallel "K computer".

In typical MD simulations, proteins are immersed in solvents or lipids. This

This chapter includes results from the previously published paper, "Petascale molecular dynamics simulation using the fast multipole method on K computer" by Y. Ohno, R. Yokota, H. Koyama, G. Morimoto, A. Hasegawa, G. Masumoto, N. Okimoto, Y. Hirano, H. Ibeid, T. Narumi, and M. Taiji [70], ©2014 Elsevier, doi:<http://dx.doi.org/10.1016/j.cpc.2014.06.004>.

corresponds to an ideal environment in a test tube – *in vitro*. On the other hand, living cells are crowded because macromolecules comprise $\sim 30\%$ of their molecular weight [71, 72, 73]. An actual intracellular environment *in vivo* is extremely different from an *in vitro* environment [74]. In order to understand the structure and dynamics of biomolecules in a living cell, systems with crowded environments have been studied using crowding agents [72].

Previous computational studies on macromolecular crowding have used reduced models because of the high computational costs of all-atom simulations. However, explicit solvent models are known to be essential for detailed analyses of protein dynamics. The precise treatment of protein flexibility is also essential for the study of protein fluctuations. In this section, we aim to clarify the effects of these important factors using a cutting-edge algorithm on a state-of-the-art high-performance computer. The K computer system has 82,944 processors that consist of 663,552 cores, and a theoretical peak performance of 10.6 PFLOPs. Therefore, we needed to extend the parallel efficiency up to nearly a million cores. To achieve this goal, we have developed a scalable MD simulation code for biomolecular systems, using a FMM.

The major bottleneck for the scalability of MD simulations is the long range force calculation that is commonly performed by a particle-mesh Ewald method (PME) [75]. The 3-D FFT in PME prevents the simulation from scaling to hundreds of thousands of cores. The FMM, on the other hand, is known to scale to the full granularity of the largest supercomputers of today [15]. This difference comes from the difference in the communication pattern of FFT and FMM. A 3-D FFT requires a global transpose of the data. On P processes the optimal communication complexity is $\mathcal{O}(\sqrt{P})$ when 2-D decomposition (pencil decomposition) is used. The FMM has $\mathcal{O}(\log P)$ communication complexity because the volume of communication decays logarithmically with the distance. Furthermore, our present FMM uses a hierarchical local communication scheme, which eliminates the use of “MPI_Alltoallv” type communications.

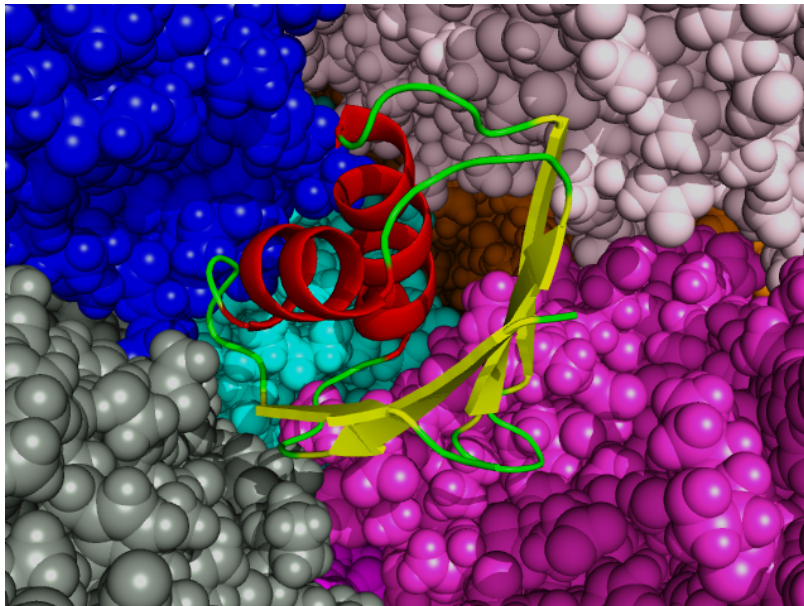


Figure 4.1: Vicinal area of TTHA for the *in vivo* system. The TTHA molecule is drawn using the ribbon model, and ovalbumin molecules are drawn using the space-filled model. Each ovalbumin molecule is colored differently.

This communication scheme maps well to the 6-D torus network of the K computer.

For measurement of performance of MD simulation on K computer, we performed all-atom classical MD simulations of TTHA1718 (a protein from *Thermus thermophilus* HB8, hereafter TTHA) using ovalbumins (Figure 4.1) as crowders that mimic the conditions of a living cell [76].

This chapter is organized as follows. In section 4.1 we explain the computational methods. Following that, we present the benchmark results of the large-scale MD simulations in section 4.2. The comparison between results of FMM and PME on macromolecular crowding are presented in section 4.3. Finally, conclusions are given in section 4.4.

4.1 Computational Methods

In a classical MD simulation, every atom's position is updated at each discrete time step. When the number of atoms is N_{atom} , the computational cost should be propor-

tional to the product of N_{atom} and the number of time steps if a cutoff is applied for long-range electrostatics. Therefore, our goal is to design software that can scale up to large N_{atom} and N_{node} .

4.1.1 Basic MD functions

MD simulation is composed of evaluation of forces between atoms and the integration of their trajectories. The following equations express the exact MD equations of our computations:

$$\begin{aligned}
 m_i \frac{d^2 \mathbf{r}_i}{dt^2} &= -\nabla U(\mathbf{r}_1, \dots, \mathbf{r}_{N_{atom}}) & (4.1) \\
 U(\{\mathbf{r}_i\}) &= \sum_{\text{bond}} \frac{1}{2} k_b (r - r_0)^2 + \sum_{\text{angle}} \frac{1}{2} k_a (\theta - \theta_0)^2 \\
 &+ \sum_{\text{torsions}} \frac{1}{2} V_n [1 + \cos(n\omega - \gamma)] \\
 &+ \sum_{|r_{ij}| < R_c} \left\{ \epsilon_{ij} \left[\left(\frac{\sigma_{ij}}{r_{ij}} \right)^{12} - 2 \left(\frac{\sigma_{ij}}{r_{ij}} \right)^6 \right] \right. \\
 &\quad \left. + \frac{q_i q_j}{4\pi\epsilon_0 r_{ij}} + \Psi(r_{ij}) \right\} & (4.2)
 \end{aligned}$$

where m_i and \mathbf{r}_i represent the mass and position of the i th particle, respectively, $r_{ij} = |\mathbf{r}_i - \mathbf{r}_j|$, and a smoothed 3rd degree polynomial, Ψ , is the GROMACS shift function [77]. In (4.2), the first three terms express bonded forces, and the last summation describes nonbonded forces. Note that bonded forces such as bond stretching, angle, and torsion refer only to several topologically connected atoms; therefore, their computational costs are relatively lower than that of nonbonded forces.

We have implemented a symplectic time integration (Verlet method) with SHAKE [78]/RATTLE [79] and SETTLE [80] holonomic constraint algorithms, and several well-established numerical algorithms to solve the above equations efficiently for both short-range and long-range interactions. In this work, we use a shift function for the

cutoff method for short-range contributions, and FMM for the long-range contributions.

4.1.2 Non-bonded short range force

Although almost all algorithms in MD simulations have linear complexity, nonbonded forces such as Coulomb and Lennard-Jones potentials are dominant because of the large number of interacting atom companions. When the cutoff method is used for these nonbonded force calculations, the number of interacting atoms depends on the cutoff radius. For example, a typical biomolecular circumstance involves 0.1 atoms in one cubic Angstrom; a spherical volume of 10 Å radius includes approximately 419 atoms. For the efficient calculation of the nonbonded forces, we organize a cell index and a “pair list,” which is a list of atom pairs within the cutoff radius. All atoms are distributed to “cells,” which are cubic space decompositions of the simulation space. For each individual cell, we build a list of cells within the reach of the cutoff radius from the cell. Since the pairing atoms are located only in the cells in the list, we can limit the search volume to a region with a fixed volume of $\{2 \times (\text{cutoff radius} + \text{cell size})\}^3$.

If pair lists are updated every time step, the total calculation cost is the same as the cost using only the cell index without the pair list. To reduce the cost, our code updates the pair list once every 40 steps. For the generation of the pair list in this case, we must use a longer distance than the cutoff radius because the list must include atoms that may move inside the cutoff sphere in the next 40 steps. The fraction of pairs within the cutoff radius in the pair list depends on this pair-list margin and cutoff. We use 28 Å cutoff radius and 2 Å margins in our code, and the fraction is 0.81.

Despite the margin, the particles may jump inside the cutoff radius during the 40-step interval. In such cases, our code rolls back to the previous update step and

recalculates with a decreased interval. In our 100,000-step simulation with 180 million atoms at a 28 Å cutoff, we observed 16 occurrences of such events, which correspond to only 0.6% redundant calculation. Because these events are so rare, the redundant calculations have no serious impact on performance.

4.1.3 Non-bonded long-range force

Particle mesh Ewald (PME) [75] and its variants such as Gaussian splitting Ewald (GSE) [81] are efficient methods for calculating lattice sums with periodic boundary conditions. Although, these lattice sum methods are accurate for periodic systems such as crystals, they suffer from long-range correlation artifacts and anisotropy effects for noncrystalline systems such as cellular environments [82, 83, 84]. To avoid such artifacts, variation of cutoff methods that include long-range effects have been proposed. For example, isotropic periodic sum (IPS) [85], which sums over the isotropic periodic images, and the Wolf method or zero-dipole summation [86, 87], which use potential damping under charge neutral or zero-dipole conditions. The accuracy of such cutoff methods depend on the uniformity of the far field, which means that the cutoff length must be larger than the scale of the structure. Therefore, a cutoff method with long cutoff length can be an effective tool for MD simulation of large protein structures. Especially when arithmetic is becoming cheaper relative to bandwidth, the selection of the optimal algorithm requires careful consideration.

Another disadvantage of PME is its scalability, since it requires a large amount of global communication due to the nature of the FFT. It is difficult to achieve even weak scaling (not to mention strong scaling) for a large number of CPUs with full electrostatics. Note that several studies (e.g., [88]) have attempted to improve the network performance of long-range force calculations. On the other hand, fast multipole methods can reduce the amount of communication in long-range interactions from $\mathcal{O}(\sqrt{P})$ to $\mathcal{O}(\log P)$, where P is the number of processes.

4.1.4 Periodic fast multipole method

In the present simulations, an FMM with periodic boundary conditions [89] is used to calculate the long range force. An FMM can approximate the long-range forces using multipole expansions and local expansions, and is able to calculate the interaction of N bodies in $\mathcal{O}(N)$ time.

When the distribution of bodies is not uniform, the tree structure is unbalanced and parallelization of the FMM becomes a non-trivial task. However, for molecular dynamics simulations, where water molecules uniformly fill the entire domain, the FMM can use a full tree structure. This has many implications when constructing a highly parallel FMM. First, the data structure of a full tree is much simpler than that of an adaptive tree. Since one can assume that all nodes of the octree are full, multipole-to-local translation stencils are identical for every cell. The periodic boundary condition is another ingredient, which helps create an entirely homogeneous translation stencil even near the boundaries of the domain. This enables optimization techniques that exploit the symmetry of the translation stencils and precalculation of the multipole-to-local translation matrix. Second, the communication on distributed memory machines is much simpler for a full tree since the data structures on remote nodes are known. Again, the periodic boundary condition also helps to create a perfect load balance, since the boundary cells have a full stencil. Third, a full tree means that there is no need to reconstruct it once it is built on the first time step. It is only a matter of updating which atoms belong to which leaf cells. This is also a significant advantage when developing a highly parallel FMM code.

4.2 Performance Benchmarks

This section presents the results of the performance benchmarks of macromolecular simulations on the K computer (machine description is provided in Appendix A). It

Table 4.1: Performance on the main loop and the kernel loop for 418,707 atom simulation by 64 process

Main loop	
Time consumption (ms/step)	114.02
Performance (MFLOPs)	3,551,808
Efficiency	0.4336
Kernel loop	
Time consumption (ms/step)	77.36
Performance (MFLOPs)	5,180,754
Efficiency	0.632
SIMD ratio ¹	0.572
FLOP counts per pair of atom ²	108.3

¹Number of SIMD instructions / Number of all instructions

²Evaluated by the performance counter and estimated number of atom pairs

is well known that the force calculation dominates the cost of an MD simulation. In the results of our benchmark, the force calculation and communication cost represent 83% and 14% of the CPU time, respectively. Here, we first provide the CPU core performance of the force calculation kernel and then describe the overall performance of macromolecular *in vitro* system simulations on the parallel processors. The *in vitro* system contains 418,707 atoms (4 proteins and 137,977 water molecules) in a $(163 \text{ \AA})^3$ cube. We simply replicate this system for larger runs. The details of this molecular system are described in Section 4.3.

4.2.1 Efficiency of the force calculation kernel

The CPU core performance was monitored by a hardware performance counter and reported using a profiler tool. Table 4.1 shows the result of the *in vitro* simulation model on 64 nodes (64 CPU, 512 cores). The scientific descriptions of these models are provided in the subsequent section. We have measured execution time with and without performance counters and confirmed that the overhead of the performance counter is less than 1.5%. Thus, we use the number of FLOP counts based on the performance counter as references. In Table 4.1, we summarize the performance

Table 4.2: Conditions of the simulation for the peak performance

Model	432 <i>in vitro</i>
Number of atoms (N_{atom})	522,546,336
Cutoff radius	28 Å + 2 Å margin
Number of pairs per atom	8,835 (+2,031 in margin)
FLOP counts for 1,000 steps	510,481,566,661 MFLOP
Spatial size Å ³	1956 × 2119 × 1304
Potential energy	every time step
Topology of computation node	48 × 52 × 32
Number of nodes	79,872
Number of cores	638,976
Theoretical peak performance	10.223616 PFLOPs
Calculation time for 1,000 step	116.357 sec
Sustained performance	4.387 PFLOPs
Efficiency	0.429

count. The force calculation kernel of the code ran at 63% of the theoretical peak, where 57% of the instructions were SIMD vectorized. By analyzing the source code of the kernel loop, we obtained (62 + division + square-root) of FLOP counts per atom-pair calculation. This is consistent with the performance of ~ 110 FLOP counts, as seen in Table 4.1.

4.2.2 Sustained performance of the large-scale simulations

Here, we report the sustained performance of the parallel runs. The wall-clock time of each block of the program is measured by inserting a Unix system call “gettimeofday()”. We have confirmed that the overhead of adding this system call is less than 0.5%. Using 523 millions atoms, we achieved sustained performance of 4.387 PFLOPs on 79,872 nodes, which is 42.9% of the theoretical peak of the 10.2 PFLOPs K computer. The parameters of this simulation are summarized in Table 4.2.

Figure 4.2 shows the time spent in blocks per step for the weak scaling of the *in vitro* system (418,707 atoms/64 nodes). These times did not increase because almost all communications were local, except for a small number of “all gather” calls for the temperature calculation. Parallel efficiency, which is defined by $T(64)/T(79,872)$, was

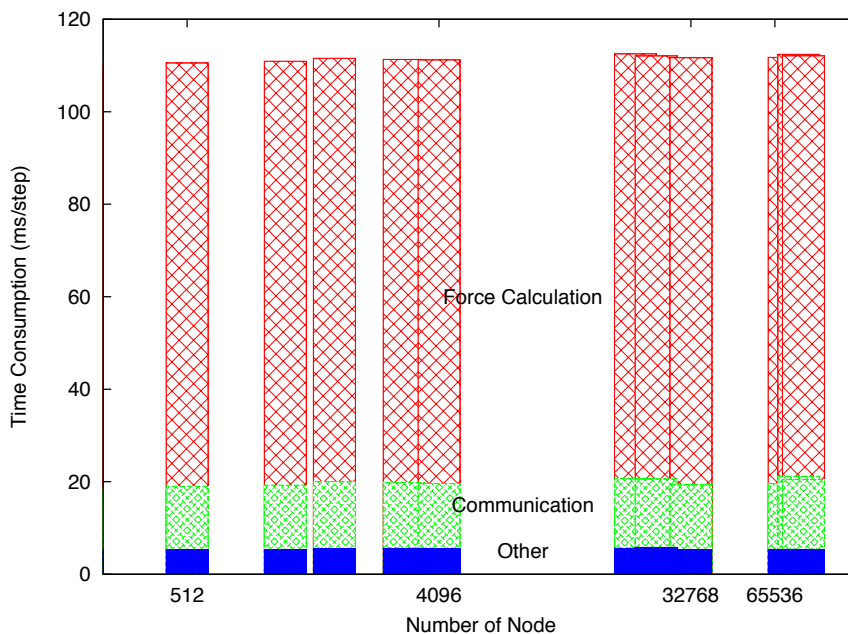


Figure 4.2: Weak scaling for 6,542 atoms/node. Constant wall clock times/step show the perfect scaling. Ratios of force calculation/communication/others are also constant, and force calculation is dominant.

above 0.96.

Figure 4.3 shows the operation time of a strong scaling *in vitro* system against number of atoms per node. Time spent on the force calculation decreased proportionally to the number of nodes N_{node} , as expected. Theoretically, the communication time is proportional to $(N_{node})^{-2/3}$, since the import volume per node is proportional to a ratio of a surface and a volume of a region assigned to each node. Though, the points of the measurements are not enough to reproduce the exact theoretical curve, we see the weak decrease of the communication time in the case of a small number of nodes. Beyond 6,400 atoms/node, we observed a saturation, which was caused by an increase in communication targets per node. In the case of 100 atoms/node, the side length of the region assigned to each node was about 10 Å, which was 1/3 of the cutoff radius (28 Å). Thus, each node had to communicate with ± 3 nodes in each direction; ignoring the corner cells. In this case, the number of communication target nodes becomes 310.

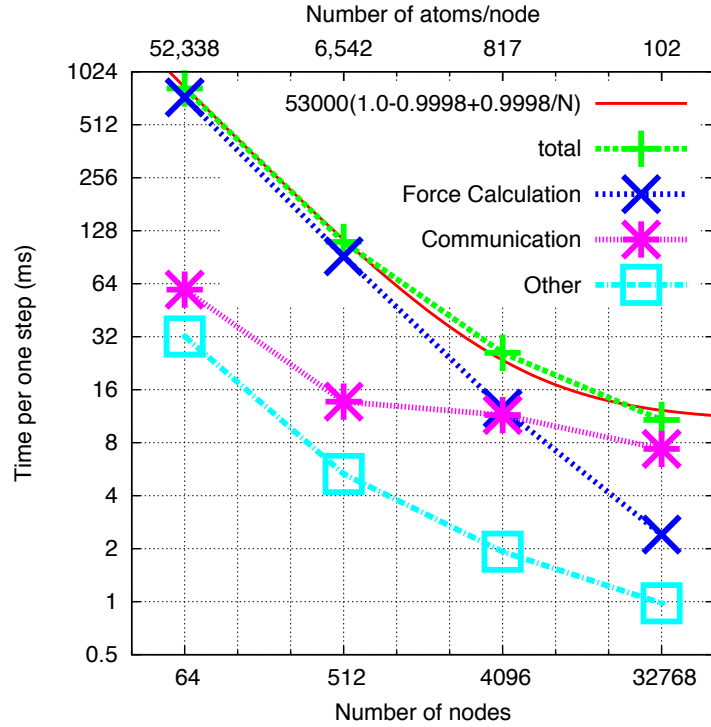


Figure 4.3: Strong scaling to number of atoms per node. The red curve shows Amdahl's law.

The total time of Amdahl's law [90], $T_1(1 - \mathcal{P} + \mathcal{P}/N_{node})$, is shown in Figure 4.3, where T_1 is the serial execution time, $1 - \mathcal{P}$ is the fraction of the serial part, and N_{node} is the number of nodes. The estimated parallelization rate \mathcal{P} is 0.9998. Note that the value of \mathcal{P} improves as the number of atoms per node increase. At this number of atoms, the rate \mathcal{P} is not sufficient for an 80,000-node system. The result shows that 50% efficiency of the ideal scaling was obtained at around 10,000 nodes with 8.4 million atoms. Thus, strong scaling was achieved up to 800 atoms per node. This means a system with 64 million atoms is sufficient to achieve 50% parallel efficiency for the full K computer system with 80,000 nodes.

4.2.3 Load balance

We now examine the workload balance among the nodes in the benchmark calculations. Figure 4.4 shows the calculation load of each node measured by FLOP counts

for the *in vitro* and *in vivo* simulations, whose average numbers of atoms per node were 6,542 and 6,224, respectively. This figure shows a relatively large load imbalance in the *in vivo* simulation, compared to the *in vitro* one. The maximum load for the *in vitro* system exceeds the minimum by 5.2%, while the maximum for the *in vivo* system exceeds the minimum by 12.7%. Note that since the measurements are based on the FLOP counts, they do not reflect the fluctuation in the execution efficiency. As we have noted previously, most of the computational workload in MD simulations is due to the force calculation, which is organized by the atom pair lists. Because of the cutoff method used, the number of atom pairs per node depends on the local particle density. Therefore, the performance number on each node also depends on the local density. In the *in vitro* system, 98.9% of atoms belong to the water molecules, which are distributed in a spatially uniform manner. Thus, the particles in the *in vitro* system are distributed more uniformly than those in the *in vivo* system, which contains a large number of proteins. This results in a better load balance in the *in vitro* system.

4.2.4 Performance of the FMM

Weak scaling of FMM part of the MD simulation for 6,542 atoms/node is shown in Figure 4.5. The execution time of the local FMM kernels remains constant throughout the entire range from $P = 64$ to $P = 24,576$. The time of all communications seem to roughly follow the upper bound of $\log P$, and the absolute time spent on communication is approximately 1 millisecond per step. Taking into account the topology of the TOFU network, the communication at the coarse level of the global tree will have a latency proportional to the number of hops $\mathcal{O}(\sqrt[3]{P})$. Therefore, the communication should ultimately scale as $\mathcal{O}(\sqrt[3]{P})$ instead of $\mathcal{O}(\log P)$ on the TOFU's 3-D torus network. Extrapolating these results to the full system of the K computer

Note that the TOFU is not a full 6-D torus, and 3 of the dimensions are smaller than the others, and are mainly designed for fault tolerance.

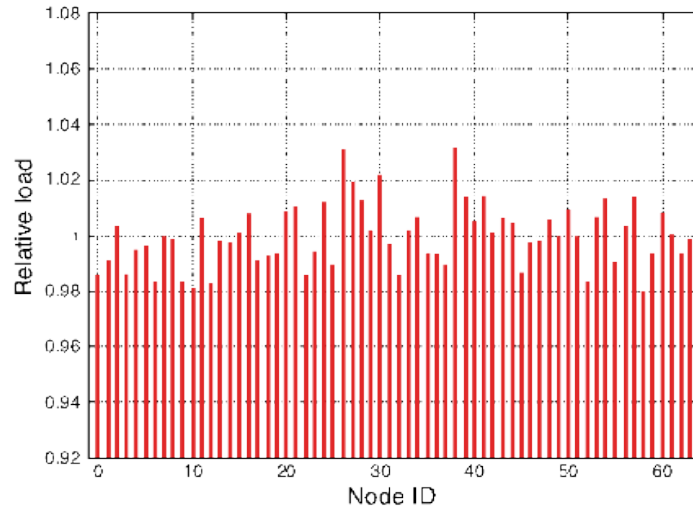
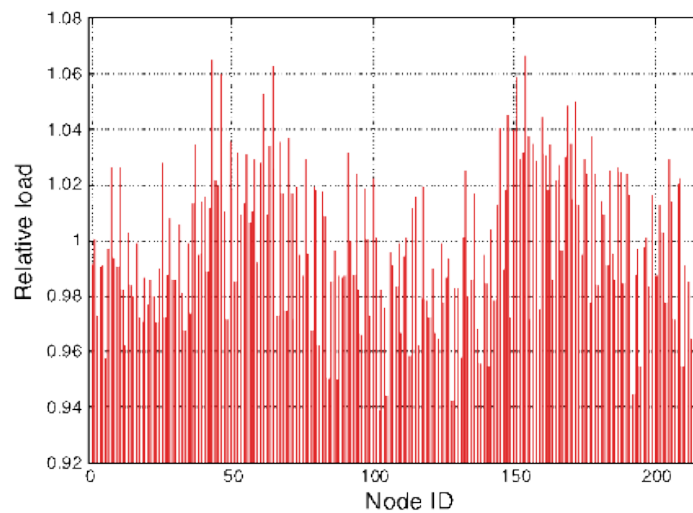
(a) *in vitro*(b) *in vivo*

Figure 4.4: Computational workload (FLOP count) in each node.

suggests that a 540 million atom simulation is possible in a few milliseconds per step, although we did not have enough run time on the full system to conduct such a run. In terms of time-to-solution of large MD simulations, we were able to calculate 210 million atoms in 14 milliseconds per step on 4,096 nodes of the K computer. Total calculation times were 43.42 ms/step and 47.90 ms/step using 64 nodes and 24,576 nodes. Parallel efficiency ($T(64)/T(24,576)$) was 0.906.

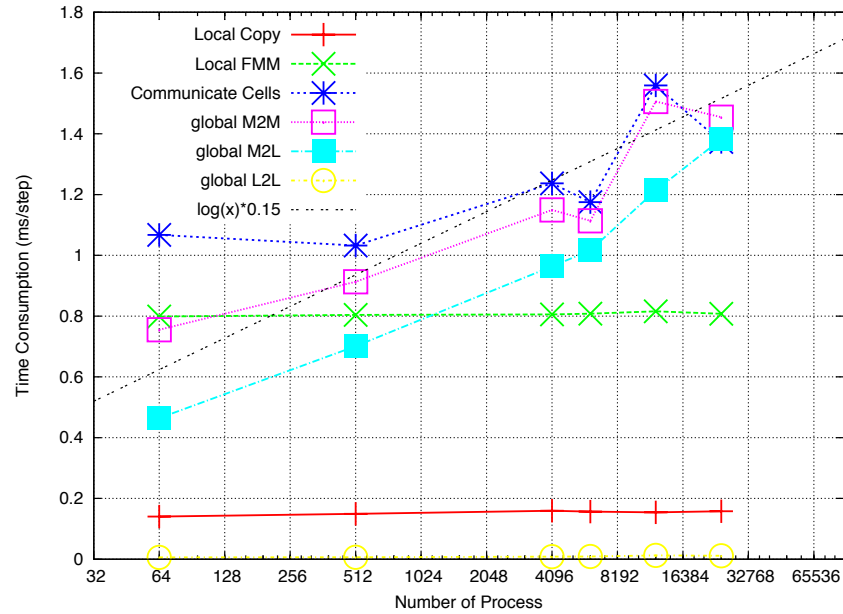


Figure 4.5: Weak scaling of FMM part of the MD simulation for 6,542 atoms/node. “Local FMM” is the aggregate time of all FMM kernels in the local tree, “Communicate Cells” is the time of the M2L communication in the local trees, “global M2M” and “global M2L” include both the communication time and computation time of the M2M and M2L kernels in the global tree. The dashed line is a reference for confirming the $\log P$ behavior of global communications.

4.2.5 Accuracy of periodic FMM

The periodic fast multipole method approximates the infinite periodic sum by placing a finite number of periodic images around the original domain. There are three sources of error in the periodic FMM:

- The truncation error of the multipole/local expansions in the FMM
- The error from using a finite number of periodic images
- The difference between the Ewald sum and use of periodic images (dipole correction)

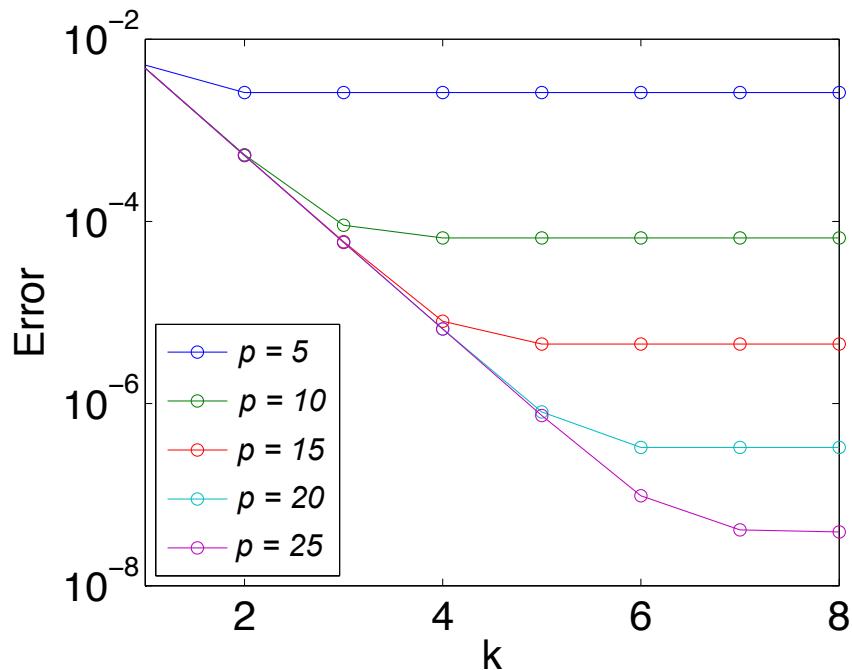


Figure 4.6: Error of periodic FMM with respect to the order of expansion P and number of periodic images $3^k \times 3^k \times 3^k$. The error is the relative L^2 norm of the difference between the force from the Ewald summation and periodic FMM. With this measure, 10^{-8} indicates that 8 significant digits are matching.

The first two are controllable and are a trade-off between the computational cost. The third source of error can be removed by adding a dipole correction term [89].

In Figure 4.6, we compare the results of a Ewald summation code, and periodic FMM code for different number of expansions p and different number of periodic images $3^k \times 3^k \times 3^k$. The number of atoms used in this test was $N = 1,000$ randomly distributed in a unit cube. When the order of expansion is too small, the FMM error dominates, and increasing the number of periodic images will not help. As p increases, it becomes possible to achieve higher accuracy with the use of sufficient number of periodic images. With $p = 15$ and $k = 4$, we are able to match 5 significant digits with the Ewald summation.

4.3 Macromolecular Crowding Simulations with All Atoms

4.3.1 Simulation methods

4.3.1.1 Simulated systems

To compare FMM and PME, we performed all-atom classical MD simulations of two different systems: an *in vitro* system and an *in vivo* system. The *in vitro* system consists of target proteins in a solvent, and the *in vivo* system consists of target proteins in a macromolecular crowding environment. The target protein is a putative heavy-metal binding protein TTHA, whose structure was determined by *in vitro* and in-cell NMR [91]. The initial structure of TTHA was taken from PDB entry 2ROE [91], which was solved by the NMR experiment *in vitro*. Ovalbumin, the major protein in egg whites, was chosen as the crowding agent. The structure of ovalbumin was taken from the PDB entry 1OVA [92]. The *in vitro* systems contained four TTHA molecules, 372 sodium ions, 364 chloride ions, and 137,977 water molecules in a 160 Å-side cube (vit). The *in vivo* system contained 8 TTHA molecules, 64 ovalbumin molecules, 1,200 potassium ions, 480 chloride ions, and 316,952 water molecules in a 240 Å-side cube (viv). The *in vivo* system mimics the conditions of a macromolecular crowding environment in living cells, where the macromolecules comprise $\sim 30\%$ of their molecular weight. These systems were equilibrated before the MD simulations, under temperature control ($T = 300$ K) and pressure control ($P = 1$ bar), using PMEMD in the AMBER program [93].

4.3.1.2 MD Simulations

The simulations of viv and vit were performed with periodic boundary conditions using the simulation software described earlier. We adopted the AMBER99SB force field [94] and used the TIP3P rigid water model [95] as the solvent molecule. The

integration time step was 2.0 fs. The bond lengths involving hydrogen atoms were constrained to equilibrium lengths using the RATTLE method. The Coulomb interactions were assessed by applying the PME and FMM methods. PME calculation [75] with a real cutoff of 12 Å and beta-spline interpolation order of 4 was used and FMM calculation with a real cutoff of 12 Å, multipole expansion order of 6, and 27 periodic images in each direction. A smooth cutoff scheme for nonbonded interactions was used for van der Waals interactions with a cutoff distance of 12 Å. The volume was kept constant in each system, and the temperature was maintained at 300 K using the Nosé-Hoover method [96, 97, 98]. The simulation time for analysis was 1 ns for each system. It is short for study of crowding but enough to detect difference of FMM and PME. Conformations of the TTHA molecules were recorded at every 1 ps.

4.3.2 Simulations of macromolecular crowding

All-atom classical MD simulations of target proteins in solution (vit) and target proteins in a crowded environment (viv) were performed to measure the difference of FMM and PME. The root mean square deviations (RMSDs) to the initial structure taken from the in vitro experiment for vit (fmm), vit (pme), viv (fmm) and viv (pme) were 1.001 ± 0.128 Å, 1.036 ± 0.142 Å, 1.310 ± 0.133 Å, and 1.211 ± 0.160 Å, respectively.

Figure 4.7 shows the root-mean square fluctuation (RMSF) curves of vit (fmm and pme), and viv (fmm and pme). These curves indicated that the difference of the fluctuations between FMM and PME was smaller than the difference of the fluctuations between vit and viv.

4.4 Conclusions

We have performed all-atom MD simulations for molecular crowding on the K computer. The performance benchmarks have shown excellent scalability of our classical

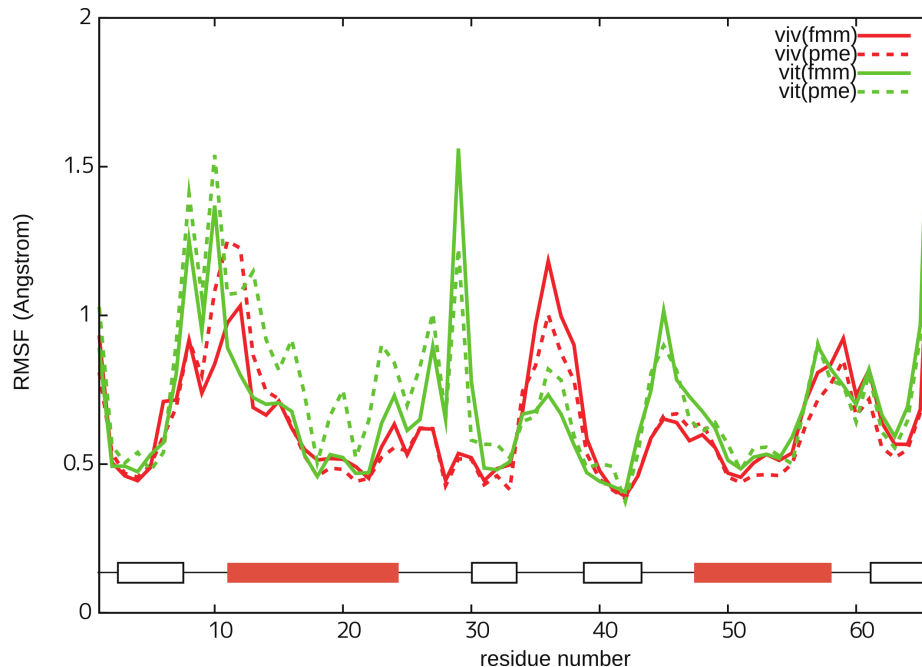


Figure 4.7: Conformational fluctuations of TTHA for *in vivo* (viv) and *in vitro* systems (vit). The red and green lines indicate the RMSFs of TTHA in viv (FMM: solid, PME: dashed) and vit (FMM: solid, PME: dashed), respectively. The abscissa axis is the residue number of TTHA and the ordinate axis is the RMSF value (in Angstrom). Red and white boxes indicate alpha-helices and beta-sheets, respectively.

MD code on up to 79,872 processors, and 638,976 cores. The sustained performance of 4.387 PFLOPs has been achieved using the K computer system with a nominal peak performance of 10.2 PFLOPs for the simulation with 523 million atoms. Good weak scalability were achieved for the simulations using the cutoff technique and FMM. We simulated the protein TTHA in ovalbumins as crowders using millions of atoms. The results obtained from the MD simulation using FMM were in good agreement with those obtained from PME, while the communication was reduced from $\mathcal{O}(P)$ in the PME to $\mathcal{O}(\log P)$ in the FMM for P processes.

Chapter 5

FMM-Based Preconditioners for Sparse Iterative Solvers

Among optimal hierarchical algorithms for the computational solution of elliptic problems, the FMM stands out for its adaptability to emerging architectures, having high arithmetic intensity, tunable accuracy, and relaxable global synchronization requirements. We demonstrate that, beyond its traditional use as a solver in problems for which explicit free-space kernel representations are available, the FMM has applicability as a preconditioner in finite domain elliptic boundary value problems, by equipping it with boundary integral capability for satisfying conditions at finite boundaries and by wrapping it in a Krylov method for extensibility to more general operators. Here, we do not discuss the well developed applications of FMM to implement matrix-vector multiplications within Krylov solvers of boundary element methods. Instead, we propose using FMM for the volume-to-volume contribution of inhomogeneous elliptic problems, where the boundary integral is a small part of the overall computation. Our method may be used to precondition sparse matrices arising from finite difference/element discretizations, and can handle a broader range of scientific applications. Like multigrid, it is capable of comparable algebraic convergence rates down to the truncation error of the discretized PDE, and it offers potentially superior multicore and distributed memory scalability properties on commodity architecture

This chapter includes results from the paper, “Fast Multipole Preconditioners for Sparse Matrices Arising from Elliptic Equations” by H. Ibeid, R. Yokota, J. Pestana, and D. Keyes [99] and the paper, “A Matrix-free Preconditioner for the Helmholtz Equation based on the Fast Multipole Method” by H. Ibeid, R. Yokota, and D. Keyes [100].

supercomputers. Compared with other methods exploiting the low-rank character of off-diagonal blocks of the dense resolvent operator, FMM-preconditioned Krylov iteration may reduce the amount of communication because it is matrix-free and exploits the tree structure of FMM.

In this chapter, we consider the Laplace, Stokes, and Helmholtz equations and devise highly scalable preconditioners for these problems. Our Poisson and Helmholtz preconditioners are based on a boundary element method in which matrix-vector multiplies are performed using FMM; the results are scalable $\mathcal{O}(N)$ and $\mathcal{O}(N \log N)$ preconditioners for the Laplace and Helmholtz equations, respectively. For the Stokes problem, we apply a block diagonal preconditioner, in which our Poisson preconditioner is combined with a simple diagonal matrix. FMM-based preconditioners were first proposed by Sambavaram *et al.* [101]. Such methods lacked practical motivation when FLOPs were expensive, since they turn a sparse matrix into a dense matrix of the same size before hierarchically grouping the off-diagonal blocks. But in a world of cheap FLOPs, the notion of a “compute-bound preconditioner” sounds more attractive.

The chapter is organized as follows. In Section 5.1 we present the model problems and in Section 5.2 we give an overview of Krylov subspace methods and preconditioning. The basis of our preconditioner is a boundary element method that is discussed briefly in Section 5.3. Our numerical results in Section 5.4 examine the convergence rates of FMM and multigrid for small Poisson, Stokes, and Helmholtz problems. Then, in Section 5.5 we scale up the test problems and perform strong scalability runs. Our conclusions are given in Section 5.6.

5.1 Model Problems

In this section we introduce the Poisson, Stokes, and Helmholtz model problems we wish to solve and describe properties of the linear systems that result from their

discretization. We focus on low-order finite elements but note that discretization by low-order finite difference or finite volume methods give linear systems with similar properties.

5.1.1 Poisson model problem

The model Poisson problems we wish to solve are of the form

$$-\nabla \cdot (a \nabla u) = f \text{ in } \Omega, \quad (5.1a)$$

$$u = g \text{ on } \Gamma, \quad (5.1b)$$

where $\Omega \in \mathbb{R}^d$, $d = 2, 3$ is a bounded connected domain with piecewise smooth boundary Γ , f is a forcing term, g defines the Dirichlet boundary condition, and $a \geq a_0 > 0$ is a sufficiently smooth function of space.

Discretization of (5.1) by finite elements or finite differences leads to a large, sparse linear system of the form

$$A\mathbf{x} = \mathbf{b}, \quad (5.2)$$

where $A \in \mathbb{R}^{N \times N}$ is the stiffness matrix and $\mathbf{b} \in \mathbb{R}^N$ contains the forcing and boundary data. The matrix A is symmetric positive definite and its eigenvalues depend on the mesh size, which we denote by h , as is typical of discretizations of elliptic PDEs. In particular, the condition number $\kappa = \lambda_{max}(A)/\lambda_{min}(A)$, the ratio of the largest and smallest eigenvalues of A , grows as $O(h^{-2})$ (see, for example, [2, Section 1.6]).

5.1.2 Stokes model problem

Incompressible Stokes problems are important when modeling viscous flows and for solving Navier-Stokes equations by operator splitting methods [102, Section 2.1]. The equations governing the velocity $\mathbf{u} \in \mathbb{R}^d$, $d = 2, 3$, and pressure $p \in \mathbb{R}$ of a Stokes fluid

in a bounded connected domain Ω with piecewise smooth boundary Γ are [102], [2]:

$$-\nabla^2 \mathbf{u} + \nabla p = 0 \quad \text{in } \Omega, \quad (5.3a)$$

$$\nabla \cdot \mathbf{u} = 0 \quad \text{in } \Omega, \quad (5.3b)$$

$$\mathbf{u} = \mathbf{w} \quad \text{on } \Gamma. \quad (5.3c)$$

Discretizing (5.3) by a stabilized finite element or finite difference approximation leads to the symmetric saddle point system

$$\underbrace{\begin{bmatrix} A & B^T \\ B & -C \end{bmatrix}}_{\mathcal{A}} \begin{bmatrix} \mathbf{u} \\ \mathbf{p} \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ \mathbf{g} \end{bmatrix}, \quad (5.4)$$

where $A \in \mathbb{R}^{N \times N}$ is the vector-Laplacian, a block diagonal matrix with blocks equal to the stiffness matrix from (5.2), $B \in \mathbb{R}^{M \times N}$ is the discrete divergence matrix, $C \in \mathbb{R}^{M \times M}$ is the symmetric positive definite pressure mass matrix and $\mathbf{f} \in \mathbb{R}^N$ and $\mathbf{g} \in \mathbb{R}^M$ contain the Dirichlet boundary data.

The matrix \mathcal{A} is symmetric indefinite and the presence of the stiffness matrix means that the condition number of \mathcal{A} increases as the mesh is refined. However, as we will see in the next section, the key ingredient in a preconditioner for \mathcal{A} that mitigates this mesh dependence is a good preconditioner for the Poisson problem. This allows us to use our preconditioner for the Poisson problem in this more complicated fluid dynamics problem as well.

5.1.3 Helmholtz model problem

The Helmholtz equation can be used to describe both wave propagations and scattering phenomena arising in many fields of science and technology. While most ap-

Although we treat only stabilized discretizations here, stable discretizations are no more difficult to precondition and are discussed in detail in Elman *et al.* [2, Chapter 6].

plications are concerned with waves propagation in exterior domains, it is common to utilize Helmholtz equations posed in interior domains with impedance boundary conditions to describe acoustic and elastic problems in finite domains. The Helmholtz equation takes the form

$$\nabla^2 u + k^2 u = f \text{ in } \Omega, \quad (5.5a)$$

$$\partial_n u - iku = g \text{ on } \Gamma, \quad (5.5b)$$

where Ω is a connected bounded domain in \mathbb{R}^d , $d = 2, 3$, with piecewise smooth boundary Γ , k represents a constant wave number, and f and g are prescribed complex functions.

Discretizing (5.5) by finite element or finite difference methods leads to a large sparse linear system of the form

$$A\mathbf{x} = \mathbf{b}, \quad (5.6)$$

where $A \in \mathbb{C}^{N \times N}$ is a large sparse symmetric matrix and $\mathbf{b} \in \mathbb{C}^N$ contains the forcing and boundary data. For large values of k , the matrix A is complex-valued and indefinite, i.e., A has eigenvalues with both positive and negative real parts.

Iterative methods, such as Krylov subspace solvers, are widely used in many areas of scientific computing for solving such large sparse linear systems where direct methods, although robust and reliable, have expensive computational requirements.

5.2 Iterative Solvers and Preconditioning

5.2.1 Krylov subspace methods

The main idea of Krylov subspace methods is to generate a basis of Krylov subspace

$$\mathcal{K}_j(A, r_0) = \text{span}\{r_0, Ar_0, A^2r_0, \dots, A^{j-1}r_0\}, \quad (5.7)$$

and then seek an approximate solution to the original problem from this subspace. Here, $r_0 = b - Ax_0$, x_0 is the initial approximate solution, and $\mathcal{K}_j(A, r_0)$ is the j th Krylov subspace associated with A and r_0 . A wide variety of iterative methods fall within the Krylov subspace framework.

The conjugate gradient method (CG) [18] is the optimal Krylov solver for solving sparse symmetric positive definite linear systems [103]. To solve symmetric indefinite linear systems, minimal residual method (MINRES) [19] can be used, as well as its generalization to the nonsymmetric case, GMRES [20]. Both algorithms have the minimization property but GMRES has the advantage that theoretically it guarantees convergence. The main problem in GMRES is that it uses long recurrences which implies that the amount of storage increases at each iteration. Therefore, applications of GMRES may be limited by available storage. To overcome this problem, restarted versions of the GMRES method are used, e.g., GMRES(m) [20]. In the restarted GMRES, computation and storage costs are limited by specifying a fixed number of vectors to be generated. However, since restarting removes the previous convergence history, GMRES(m) does not guarantee convergence. The biconjugate gradient stabilized (BiCGSTAB) [104] and conjugate gradient squared (CGS) [105] methods are short recurrence alternatives to GMRES. Even though BiCGSTAB is generally more stable and robust than CGS [106], neither method guarantees monotonically decreasing residuals.

In this work we focus on three Krylov methods: the CG for systems with symmetric positive definite coefficient matrices, and the MINRES and GMRES methods for systems with symmetric indefinite matrices. For implementation and convergence details, we refer the reader to the books by Greenbaum [107] and Saad [108].

5.2.2 Preconditioning

The convergence of Krylov subspace methods depends on the spectrum of the coefficient matrix which for the Poisson, Stokes, and Helmholtz problems, as well as other elliptic PDEs, deteriorates as the mesh is refined. This dependence can be removed by preconditioning. The general rule for preconditioners is that the preconditioned system should be easy to solve, i.e., converges rapidly, and cheap to apply [109]. It is important to strike a balance between these two requirements as they are competing with each other.

One can apply a preconditioner on the left of the linear system, the right, or a combination of both. By left preconditioning, we solve a linear system premultiplied by a preconditioning matrix M^{-1} , i.e.,

$$M^{-1}Ax = M^{-1}b. \quad (5.8)$$

On the other hand, right preconditioning is based on solving

$$AM^{-1}\hat{x} = b, \quad (5.9)$$

where $\hat{x} = Mx$. Both preconditionings show typically a similar convergence behavior and the type of preconditioning to use depends mainly on the choice of the iterative method and the problem characteristics. For example, right preconditioning is often used with the GMRES method [109]. The essential difference between left and right preconditioned GMRES is that the left-preconditioned method computes residuals based on the preconditioned system while the residuals for the right-preconditioned GMRES are identical to the true residuals. This difference may affect the stopping criterion [108].

When Krylov subspace methods are used, it is not necessary to form the pre-

conditioning matrix M^{-1} explicitly. Instead, the preconditioning matrix can be a linear operation that defines the inverse of a matrix implicitly. This enables us to use matrix-free approaches such as multigrid or the fast multipole method.

Many preconditioners for the Poisson problem reduce the number of iterations, with geometric and algebraic multigrid among the most effective strategies [2], [10]. However, to achieve a lower time-to-solution than can be obtained for the original system, it is also necessary to choose a preconditioner that can be cheaply applied at each iteration. Both geometric and algebraic multigrid methods are $\mathcal{O}(N)$, and therefore exhibit good performance on machines and problems for which computation is expensive. However, stresses arise in parallel applications as discussed in the introduction.

For Stokes problems we consider the block diagonal preconditioner

$$\mathcal{P} = \begin{bmatrix} P_A & 0 \\ 0 & P_S \end{bmatrix}, \quad (5.10)$$

where $P_A \in \mathbb{R}^{N \times N}$ and $P_S \in \mathbb{R}^{M \times M}$ are symmetric positive definite matrices. The advantage of this preconditioner is that there is no coupling between the blocks, so \mathcal{P} is scalable provided the blocks P_A and P_S are.

Appropriate choices for P_A and P_S have been well studied and it is known that mesh-independent convergence of MINRES can be recovered when P_A is spectrally equivalent to A in (5.4) and P_S is spectrally equivalent to the pressure mass matrix $Q \in \mathbb{R}^{M \times M}$ [110], [2, Chapter 6]. These spectral equivalence requirements imply that the eigenvalues of $P_A^{-1}A$ and $P_S^{-1}Q$ are bounded in an interval on the positive real line independently of the mesh width h .

It typically suffices to use the diagonal of Q [2, Chapter 6], [111] or a few steps of Chebyshev semi-iteration [112] for P_S . Moreover, the diagonal matrix is extremely parallelizable. Thus, the key to obtaining a good preconditioner for \mathcal{A} is to approx-

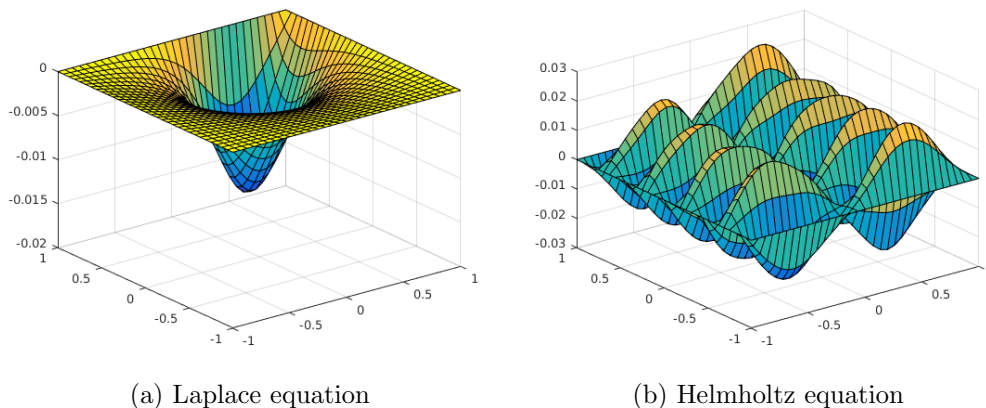


Figure 5.1: Solution of the Laplace and Helmholtz equations with the same boundary conditions.

imate the vector Laplacian effectively. This is typically the most computationally intensive part of the preconditioning process, since in most cases $M \ll N$.

For Helmholtz equations Krylov methods are not effective solvers without a good preconditioner [113]. Using the problem defined in (5.21), Figure 5.1 shows the fundamental influence of the wave number k on the solution of the Helmholtz equation by comparing it against the Laplace equation ($k = 0$). The solution of the Laplace equation is large only near the point source while for the Helmholtz equation, with $k = 15$, the solution takes on large values periodically throughout the domain. Figure 5.2 shows how this influences the convergence of the unpreconditioned GMRES method. While the residual decreases rapidly for the Laplace equation, convergence stagnates for the Helmholtz problem. It is therefore important to have a preconditioner as GMRES method alone is not competitive for Helmholtz equations.

Multigrid methods, while enormously effective when applied to coercive equations, have severe convergence problems when applied to the indefinite Helmholtz equation [10]. The reason for this is while the characteristic components of the Helmholtz problem can be accurately approximated by the discrete equations on the fine grids, these components are invisible to any local relaxation since their errors can have

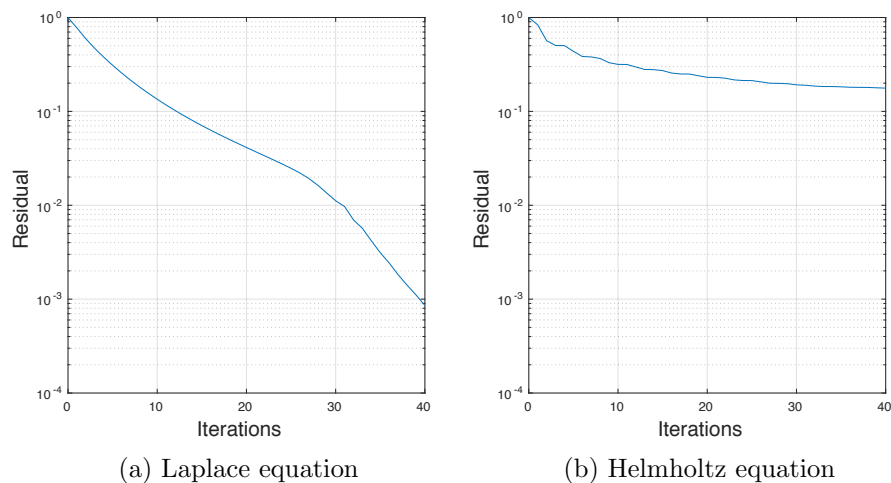


Figure 5.2: Evolution of the residual of the unpreconditioned GMRES method for the Laplace equation, $k = 0$, and the Helmholtz equation, $k = 15$.

very small residuals. On the other hand, the characteristic components can not be approximated on coarser grids since these grids do not resolve their oscillations [114].

5.2.3 The FMM-BEM preconditioner

In this chapter we propose an alternative preconditioner for Poisson, Stokes, and Helmholtz problems that heavily utilizes the fast multipole method. The FMM is $\mathcal{O}(N)$ with compute intensive inner kernels. It has a hierarchical data structure that allows asynchronous communication and execution. These features make the FMM a promising preconditioner for large scale problems on future computer architectures. We show that this preconditioner improves the convergence of Krylov subspace methods, and is effectively parallelized on today's highly distributed architectures.

The FMM in its original form relies on free-space Green's functions and is able to solve problems with free-field boundary conditions. In Section 5.3 the FMM preconditioner is extended to Dirichlet, Neumann or Robin boundary conditions for arbitrary geometries by coupling it with a boundary element method (BEM). Our approach uses the FMM as a *preconditioner* inside a *sparse* matrix solver and the

BEM solve is inside the preconditioner. Numerous previous studies use FMM for the matrix-vector multiplication inside the Krylov solver for the dense matrix arising from the boundary element discretization. In the present method we are calculating problems with non-zero sources in the volume, and the FMM is used to calculate the volume-to-volume contribution. This means we are performing the action of an $N \times N$ dense matrix-vector multiplication, where N is the number of points in the volume (not the boundary). Additionally, as discussed in Section 5.3.4, it is possible to extend the boundary element method to problems with variable diffusion coefficients, particularly since low accuracy solves are often sufficient in preconditioning.

Figure 5.3 shows the flow of calculation of our FMM-BEM preconditioner within the conjugate gradient method; its role in other Krylov solvers is similar. The FMM is used to approximate the matrix-vector multiplication of A^{-1} within the preconditioner. The BEM solver adapts the FMM to finitely applied boundary conditions. During each step of the iteration, the u vector from the previous iteration is used to determine $\partial u/\partial n$ at the boundary from (5.13), then (5.14) is used to compute the new u in the domain Ω .

5.3 Boundary Element Method

5.3.1 Formulation

We use a standard Galerkin boundary element method [115] with volume contributions to solve the Poisson and Helmholtz equations. A brief description of the formulation is given here, more details are provided in Appendix C. Applying Green's third identity to (5.1a) with $a \equiv 1$ or to (5.5a) gives

$$\int_{\Gamma} \frac{\partial u}{\partial n} G d\Gamma - \int_{\Gamma} u \frac{\partial G}{\partial n} d\Gamma + \int_{\Omega} u (\nabla^2 G) d\Omega = \int_{\Omega} f G d\Omega, \quad (5.11)$$

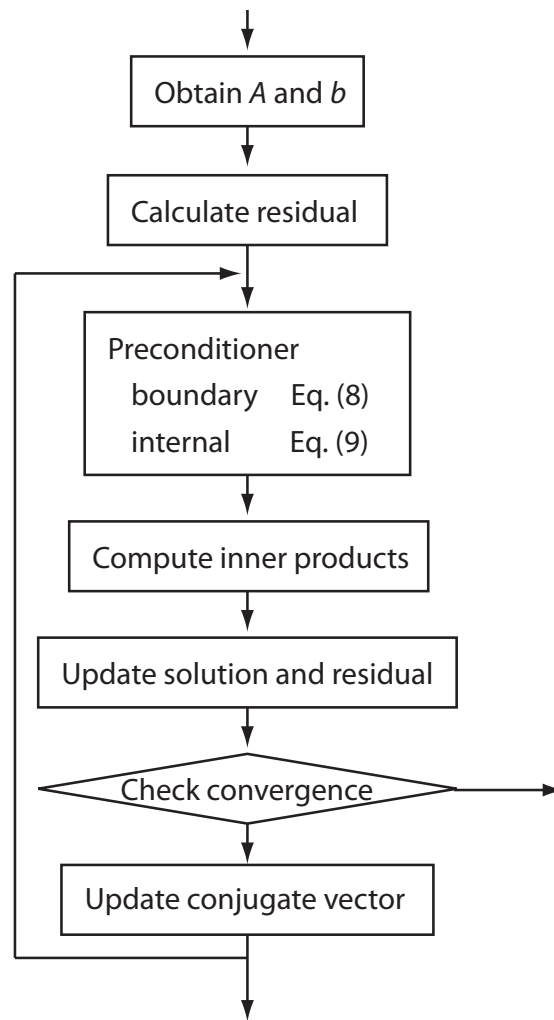


Figure 5.3: Flow chart of the FMM-BEM preconditioner within the conjugate gradient method.

where G is the Green's function of the Laplace or Helmholtz operators, $\frac{\partial}{\partial n}$ is the derivative in the outward normal direction, and Γ is the boundary. Following the definition of the Green's function $\nabla^2 G = \delta$, the third term in (5.11) becomes

$$\int_{\Omega} u(\nabla^2 G)d\Omega = \int_{\Omega} u\delta d\Omega = \begin{cases} \frac{1}{2}u & \text{on } \partial\Omega, \\ u & \text{in } \Omega. \end{cases} \quad (5.12)$$

Therefore, we may solve the constant coefficient inhomogeneous Poisson and Helmholtz problems by solving the following set of equations

$$\int_{\Gamma} \frac{\partial u}{\partial n} G d\Gamma = \int_{\Gamma} u \left(\frac{\partial G}{\partial n} - \frac{1}{2}\delta \right) d\Gamma + \int_{\Omega} f G d\Omega \quad \text{on } \partial\Omega, \quad (5.13)$$

$$u = \int_{\Gamma} u \frac{\partial G}{\partial n} d\Gamma - \int_{\Gamma} \frac{\partial u}{\partial n} G d\Gamma + \int_{\Omega} f G d\Omega \quad \text{in } \Omega. \quad (5.14)$$

As an example, consider the case where Dirichlet boundary conditions are prescribed on Γ . The unknowns are $\partial u/\partial n$ on Γ and u in $\Omega \setminus \Gamma$, where (5.13) solves for the former and (5.14) can be used to determine the latter. For Neumann boundary conditions one can simply switch the two boundary integral terms in (5.13) and solve for u instead of $\partial u/\partial n$. In either case, we obtain both u and $\partial u/\partial n$ at each point on the boundary, then calculate (5.14) to obtain u at the internal points. The last term in (5.14) takes up most of the calculation time since it is a volume integral for every point in the volume, whereas other terms are either for every point on the boundary or are boundary integrals.

5.3.2 Singular integrals

The Laplace Green's function in 2-D

$$G = -\frac{1}{2\pi} \log r \quad (5.15)$$

is singular. Therefore, the integrals involving G or $\partial G/\partial n$ in (5.13) and (5.14) are singular integrals. As described in the following section, these singular integrals are discretized into piecewise integrals, which are evaluated using Gauss-Legendre quadratures with special treatment for the singular piecewise integral. For boundary integrals in (5.13) and (5.14), analytical expressions exist for the piecewise integral. However, for the volume integral an analytical expression does not exist [116]. For this reason, we use a smoothed Green's function of the form

$$G = -\frac{1}{2\pi} \log(\sqrt{r^2 + \epsilon^2}), \quad (5.16)$$

where ϵ is a small number that changes with the grid resolution.

5.3.3 Discretization

The integrals in equations (5.13) and (5.14) are discretized in a similar fashion to finite element methods. In the following description of the discretization process, we will use the term on the left hand side in (5.13) as an example. The first step is to break the global integral into a discrete sum of piecewise local integrals over each element

$$\int_{\Gamma} \frac{\partial u}{\partial n} G d\Gamma \approx \sum_{j=1}^{N_{\Gamma}} \int_{\Gamma_j} \frac{\partial u_j}{\partial n} G d\Gamma_j, \quad (5.17)$$

where N_{Γ} is the number of boundary nodes. These piecewise integrals are performed by using quadratures over the basis functions [115]. In the present case, we use

constant elements so there are no nodal points at the corners of the domain for the tests in Sections 5.4 and 5.5. By applying this discretization technique to all terms in (5.13) we obtain

$$N_\Gamma \left\{ \begin{array}{c} \overbrace{\begin{bmatrix} \ddots & & \\ & G_{ij} & \\ & & \ddots \end{bmatrix}}^{N_\Gamma} \underbrace{\begin{bmatrix} \vdots \\ \frac{\partial u_j}{\partial n} \\ \vdots \end{bmatrix}}_{\text{unknown}} = \overbrace{\begin{bmatrix} \ddots & & \\ & \frac{\partial G_{ij}}{\partial n} - \frac{1}{2}\delta_{ij} & \\ & & \ddots \end{bmatrix}}^{N_\Gamma} \begin{bmatrix} \vdots \\ u_j \\ \vdots \end{bmatrix} + \overbrace{\begin{bmatrix} \ddots & & \\ & G_{ij} & \\ & & \ddots \end{bmatrix}}^{N_\Omega} \begin{bmatrix} \vdots \\ f_j \\ \vdots \end{bmatrix}, \end{array} \right.$$

where N_Ω is the number of internal nodes. All values on the right hand side are known, and $\partial u/\partial n$ at the boundary is determined by solving the linear system. Similarly, we apply the discretization to (5.14) to have

$$N_\Omega \left\{ \begin{array}{c} \begin{bmatrix} \vdots \\ u_i \\ \vdots \end{bmatrix} = \overbrace{\begin{bmatrix} \ddots & & \\ & \frac{\partial G_{ij}}{\partial n} & \\ & & \ddots \end{bmatrix}}^{N_\Gamma} \begin{bmatrix} \vdots \\ u_j \\ \vdots \end{bmatrix} - \overbrace{\begin{bmatrix} \ddots & & \\ & G_{ij} & \\ & & \ddots \end{bmatrix}}^{N_\Gamma} \begin{bmatrix} \vdots \\ \frac{\partial u_j}{\partial n} \\ \vdots \end{bmatrix} + \overbrace{\begin{bmatrix} \ddots & & \\ & G_{ij} & \\ & & \ddots \end{bmatrix}}^{N_\Omega} \begin{bmatrix} \vdots \\ f_j \\ \vdots \end{bmatrix}. \end{array} \right.$$

At this point, all values on the right hand side are known so one can perform three matrix-vector multiplications to obtain u at the internal nodes, and the solution to the original equation (5.1a) or (5.5a). The third term on the right hand side involves an $N_\Omega \times N_\Omega$ matrix, and is the dominant part of the computational load. This matrix-vector multiplication can be approximated in $\mathcal{O}(N)$ time by using the FMM described in Chapter 2. We also use the FMM to accelerate all other matrix-vector multiplications.

5.3.4 Variable coefficient problems

A natural question that arises is how to extend the boundary element method, which is the basis of our preconditioner, to problems with variable diffusion coefficients.

Several strategies for extending boundary element methods to problems with variable diffusion coefficients have been proposed (see, for example, the thesis of Brunton [117, Chapter 3]). Additionally, in this preconditioner setting we may not need to capture the variation in the diffusion coefficient to a high degree of accuracy; for a similar discussion in the context of additive Schwarz preconditioners see, for example, Graham *et al.* [118].

Although analytic fundamental solutions can sometimes be found for problems with variable diffusion (see, e.g., Cheng [119] and Clements [120]), in most cases numerical techniques are employed. One popular method is to introduce a number of subdomains, on each of which the diffusion coefficient is approximated by a constant function [121, 122].

A second option is to split the differential operator into a part for which a fundamental solution exists and another which becomes part of the source term. Specifically, starting from (5.1), a similar approach to that described in Banerjee [123] and Cheng [119] leads to

$$\int_{\Gamma} au \frac{\partial G}{\partial n} d\Gamma - \int_{\Gamma} a \frac{\partial u}{\partial n} G - \int_{\Omega} u \nabla a \cdot \nabla G d\Omega - \int_{\Omega} au \nabla^2 G d\Omega = \int_{\Omega} f G d\Omega,$$

where again G is the standard fundamental solution for the Laplace operator, i.e., not the fundamental solution for (5.1). We can then proceed as described above for (5.11). It is also possible (see Concus and Golub [124]) to change the dependent variable to soak up the variation in a prior to discretization, again resulting in a modified source FEM.

5.4 Numerical Results

In this section we demonstrate the potential of the FMM-based preconditioner by applying it to a number of test problems and comparing it with standard preconditioners. The primary aim is to assess the effectiveness of the preconditioner at reducing the number of Krylov subspace iterations that are required for convergence to a given tolerance. Additionally, we seek to ascertain whether mesh independence is achieved. We defer reporting on performance to Section 5.5. Accordingly, we choose problems that are small enough to enable solution by MATLAB.

Our Poisson and Helmholtz problems are all two dimensional and include examples with homogeneous and inhomogeneous Dirichlet boundary conditions. We additionally present a 2-D Stokes flow problem and show that, as predicted, combining the FMM-based Poisson preconditioner with a block diagonal matrix gives an effective preconditioner for the saddle point problem (5.4).

Throughout, our stopping criterion is a decrease in the relative residual of six orders of magnitude. If such a decrease is not achieved after *maxit* iterations the computations are terminated; this is denoted by ‘—’ in the tables. This maximum number of iterations is stated for each problem below. For all problems and preconditioners the initial iterate is the zero vector. Our MATLAB implementation of the FMM used in this section is a direct N -body summation that is subsequently degraded to simulate a truncated FMM.

5.4.1 The Poisson equation

We first test our preconditioner on three 2-D Poisson problems with a constant diffusion coefficient on the domain on $[-1, 1]^2$. Though these problems differ only in their inhomogeneities, they verify different segments of code for the PDE and boundary element discretization. We discretize the problems by Q_1 finite elements using

h	GMG	AMG	FMM	IC
2^{-4}	6	5	5	10
2^{-5}	6	6	6	18
2^{-6}	7	6	6	—
2^{-7}	7	6	6	—
2^{-8}	7	6	7	—

Table 5.1: Preconditioned CG iterations for the relative residual to reduce by six orders of magnitude for the problem with $-\nabla^2 u = 1$ and homogeneous boundary conditions.

h	$\lambda_{min}(A)$	$\lambda_{max}(A)$	$\kappa(A)$	$\lambda_{min}(P^{-1}A)$	$\lambda_{max}(P^{-1}A)$	$\kappa(P^{-1}A)$
2^{-4}	0.076	3.94	52	0.73	1.29	1.77
2^{-5}	0.019	3.99	207	0.72	1.29	1.79
2^{-6}	0.005	4.00	830	0.72	1.30	1.80

Table 5.2: Smallest (λ_{min}) and largest (λ_{max}) eigenvalues and condition number (κ) of the stiffness matrix A and FMM-preconditioned matrix $P^{-1}A$ for the problem with $-\nabla^2 u = 1$ and homogeneous boundary conditions.

IFISS [125], [126], with default settings. Our fast multipole preconditioner is compared with the incomplete Cholesky (IC) factorization [127] with zero fill implemented in MATLAB and the algebraic multigrid (AMG) and geometric multigrid (GMG) methods in IFISS. Within the GMG preconditioner we select point-damped Jacobi as a smoother instead of the default ILU, which is less amenable to parallelization. Otherwise, default settings for both multigrid methods are used. For all preconditioners, $maxit = 20$ and we apply preconditioned conjugate gradients.

Our first example is the first reference problem in Elman *et al.* [2, Section 1.1] for which

$$\nabla^2 u = 1 \text{ in } \Omega = [-1, 1]^2, \quad u = 0 \text{ on } \Gamma.$$

Table 5.1 lists the preconditioned CG iterations for each preconditioner applied. The FMM preconditioner, as well as GMG and AMG appear to give mesh-independent convergence, although the incomplete Cholesky factorization does not.

In Table 5.2 we plot the eigenvalues of the FMM preconditioned stiffness matrix

for $h = 2^{-4}, 2^{-5}$ and 2^{-6} . It is clear that the smallest eigenvalue of A decreases as the mesh is refined leading to an increase in the condition number; this is particularly problematic for Krylov subspace methods, such as CG iteration, whose iteration count can grow as the square root of the condition number. However, the eigenvalues of the FMM-preconditioned matrix are bounded away from the origin in a small interval that does not increase in size as the mesh is refined. This hints at spectral equivalence between the FMM-based preconditioner and the stiffness matrix which is unsurprising given that FMM is derived from the exact inverse of the continuous problem. The condition number appears to be bounded, which is a consequence of the mesh-independent convergence observed.

Our second example is the third reference problem from Elman *et al.* [2, Section 1.1] posed on $[-1, 1]^2$ which is characterized by inhomogeneous Dirichlet boundary conditions and the analytic solution

$$u(x, y) = \frac{2(1 + y)}{(3 + x)^2 + (1 + y)^2}.$$

From Table 5.3 we find that, similarly to the previous problem, the FMM preconditioner and both multigrid preconditioners are mesh independent but the Cholesky preconditioner is not. The FMM preconditioner is also competitive with the multigrid methods. Thus, on systems on which applying the FMM preconditioner is significantly faster than applying the multigrid preconditioners, we will achieve a faster time-to-solution with the former. We note that the eigenvalues and condition numbers obtained for the FMM preconditioned stiffness matrix are the same as those computed for the previous example.

The final problem we consider in this section is the Poisson problem with solution

$$u(x, y) = x^2 + y^2$$

h	GMG	AMG	FMM	IC
2^{-4}	5	5	5	11
2^{-5}	5	5	5	19
2^{-6}	5	5	5	—
2^{-7}	5	5	5	—
2^{-8}	5	5	5	—

Table 5.3: Preconditioned CG iterations for the relative residual to reduce by six orders of magnitude for the problem with $-\nabla^2 u = 0$ and inhomogeneous boundary conditions.

h	GMG	AMG	FMM	IC
2^{-4}	5	5	5	10
2^{-5}	5	5	5	18
2^{-6}	5	5	5	—
2^{-7}	5	5	5	—
2^{-8}	5	5	5	—

Table 5.4: Preconditioned CG iterations for the relative residual to reduce by six orders of magnitude for the problem with $-\nabla^2 u = -4$ and inhomogeneous boundary conditions.

on $[-1, 1]^2$, which has forcing term $f \equiv -4$ in the domain and inhomogeneous Dirichlet boundary conditions. The convergence results for this problem, given in Table 5.4, are similar to those for the previous problems. They show that the FMM preconditioner gives mesh independent convergence and is competitive with AMG and GMG. We also obtain the same eigenvalue results as for the previous examples.

5.4.2 Variable-coefficient Poisson equation

For sufficiently smooth diffusion coefficient variation, we can precondition the variable-coefficient problem with the constant-coefficient problem, since they are spectrally equivalent. We test this approach on the variable-coefficient Poisson equation of the

μ	AMG	FMM
2^{-16}	6	6
2^{-8}	6	6
2^{-4}	6	7
2^{-2}	6	8

Table 5.5: Preconditioned CG iterations for the relative residual to reduce by six orders of magnitude ($h = 2^{-6}$, $m = 1$ and $n = 1$).

m	n	AMG	FMM
1	1	6	7
2	2	6	7
4	4	6	7
8	8	6	7
16	16	6	7

Table 5.6: Preconditioned CG iterations for the relative residual to reduce by six orders of magnitude ($h = 2^{-6}$, $\mu = 2^{-4}$).

form

$$\begin{aligned}
 -\nabla \cdot (a \nabla u) &= 1 \text{ in } \Omega, \\
 u &= 0 \text{ on } \Gamma,
 \end{aligned}$$

where

$$a = 1 + \mu(\sin(m\pi x) \sin(n\pi y)).$$

Tables 5.5, 5.6, and 5.7 show that the FMM preconditioner and both multigrid preconditioners achieved mesh independent convergence for different amplitudes μ and frequencies m and n . Also, the FMM preconditioner is competitive with the algebraic multigrid method requiring comparable number of iterations.

Similar to Table 5.2, Table 5.8 shows that the eigenvalues of the FMM-preconditioned matrix are bounded away from the origin.

n	AMG	FMM
1	6	7
2	6	7
4	6	7
8	6	7
16	6	7

Table 5.7: Preconditioned CG iterations for the relative residual to reduce by six orders of magnitude ($m = 4$, $h = 2^{-6}$, $\mu = 2^{-4}$).

h	m	n	$\lambda_{min}(A)$	$\lambda_{max}(A)$	$k(A)$	$\lambda_{min}(P^{-1}A)$	$\lambda_{max}(P^{-1}A)$	$k(P^{-1}A)$
2^{-4}	3	3	0.0759	3.9923	53	0.4423	1.0000	2.26
2^{-5}	6	6	0.0192	4.0199	209	0.4371	1.0040	2.29
2^{-6}	12	12	0.0048	4.0280	839	0.4360	1.0061	2.31

Table 5.8: Smallest (λ_{min}) and largest (λ_{max}) eigenvalues and condition number (k) of the stiffness matrix A and FMM-preconditioned matrix $P^{-1}A$ with $\mu = 2^{-4}$.

5.4.3 Stokes problem

Next, we examine convergence for a 2-D Stokes flow. The leaky cavity problem [2, Example 5.1.3] on $[-1, 1]$ is discretized by $Q_1 - P_0$ elements in MATLAB using IFISS with default settings. As described in Section 5.2, by combining a stiffness matrix preconditioner P_A with the diagonal of the pressure mass matrix P_S , an effective preconditioner (5.10) for the saddle point system (5.4) is obtained. Here, we are interested in using the FMM preconditioner for P_A , and we compare its performance with AMG and GMG in Table 5.9. We do not consider the incomplete Cholesky factorization of A because of its poor performance on the stiffness matrix (see Tables 5.1, 5.3 and 5.4). We set $maxit = 50$ and apply preconditioned MINRES to the saddle point system.

As for the Poisson problem, the FMM-based preconditioner provides a mesh-independent preconditioner that is comparable to algebraic and geometric multigrid. Although two or three more iterations are required by the FMM preconditioner than the AMG preconditioner, if each iteration is faster the time-to-solution may be lower.

h	GMG	AMG	FMM
2^{-4}	32	31	35
2^{-5}	32	32	35
2^{-6}	33	32	33
2^{-7}	31	31	33
2^{-8}	31	31	31

Table 5.9: Preconditioned MINRES iterations for the relative residual to reduce by six orders of magnitude for the Stokes problem.

5.4.4 The Helmholtz equation

Here, several 2-D numerical experiments are performed to assess the convergence of the FMM preconditioner for the Helmholtz equation for an increasing mesh size (h^{-1}) and/or increasing wavenumber (k). The domain Ω is discretized by Q_1 finite elements in MATLAB using IFISS where we construct the coefficient matrix by adding the stiffness matrix to k^2 times the mass matrix. The discretization results in a large sparse symmetric linear system which we solve using the GMRES iterative solver with $maxit = 20$.

Our first Helmholtz problem [128] is posed on a unit square $[0, 1]^2$ of homogeneous medium with homogeneous Dirichlet boundary conditions as follows

$$\nabla^2 u + k^2 u = (k^2 - 5\pi^2) \sin(\pi x) \sin(2\pi y) \text{ in } \Omega, \quad (5.19a)$$

$$u = 0 \quad \text{on } \Gamma. \quad (5.19b)$$

The exact solution of (5.19) is

$$u = \sin(\pi x) \sin(2\pi y). \quad (5.20)$$

The sub-figures in Figure 5.4 show the eigenvalues of the coefficient matrix of the original linear system for $k = 5, 10, 20$, and 40 , respectively. Notice that the

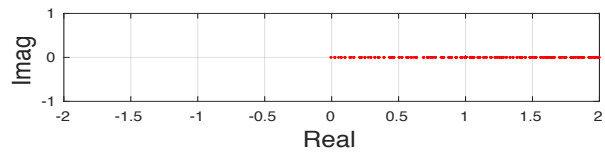
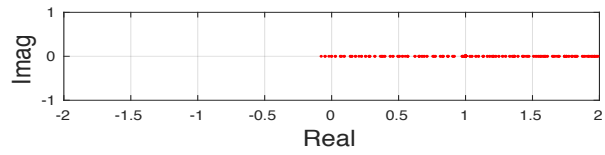
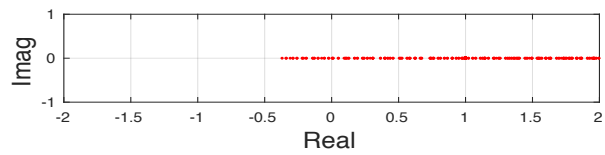
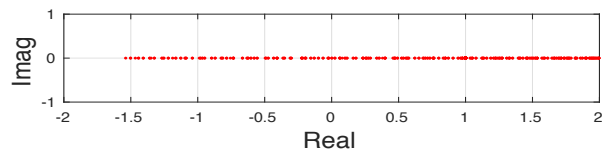
(a) $k = 5$ (b) $k = 10$ (c) $k = 20$ (d) $k = 40$

Figure 5.4: Eigenvalues of the the coefficient matrix A for the problem described in (5.19) with $k = 5, 10, 20,$ and $40,$ respectively, $h = 2^{-5}$.

h	k	GMG	AMG	FMM	IC
2^{-4}	5	11	5	4	13
2^{-5}	10	—	6	4	—
2^{-6}	20	—	—	4	—
2^{-7}	40	—	—	5	—

Table 5.10: Preconditioned GMRES iterations for the relative residual to reduce by six orders of magnitude, $kh = 0.3125$.

h	GMG	AMG	FMM	IC
2^{-5}	—	15	5	—
2^{-6}	—	15	4	—
2^{-7}	—	15	4	—

Table 5.11: Preconditioned GMRES iterations for the relative residual to reduce by six orders of magnitude, $k = 5$.

coefficient matrix becomes more indefinite as the wavenumber increases with $k = 5$ resembles the slightly indefinite problem. Different mesh refinements are used to solve (5.19) with various wave numbers k . Table 5.10 lists the number of GMRES iterations required to reach the pre-specified tolerance for the GMG, AMG, FMM, and IC preconditioners. For small wave numbers, all preconditioners show a satisfactory performance. GMG and IC become less effective for increasing values of k where the number of iterations required for convergence increases rapidly. For larger k , preconditioning with FMM shows the best performance where other preconditioners fail to converge within *maxit*.

The second Helmholtz example is posed on the square domain $[-1, 1]^2$ and is characterized by homogeneous Dirichlet boundary conditions as follows

$$\nabla^2 u + k^2 u = e^{-10((y-1)^2 + (x-0.5)^2)} \text{ in } \Omega, \quad (5.21a)$$

$$u = 0 \quad \text{on } \Gamma. \quad (5.21b)$$

k	GMG	AMG	FMM	IC
0.8	5	5	4	—
2	8	6	4	—
5	—	15	4	—

Table 5.12: Preconditioned GMRES iterations for the relative residual to reduce by six orders of magnitude, $h = 2^{-6}$.

Table 5.11 gives the number of GMRES iterations required for convergence on various mesh sizes with $k = 5$ for the GMG, AMG, FMM and IC preconditioners. Both FMM and AMG preconditioners appear to give mesh independent convergence, whereas GMG and IC factorization fail to converge within *maxit*. Table 5.12 lists the number of preconditioned GMRES iterations for each preconditioner applied as a function of the wave number k with $h = 2^{-6}$. For small wave numbers, the GMG, AMG, and FMM preconditioners show a very comparable performance. As k increases, both multigrid methods start to diverge while the FMM preconditioner maintains a wavenumber-independent convergence for the given range of k .

The third Helmholtz example [129] is posed on $[0, 1]^2$ and is characterized by inhomogeneous Dirichlet boundary conditions as follows

$$\nabla^2 u + k^2 u = 2 \sin(\mu x) \cos(\mu y) + 4\mu x \cos(\mu x) \cos(\mu y) \text{ in } \Omega, \quad (5.22a)$$

$$u = x^2 \sin(\mu x) \cos(\mu y) \text{ on } \Gamma, \quad (5.22b)$$

where $k = \mu\sqrt{2}$.

The right-hand term f and exact solution u in a unit square domain with various parameter μ are shown in Figure 5.5. Notice that the larger the value of μ , the greater the variation of the function. The subtables in Table 5.13 list the number of preconditioned GMRES iterations for the GMG, AMG, FMM, and IC preconditioners for $\mu = 1, 4, 6,$ and $8,$ respectively. For low frequencies, the GMG, AMG, and FMM preconditioners show a very satisfactorily comparable performance. The GMG preconditioner

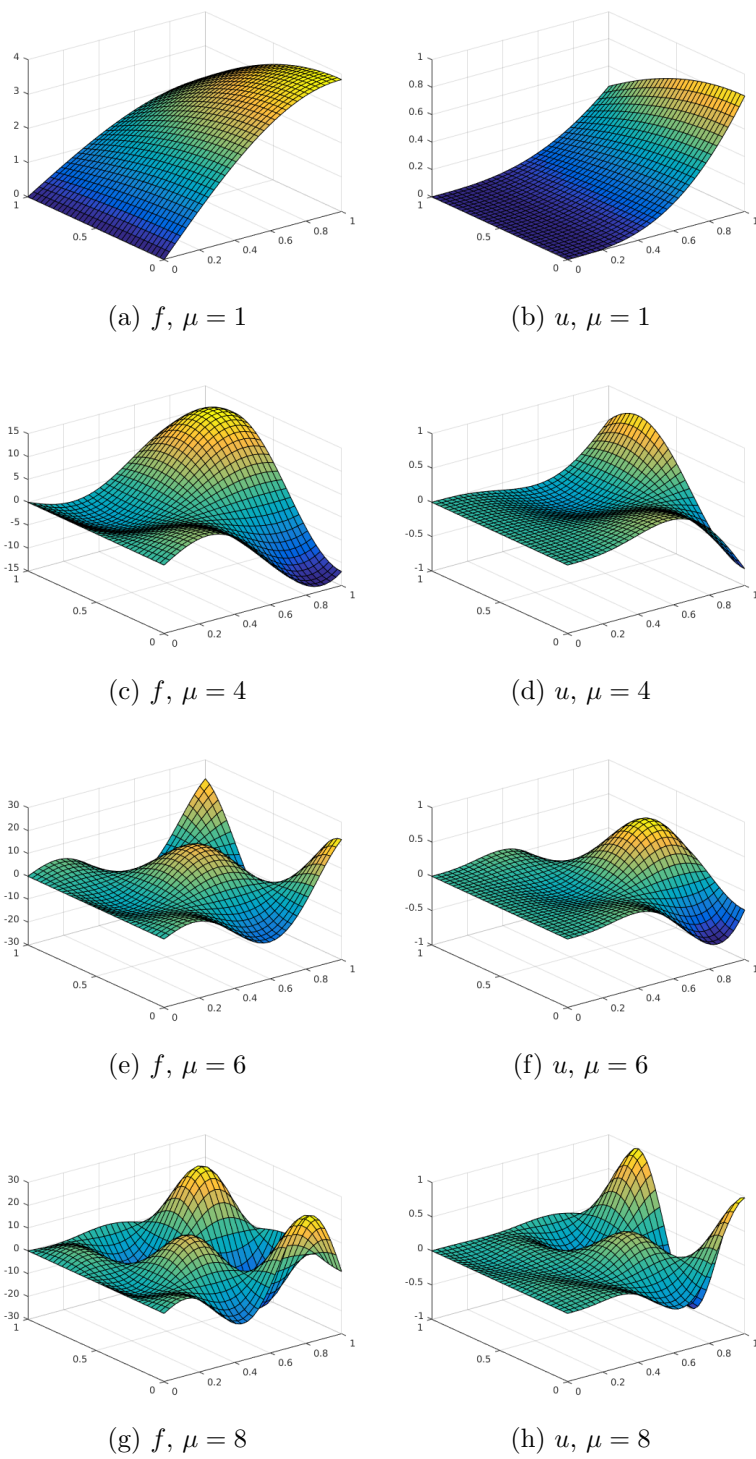


Figure 5.5: The right-hand term f and solution u of (5.22) for $\mu = 1, 4, 6,$ and 8 .

h	GMG	AMG	FMM	IC
2^{-5}	11	5	7	19
2^{-6}	14	5	6	—
2^{-7}	17	5	6	—

(a) $\mu = 1$

h	GMG	AMG	FMM	IC
2^{-5}	16	7	7	—
2^{-6}	—	7	6	—
2^{-7}	—	7	6	—

(b) $\mu = 4$

h	GMG	AMG	FMM	IC
2^{-5}	—	8	6	—
2^{-6}	—	8	6	—
2^{-7}	—	8	6	—

(c) $\mu = 6$

h	GMG	AMG	FMM	IC
2^{-5}	—	15	7	—
2^{-6}	—	15	7	—
2^{-7}	—	15	6	—

(d) $\mu = 8$

Table 5.13: Preconditioned GMRES iterations for the relative residual to reduce by six orders of magnitude for $\mu = 1, 4, 6,$ and 8 .

becomes less effective for increasing values of k where the number of iterations required for convergence exceeds *maxit*. For larger k , the FMM preconditioner requires the smallest number of iterations to converge to the predefined tolerance. Examining all the subtables in Table 5.13 shows that the FMM preconditioner achieves both mesh-independent and wavenumber-independent convergence for the given values of h and k .

5.4.5 Variable-coefficient Helmholtz equation

Similarly to the Poisson equation, we can precondition the variable-coefficient Helmholtz problem with the constant-coefficient one. We test this approach on the variable-coefficient Helmholtz equation of the form

$$\nabla \cdot (a \nabla u) + k^2 u = 1 \text{ in } \Omega,$$

$$u = 0 \text{ on } \Gamma,$$

where

$$a = 1 + \mu(\sin(m\pi x) \sin(n\pi y)).$$

μ	AMG	FMM
2^{-16}	7	4
2^{-8}	8	5
2^{-4}	10	6
2^{-2}	10	8

Table 5.14: Preconditioned GMRES iterations for the relative residual to reduce by six orders of magnitude ($h = 2^{-5}$, $m = 4$, $n = 16$, and $k = 10$).

m	n	AMG	FMM
1	1	12	5
2	2	11	6
4	4	11	5
8	8	11	5
16	16	11	5

Table 5.15: Preconditioned GMRES iterations for the relative residual to reduce by six orders of magnitude ($h = 2^{-6}$, $\mu = 2^{-4}$, and $k = 15$).

Tables 5.14, 5.15, and 5.16 show that the FMM preconditioner is competitive with the algebraic multigrid method requiring smaller numbers of iterations for different wave numbers k , amplitudes μ , and frequencies m and n .

5.4.6 Effect of FMM precision on convergence

For the results shown above, the FMM precision was set to preserve six significant digits. However, the FMM can be accelerated further by trading precision for speed. Since we are using the FMM as a preconditioner, the accuracy requirements are somewhat lower than that of general applications of FMM. Although this balance between the accuracy and speed of FMM is a critical factor for evaluating the usefulness of FMM as a preconditioner, the relation between the FMM precision and convergence rate has not been studied previously.

In Figure 5.6 the relative residual at each CG iteration is plotted against the number of iterations for FMM, AMG, GMG, and IC. The problem is the same as in Table 5.1. Three cases of FMM are used with six, four, and two significant digits of

k	AMG	FMM
5	4	4
10	7	5
15	11	5

Table 5.16: Preconditioned GMRES iterations for the relative residual to reduce by six orders of magnitude ($m = 4$, $n = 4$, $h = 2^{-6}$, $\mu = 2^{-4}$).

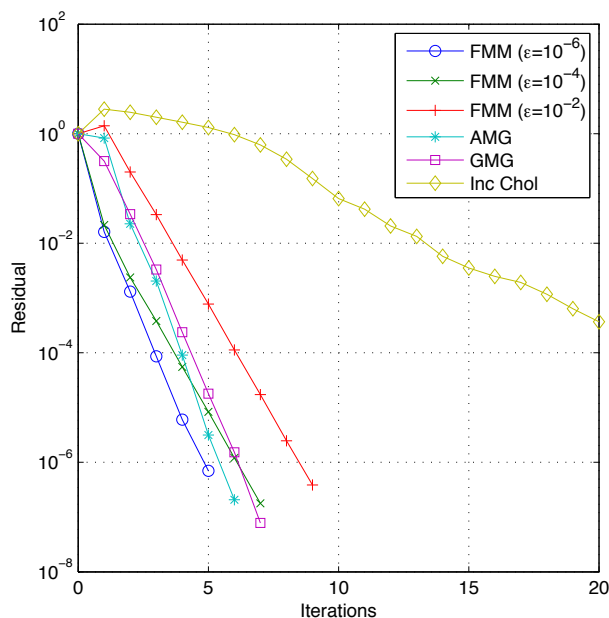


Figure 5.6: Convergence rate of the FMM preconditioner for the Poisson equation with different precision, plotted along with algebraic multigrid, geometric multigrid, and incomplete Cholesky preconditioners. The ϵ represents the precision of the FMM, where $\epsilon = 10^{-6}$ corresponds to six significant digits of accuracy.

accuracy, respectively. The $\epsilon = 10^{-6}$ case corresponds to the condition for the tests in Tables 5.1–5.4. Decreasing the FMM accuracy to four digits has little effect during the first few iterations, but slows down the convergence near the end. Decreasing the FMM accuracy further to two digits slows down the convergence somewhat, but is still much better than the incomplete Cholesky.

A similar plot is shown for the Helmholtz equation using the problem defined in (5.21) with grid spacing of $h = 2^{-5}$ and wave number $k = 7$ in Figure 5.7. The nomenclature of the legend is identical to that of Figure 5.6. In this case, we see a larger difference between the convergence rate of FMM and Multigrid. Even the FMM

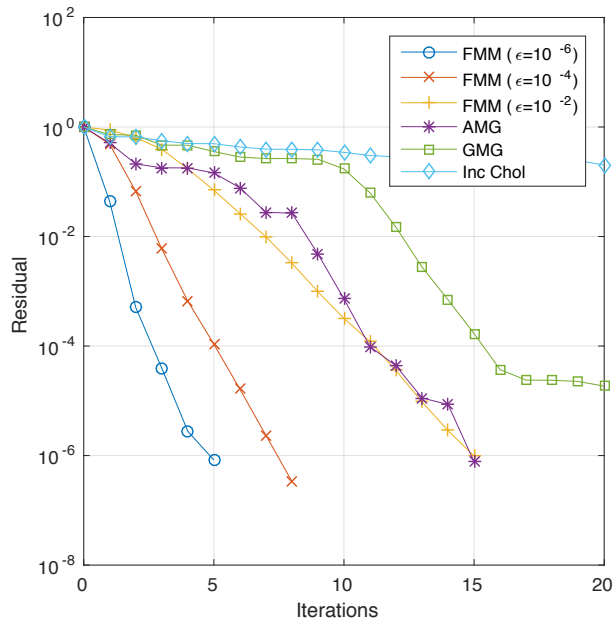


Figure 5.7: Convergence rate of the FMM preconditioner for the Helmholtz equation with different precision, plotted along with AMG, GMG, and IC preconditioners. The ϵ represents the precision of the FMM where $\epsilon = 10^{-6}$ corresponds to six significant digits of accuracy, $h = 2^{-5}$.

with the worst accuracy does better than the multigrid. We have also confirmed that the FMM preconditioner has a convergence rate that is independent of the problem size, up to moderate wave numbers of k .

Increasing the precision of the FMM past six digits does not result in any noticeable improvement because truncation error begins to dominate. We are preconditioning a matrix resulting from a FEM discretization by using an integral equation with Green's function kernels. Each has its own error, below which algebraic error need not be reduced. We show in Figure 5.8 the convergence of spatial discretization error for the FEM and BEM approaches. We use the same reference problem as in Table 5.1, which has an analytical solution. The discretization error is measured by taking the relative L^2 norm of the difference between the analytical solution and the individual numerical solutions. We see that the FEM is second order and BEM is first order. The five different values of Δx correspond to $h = \{2^{-4}, 2^{-5}, 2^{-6}, 2^{-7}, 2^{-8}\}$,

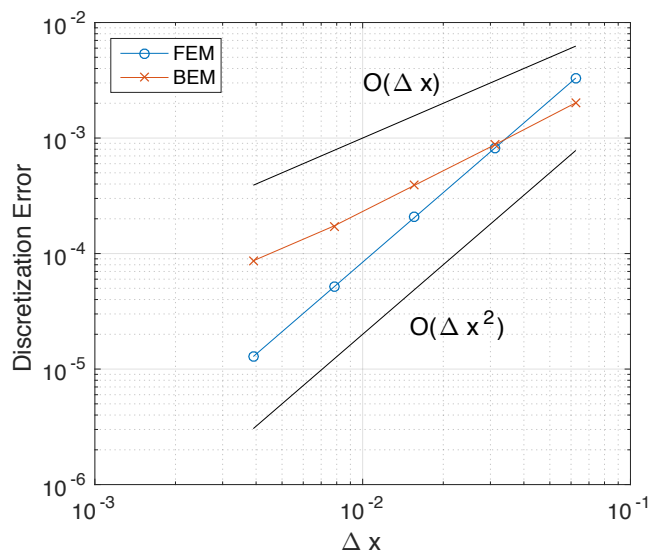


Figure 5.8: Convergence of spatial discretization error for the FEM and BEM. The relative L^2 norm of the difference between the analytical solution is plotted against the grid spacing Δx .

which were used in the previous experiments. For the current range of grid spacing, the discrepancies between the FEM and BEM truncation error is in the range of 10^{-3} to 10^{-4} .

The best preconditioners for Krylov subspace methods move the smallest eigenvalues of the coefficient matrix away from the origin. Convergence bounds for Krylov subspace methods are based on the condition number of the preconditioned system, which, for symmetric positive definite systems, as here, is the ratio of the maximum to the minimum eigenvalue. Our linear system is scaled such that the largest eigenvalue of the discrete Laplacian is bounded by four, while the smallest is proportional to the inverse mesh spacing squared. The sub-figures in Figure 5.9 show the eigenvalues clustering for the FMM-preconditioned coefficient matrix with different FMM precisions for the same reference problem as in Figure 5.7. Notice that as the FMM accuracy increases, the eigenvalues of the FMM-preconditioned matrix become better clustered and more bounded away from zero.

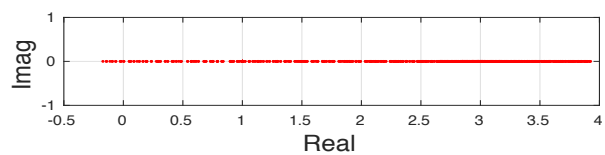
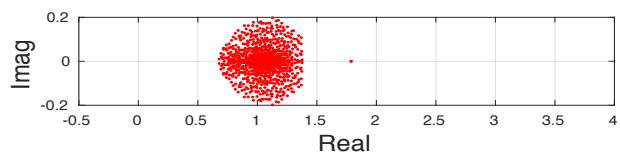
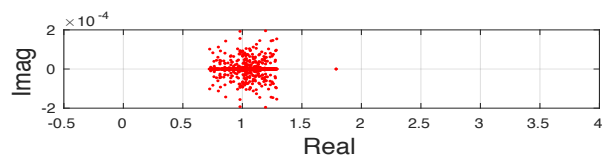
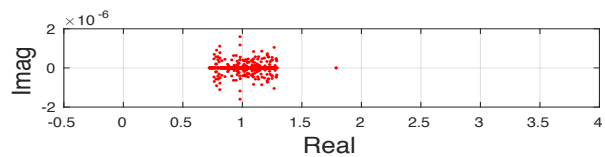
(a) $\lambda(A)$ (b) $\lambda(M^{-1}A)$, $\epsilon = 10^{-2}$ (c) $\lambda(M^{-1}A)$, $\epsilon = 10^{-4}$ (d) $\lambda(M^{-1}A)$, $\epsilon = 10^{-6}$

Figure 5.9: Eigenvalues of the coefficient matrix A and the FMM-preconditioned matrix $M^{-1}A$ with different FMM precisions $\epsilon = 10^{-2}$, 10^{-4} , and 10^{-6} , $h = 2^{-5}$.

5.5 Performance Analysis

In this section we evaluate the performance of the FMM-based preconditioner by comparing its time-to-solution to an algebraic multigrid code BoomerAMG. We have implemented our FMM-preconditioner in PETSc [130], [131] via PetIGA [132]. PetIGA is a software layer that sits on top of PETSc that facilitates NURBS-based Galerkin finite element analysis. For our present analysis, we simply use PetIGA to reproduce the same finite element discretization as the tests shown in Section 5.4, but in a high performance computing environment. All codes that were used for the current study are publicly available. A branch of PetIGA that includes the FMM preconditioner is hosted on bitbucket.

5.5.1 The Poisson equation

For the following Poisson performance evaluation, we select the first problem in Section 5.4.1 with $-\nabla^2 u = 1$ and homogeneous Dirichlet boundary conditions.

All calculations in this section were performed on the TACC Stampede system without using the coprocessors (machine description is provided in Appendix A). We used the Intel compiler (version 13.1.0.146) and configured PETSc with “COPTFLAGS=-O3 FOPTFLAGS=-O3 -with-clanguage=cxx -download-fblaslapack -download-hypre -download-metis -download-parmetis -download-superlu_dist -with-debugging=0”.

5.5.1.1 Serial results

We first evaluate the serial performance of our method using the same 2-D Poisson problem used in Section 5.4.1. We confirmed that the iteration counts shown in Table 5.1 did not change for the PETSc version of our code. Then, we measured

<https://bitbucket.org/rioyokota/petiga-fmm>

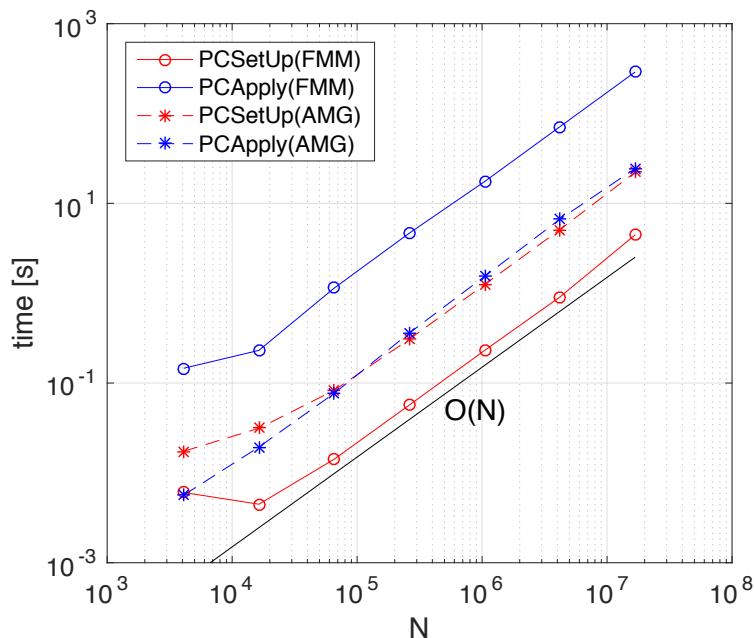


Figure 5.10: Time-to-solution for different problem sizes of the FMM and AMG preconditioners on a single core of a Xeon E5-2680.

the time-to-solution for different problem sizes. Since the domain size is $[-1, 1]$, the grid spacing of $h = \{2^{-4}, 2^{-5}, 2^{-6}, 2^{-7}, 2^{-8}\}$ in Section 5.4 correspond to a grid size of $N = \{32^2, 64^2, 128^2, 256^2, 512^2\}$. In the PETSc version, the time-to-solution improves significantly so we tested for larger problem sizes of $N = \{64^2, 128^2, 256^2, 512^2, 1024^2, 2048^2, 4096^2\}$.

The time-to-solution is plotted against the problem size N in Figure 5.10. Since we are using PETSc, we can change the preconditioner to AMG in the command line by passing the option “`-pc.type hypre`” during runtime. Therefore, the time-to-solution of BoomerAMG is shown as a reference in the same figure. For BoomerAMG we compared different relaxation, coarsening, and interpolation methods and found that

```

“-pc.hypre_boomeramg_relax_type_all backward-SOR/Jacobi
-pc.hypre_boomeramg_coarsen_type modifiedRuge-Stueben
-pc.hypre_bommeramg_interp_type classical” gives the best performance.

```

Both FMM and AMG runs are serial, where we used a single MPI process and a single thread. The majority of the time goes into the setup of the preconditioner “PC-Setup” and the actual preconditioning “PCApply,” so only these events are shown in the legend. The “PCSetup” is called only once for the entire run, while “PCApply” is called every iteration. For the present runs, both FMM and AMG required six iterations for the relative residual to drop six digits, so all runs are calling “PCApply” six times. The order of expansion for the FMM is set to $p = 6$, which gives about six significant digits of accuracy. With this accuracy for the FMM, we are still able to converge in six iterations.

By taking a closer look at Figure 5.10, one can see that both the FMM and AMG show $\mathcal{O}(N)$ asymptotic behavior. The FMM seems to have a slower preconditioning time, but a much faster setup time compared to AMG. The FMM also has a constant overhead which becomes evident when N is small. In summary, the time-to-solution of the FMM is approximately an order of magnitude larger than that of AMG for the serial runs. This is consistent with our intuition that FMM is not the preconditioner of choice for solving small problems on a single core. We will show in the following section that the FMM becomes competitive when scalability comes into the picture.

5.5.1.2 Parallel results

Using the same Poisson problem, we now compare the performance of FMM and AMG for parallel runs on Stampede. We also compare with a sparse direct solver MUMPS by invoking at runtime “-ksp_type preonly -pc_type lu -pc_factor_mat_solver_package mumps”.

The strong scaling of FMM, AMG, and MUMPS are shown in Figure 5.11. We use the largest grid size in the previous runs $N = 4096^2$. Stampede has 16 cores per node so all runs first parallelize among these cores and then among the nodes after the 16 cores are filled. The FMM strong scales quite well up to 1024 cores, while the parallel

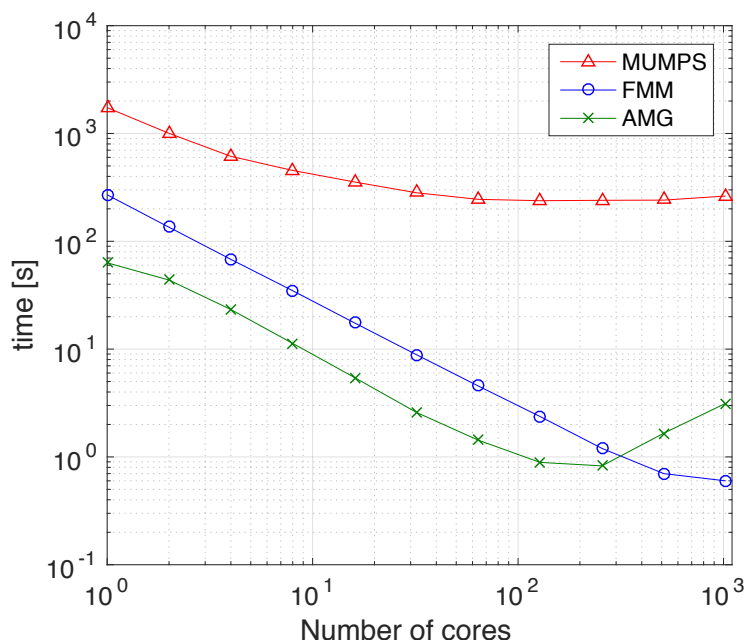


Figure 5.11: Strong scaling of the 2-D FMM and AMG preconditioners.

efficiency of AMG starts to decrease after 128 cores. The sparse direct solver has a much larger time-to-solution even on a single core, and is much less scalable than the other two hierarchical preconditioners. It is worth mentioning that the setup cost of the direct solver is dominant and so if several linear systems are solved with the same coefficient matrix then this cost is amortized. For this particular Poisson problem on this particular machine using this particular FMM code we see an advantage over BoomerAMG past 512 cores.

5.5.1.3 Extension to 3-D

The results above are all two-dimensional. A natural question that arises is whether the extension to 3-D is straightforward, and whether FMM will still be competitive as a preconditioner or not. Our results showed that a dominant part of the calculation time for the FMM preconditioner is the “PCApply” stage, which is the dual tree traversal for calculation of M2L and P2P kernels. For 3-D kernels, the M2L operation is much more complicated so the calculation time of the FMM will increase, even for

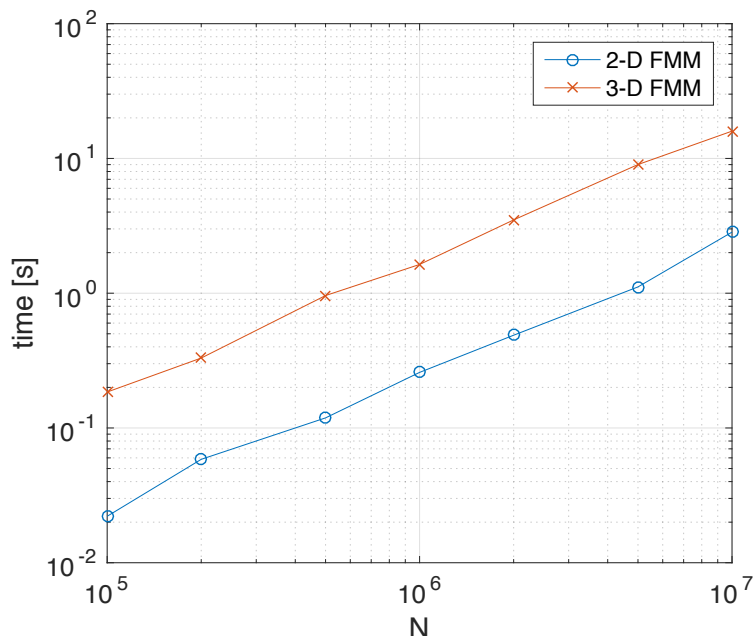


Figure 5.12: Calculation time of 2-D and 3-D FMM for the same problem size.

the same number of unknowns N .

Figure 5.12 shows the calculation time of our 2-D FMM and 3-D FMM, both for the Laplace kernel with four significant digits of accuracy on a single core of a Xeon E5-2680, 2.7 GHz CPU. The problem size N varies from 10^5 to 10^7 . We see that the 3-D FMM is about an order of magnitude slower than the 2-D FMM for the same problem size. Nevertheless, Figure 5.13 shows that the 3-D FMM preconditioner strong scales quite well up to 1024 cores for $N = 256^3$ when compared to BoomerAMG with these configurations “-pc_hypre_boomeramg_coarsen_type hmis -pc_hypre_boomeramg_interp_type ext+i -pc_hypre_boomeramg_p_max 4 -pc_hypre_boomeramg_agg_nl 1”, and to GMG with “-da_grid_x 5 -da_grid_y 5 -pc_mg_levels 5”. These runs were performed on Shaheen II (machine description is provided in Appendix A).

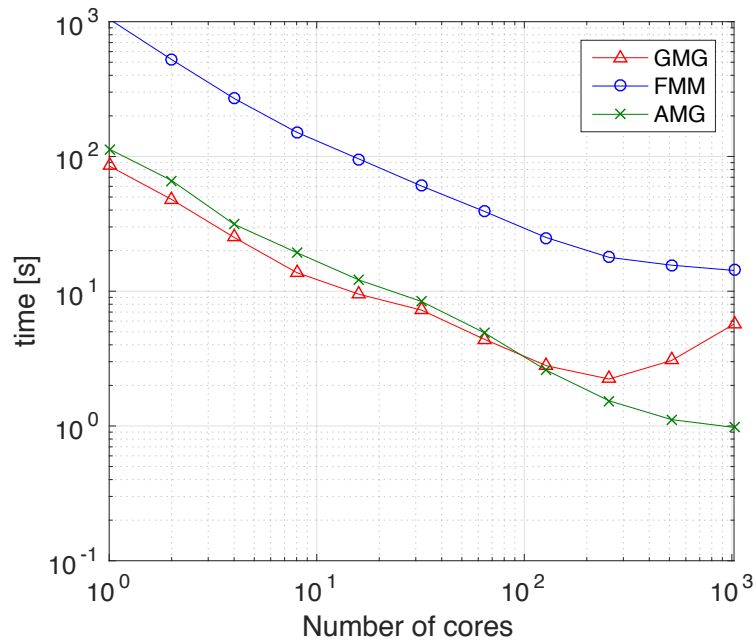


Figure 5.13: Strong scaling of the 3-D Poisson FMM and AMG preconditioners.

5.5.2 The Helmholtz equation

Figure 5.14 shows the calculation time of the 3-D Laplace and 3-D Helmholtz FMM. For the same problem size, the 3-D Helmholtz kernel is about an order of magnitude slower than the 3-D Poisson because the Helmholtz operations are more costly to compute as higher order of expansion is needed to get meaningful results for the Helmholtz problem. Higher order of expansion is necessary at the coarsest scale in the tree to resolve the oscillatory part of the Helmholtz kernel. Ideally, higher order of expansion should be used only at the coarsest levels of the octree. This feature is not supported in our implementation, however. Nevertheless, our model problems in Section 5.4.4 show that the FMM-based preconditioner requires less number of iterations to converge to the predefined tolerance when compared to the GMG, AMG, and IC preconditioners which may lead to a lower time-to-solution.

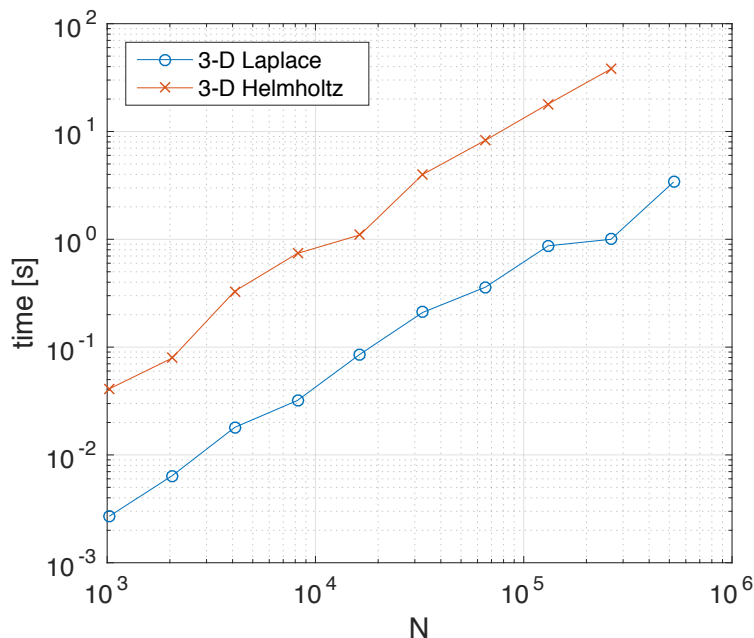


Figure 5.14: Calculation time of 3-D Laplace and 3-D Helmholtz FMM for the same problem size.

5.6 Conclusions

The Fast Multipole Method, originally developed as a free-standing solver, can be effectively combined with Krylov iteration as a scalable and highly performant preconditioner for traditional low-order finite discretizations of elliptic boundary value problems. In model problems it performs similarly to algebraic multigrid in convergence rate, while excelling in scalings where AMG becomes memory bandwidth-bound locally and/or synchronization-bound globally. No preconditioner considered in isolation can address the fundamental architectural challenges of Krylov methods for sparse linear systems, which are being simultaneously adapted to less synchronization tolerant computational environments through pipelining, but it is important to address the bottlenecks of preconditioning this most popular class of solvers by making a wide variety of tunable preconditioners available and better integrating them into the overall solver. Fast multipole-based preconditioners are demonstrably ready

to play an important role in the migration of sparse iterative solvers to the exascale.

Chapter 6

FMM as a Matrix-Free Hierarchically Low Rank Approximation

Many of the original FMM researchers have now moved on to develop algebraic variants of FMM, such as \mathcal{H} -matrix [12], \mathcal{H}^2 -matrix [134], hierarchically semi-separable (HSS) [27], hierarchically block-separable (HBS) [135], and hierarchically off-diagonal low-rank (HODLR) [136] matrices. The differences between these methods are concisely summarized by Ambikasaran & Darve [137]. These algebraic generalizations of the FMM can perform addition, multiplication, and even factorization of dense matrices with near linear complexity. This transition from analytic to algebraic did not happen suddenly, and semi-analytic variants were developed along the way [138, 139]. Optimization techniques for the FMM such as compressed translation operators and their precomputation, also fall somewhere between the analytic and algebraic extremes.

The spectrum that spans purely analytic and purely algebraic forms of these hierarchically low rank approximation methods, represents the trade-off between computation (FLOPs) and memory (Bytes). The purely analytic FMM is a matrix-free \mathcal{H}^2 -matrix-vector product, and due to its matrix-free nature it has very high arithmetic intensity (Flop/Byte) [140]. On the other end we have the purely algebraic methods, which precompute and store the entire hierarchical matrix. This results

This chapter includes results from the paper, “Fast Multipole Method as a Matrix-Free Hierarchical Low-Rank Approximation” by R. Yokota, H. Ibeid, and D. Keyes [133].

in more storage and more data movement, both vertically and horizontally in the memory hierarchy. When the cost of data movement increases faster than arithmetic operations on future architectures, the methods that compute more to store/move less will become advantageous. Therefore, it is important to consider the whole spectrum of hierarchically low rank approximation methods, and choose the appropriate method for a given pair of application and architecture.

There have been few attempts to quantitatively investigate the trade-off between the analytic and algebraic hierarchically low rank approximation methods. Previously, the applicability of the analytic variants were limited to problems with Green's functions, and could only be used for matrix-vector products but not to solve the matrix. With the advent of the kernel-independent FMM (KIFMM) [138] and inverse FMM (IFMM) [137], these restrictions no longer apply to the analytic variants. Furthermore, the common argument for using the algebraic variants because they can operate directly on the matrix without the need to pass geometric information is not very convincing. Major libraries like PETSc offer interfaces to insert one's own matrix free preconditioner as a function, and passing geometric information is something that users are willing to do if the result is increased performance. Therefore, there is no strong reason from the user perspective to be monolithically inclined to use the algebraic variants. It is rather a matter of choosing the method with the right balance between its analytic (FLOPs) and algebraic (Bytes) features.

In this chapter, we categorize the recent advances in the field of HLRA from the perspective of compute-memory trade-off in a comparison between FMM and HSS for the Laplace kernel for some simple yet representative test cases.

6.1 Hierarchically Low Rank Approximation: Analytic or Algebraic?

In this section we review the full spectrum of hierarchically low rank approximations starting from the analytic side and proceeding to the algebraic side. The spectrum is depicted in Figure 6.1, where various techniques lie between the analytic and algebraic extremes. One can choose the appropriate method for a given architecture to achieve the best performance.

6.1.1 Analytic low-rank approximation

On the analytic end of the spectrum, we have classical methods such as the Treecode [42], FMM [141, 11], and panel clustering methods [142]. These methods have extremely high arithmetic intensity (Flop/Byte) due to their matrix-free nature, and are compute-bound on most modern architectures. These are not brute force methods that do unnecessary FLOPs, but are (near) linear complexity methods that are only doing useful FLOPs, but they are still able to remain compute-bound. This is very different from achieving high FLOPs counts on dense matrix-matrix multiplication or LU decomposition that have $\mathcal{O}(N^3)$ complexity. The methods we describe in this section can approximate the same dense linear algebra calculation in $\mathcal{O}(N)$ or $\mathcal{O}(N \log N)$ time.

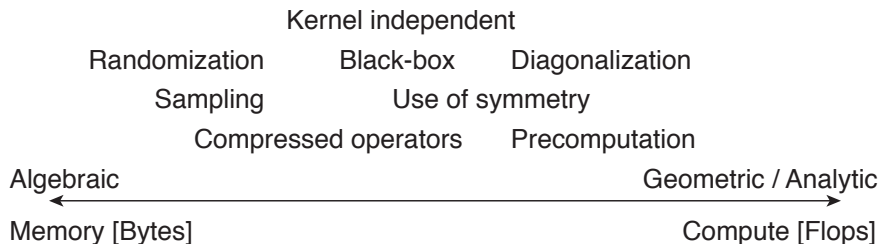


Figure 6.1: The compute-memory trade-off between the analytic and algebraic hierarchically low rank approximation methods. Various techniques lie between the analytic and algebraic extremes.

As an example of the absolute performance of the analytic variants, we refer to the Treecode implementation — “Bonsai,” which scales to the full size of Titan using 18,600 GPUs achieving 24.77 PFLOPs [143]. Bonsai’s performance comes not only from its matrix-free nature, but also from domain specific optimizations for hardcoded quadrupoles and an assumption that all charges are positive. Therefore, this kind of performance cannot be transferred to other applications that require higher accuracy. However, viewing these methods as a preconditioner instead of a direct solver significantly reduces the accuracy requirements [99, 28].

6.1.2 Semi-analytical FMM

The methods described in the previous section all require the existence of an analytical form of the multipole/local translation operator, which is kernel dependent. There are a class of methods that remove this restriction by using equivalent charges instead of multipole expansions [144, 145, 146]. A well known implementation of this method is the kernel independent FMM (KIFMM) code [138]. There are also variants that use Chebychev polynomials [37], and a representative implementation of this is the Black-box FMM [139]. As the name of these codes suggest, these variants of the FMM have reduced requirements for the information that has to be provided by the user. The translation operators are kernel independent, which frees the user from the most difficult task of having to provide an analytical form of the translation operators. It is important to note that these methods are not entirely kernel independent or black-box because the user still needs to provide the kernel dependent analytic form of the original equation they wish to calculate. Using the vocabulary of the algebraic variants, one could say that these analytical expressions for the hierarchical matrices are kernel independent only for the off-diagonal blocks, and for the diagonal blocks the analytical form is kernel dependent.

Table 6.1: Categorization of algebraic low-rank approximation methods.

Method	Hierarchical	Weak admissibility	Nested basis
\mathcal{H} -matrix [12]	yes	maybe	no
\mathcal{H}^2 -matrix [134]	yes	maybe	yes
HODLR [136]	yes	yes	no
HSS [27]/HBS [135]	yes	yes	yes
BLR [147]	no	yes	no

6.1.3 Algebraic low-rank approximation

There are many variants of algebraic low-rank approximation methods. They can be categorized based on whether they are hierarchical, whether they use weak admissibility, or if the basis is nested, as shown in Table 6.1. For the definition of admissibility see [33]. Starting from the top, \mathcal{H} -matrices [12, 148] are hierarchical, usually use standard or strong admissibility, and no nested basis. The analytic counterpart of the \mathcal{H} -matrix is the Treecode. The \mathcal{H}^2 -matrices [134, 149] are also hierarchical and use standard or strong admissibility, but unlike \mathcal{H} -matrices use a nested basis. This brings the complexity down from $\mathcal{O}(N \log N)$ to $\mathcal{O}(N)$. The analytic counterpart of the \mathcal{H}^2 -matrix is the FMM. The next three entries in Table 6.1 do not have analytic counterparts because analytic low-rank approximations do not converge under weak admissibility conditions. Hierarchical off-diagonal low-rank (HODLR) matrices [136, 28], are basically \mathcal{H} -matrices with weak admissibility conditions. Similarly, hierarchically semi-separable (HSS) [27, 150], and hierarchically block-separable (HBS) [135] matrices are \mathcal{H}^2 -matrices with weak admissibility conditions. The block low-rank (BLR) matrices [147] are a non-hierarchical version of the HODLR, with just the bottom level. A summary of implementations and their characteristics are presented in [151].

For methods that do not have weak admissibility, it is common to use geometrical information to calculate the standard/strong admissibility condition. This depen-

dence on the geometry of the algebraic variants is not ideal. There have been various proposals for algebraic clustering methods [152, 153, 39]. This problem requires even more advanced solutions for high dimension problems [154]. Stronger admissibility is also problem for parallelization since it results in more communication. There have been studies on how to partition hierarchical matrices on distributed memory [155]. There are also methods to reduce the amount of memory consumption during the construction of HSS matrices [156].

The categorization in Table 6.1 is for the hierarchical matrix structure, and any low-rank approximation method can be used with each of them during the compression phase. The singular value decomposition is the most naïve and expensive way to calculate a low-rank approximation. QR or LU decompositions can be used to find the numerical rank by using appropriate pivoting. Rank-revealing QR [157] has been proposed along with efficient pivoting strategies [158, 159, 31]. Rank-revealing LU [160] also requires efficient pivoting strategies [161, 162, 163]. Rank-revealing LU is typically faster than rank-revealing QR [30].

6.2 Experimental Results

There have been very few comparisons between the analytic and algebraic hierarchically low rank approximation methods. From a high performance computing perspective, the practical performance of highly optimized implementations of these various methods is of great interest. There have been many efforts to develop new methods in this area, which has resulted in a large amount of similar methods with different names without a clear overall picture of their relative performance on modern HPC architectures. The trend in architecture where arithmetic operations are becoming cheap compared to data movement, is something that must be considered carefully when predicting which method will perform better on computers of the future.

We acknowledge that the comparisons we present here are far from complete,

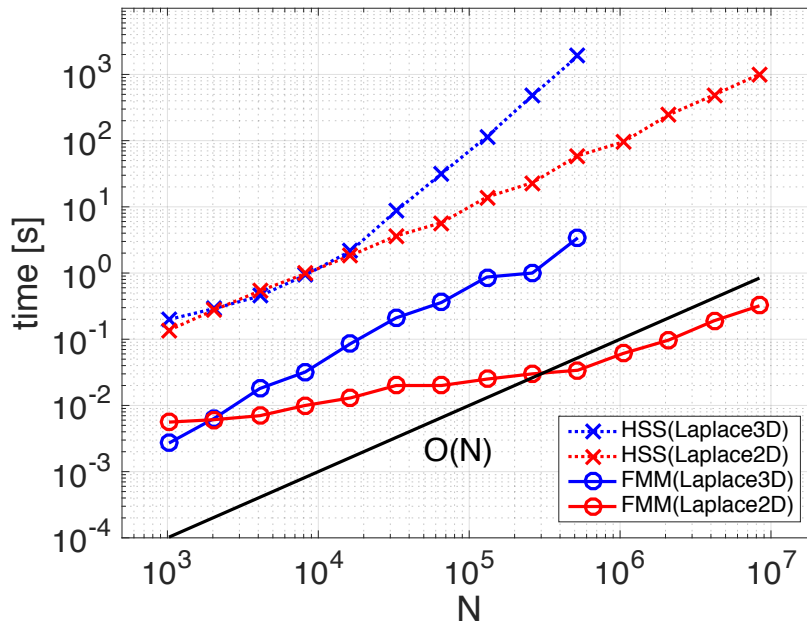


Figure 6.2: Elapsed time for the matrix-vector multiplication using FMM and HSS for different problem sizes.

and much more comparisons between all the different methods are needed in order to achieve our long term objective. The limitation actually comes from the lack of highly optimized implementations of these methods that are openly available to us at the moment.

6.2.1 Matrix-vector multiplication

In this section we compare exaFMM – a highly optimized implementation of FMM, with STRUMPACK – a highly optimized implementation of HSS. We select the 2-D and 3-D Laplace equation on uniform lattices as test cases. For HSS we directly construct the compressed matrix by calling the Green’s function in the randomized low-rank approximation routine. We perform the matrix-vector multiplication using the FMM and HSS, and measure the time for the compression/precalculation and application of the matrix-vector multiplication. We also measure the peak memory consumption of both methods.

The elapsed time for the FMM and HSS for different problem sizes is shown in

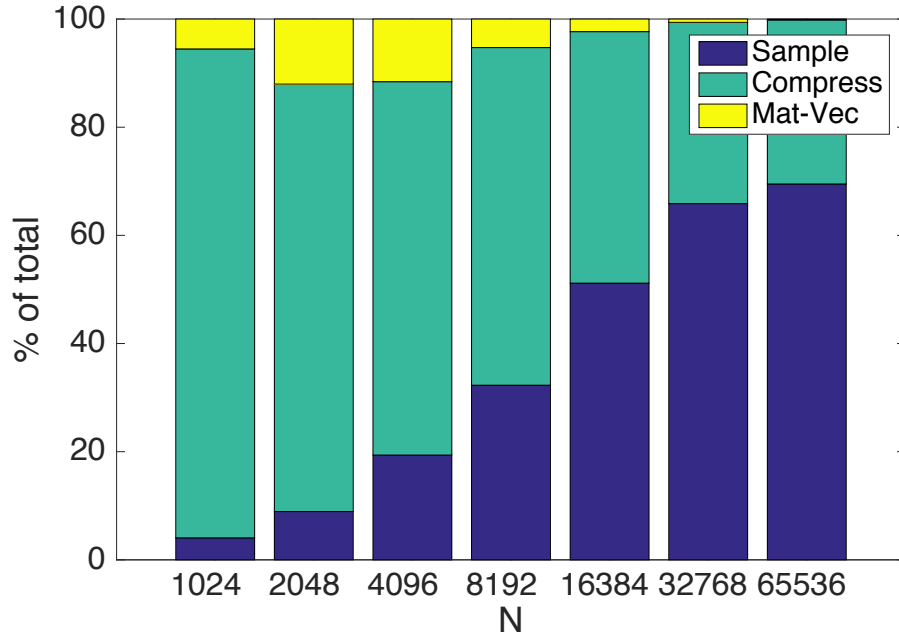


Figure 6.3: Percentage of the computation time of HSS for different problem sizes.

Figure 6.2. In order to isolate the effect of the thread scalability of the two methods, these runs are performed on a single core of a 12-core Ivy Bridge (E5-2695 v2). For the 2-D Laplace equation, the FMM shows some overhead for small N , but is about three orders of magnitude faster than HSS for larger problems. For the 3-D Laplace equation, the FMM is about 2 orders of magnitude faster than HSS for smaller N , but HSS exhibits non-optimal behavior for large N because the rank keeps growing.

The large difference in the computational time is actually coming from the heavy computation in the sampling phase and compression phase of the HSS. In Figure 6.3, we show the percentage of the computation time of HSS for different problem sizes N . “Sample” is the sampling time, “Compress” is the compression time, and “Mat-Vec” is the matrix-vector multiplication time. We can see that the sampling is taking longer and longer as the problem size increases. This is because the rank k increases with the problem size N , and both sampling and compression time increase with the k and N .

The peak memory usage of FMM and HSS is shown in Figure 6.4 for the 3-D

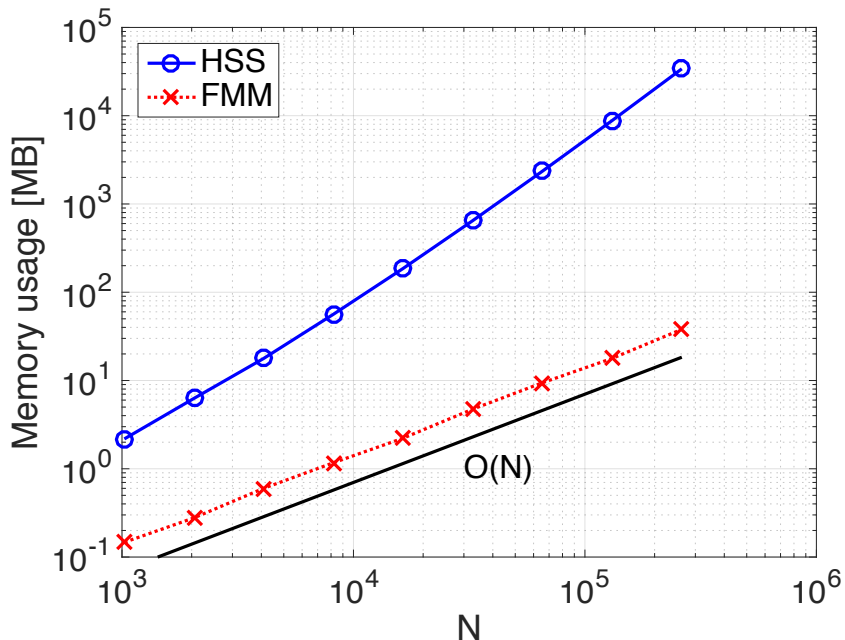


Figure 6.4: Peak memory usage of FMM and HSS for the 3-D Laplace equation.

Laplace equation. We see that the FMM has strictly $\mathcal{O}(N)$ storage requirements, but since the rank in the HSS grows for 3-D kernels it does not show the ideal $\mathcal{O}(N \log N)$ behavior. The disadvantage of HSS is two-fold. First of all, its algebraic nature requires it to store the compressed matrix, where as the FMM is analytic and therefore matrix-free. Secondly, the weak admissibility causes the rank to grow for 3-D problems, and with that the memory consumption grows at a suboptimal complexity.

6.3 Conclusions

We have shown the contrast between the analytical and algebraic hierarchically low rank approximations, by reviewing the contributions over the years and placing them along the analytical-algebraic spectrum. The relation between Treecode, FMM, KIFMM, black-box FMM, \mathcal{H} -matrix, \mathcal{H}^2 -matrix, HODLR, HSS, HBS, and BLR were explained from the perspective of compute-memory trade-off.

Our comparison benchmarks between FMM and HSS are still preliminary tests for

a very simple case. However, they clearly demonstrate the magnitude of the difference that lies between the various hierarchically low rank approximation methods.

Chapter 7

Summary and Future Work

7.1 Summary

The main contributions and results of this thesis can be summarized as follows.

- We discuss the challenges for FMM on current parallel computers and future exascale architectures, with a focus on inter-node communication. We also develop a performance model that considers the communication patterns of the FMM for spatially uniform distributions and observe a good match between our model and the actual communication time on four HPC systems, when latency, bandwidth, network topology, and multi-core penalties are all taken into account.
- We discuss the performance and scaling of FMMs for molecular dynamics simulations of uniformly distributed particles on the K computer. To measure the performance of the K computer, we performed all-atom classical molecular dynamics simulations of two systems: target proteins in a solvent, and target proteins in an environment of molecular crowders that mimic the conditions of a living cell. Using the full system, we achieved 4.4 PFLOPS during a 520 million-atom simulation with cutoff of 28 Å.
- We demonstrate that, beyond its traditional use as a solver in problems for which explicit free-space kernel representations are available, the FMM has ap-

plicability as a preconditioner in finite domain elliptic boundary value problems, by equipping it with boundary integral capability for satisfying conditions at finite boundaries and by wrapping it in a Krylov method for extensibility to more general operators.

- We provide a review of the recent advancements in the field of HLRA from both analytical and algebraic perspectives, and present a comparative benchmark of highly optimized implementations of contrasting methods for some simple yet representative test cases.
- We describe all our tests in reproducible detail with freely available codes and outline directions for further extensibility.

7.2 Future Research Work

The work presented in this thesis can be extended in the following directions.

7.2.1 Nonuniform distributions

In Chapter 3, we limited our FMM analysis and performance modeling to the uniform distribution which is applicable to our molecular dynamic simulations presented in Chapter 4. However, for many other practical applications, the distribution of points results in an adaptive tree. Modeling nonuniform distributions in general is a hard problem. However, recent work by Yokota *et al.* [164] showed that the tightest upper bound of our model still holds for the nonuniform case with adaptive tree structures. Same complexity also holds for algebraic variants of FMM, such as \mathcal{H} -matrices and HSS, wherever an upper bound exists on the number of blocks in a block row in the low-rank representation.

7.2.2 Future vision of the macromolecular crowding effect

Large-scale and all-atom simulations are essential for a deep understanding of the macromolecular crowding effect. In Chapter 4, we obtained a prevision of the macromolecular crowding effect from all-atom simulations in a large-scale system containing about 1.35 million atoms. However, the simulation lengths were insufficient for equilibrium of the systems. Longer time simulations can provide clear information on the conformation and dynamics of all macromolecules in a crowded environment. Innovations in the field of high performance computing would lead to an understanding of life phenomena such as immune responses and intracellular signaling, which are topics that attract a lot of medical attention.

Our MD simulation exhibits $0.2\mu\text{m}$ of spatial scale, while a typical biological cell has a size of at least $1\mu\text{m}$. Exascale supercomputing will enable cellular scale all-atom simulations, and our simulation is the first step towards molecular simulations of a whole cell.

7.2.3 Practical applications of the FMM preconditioner

The current trend in computer architecture favors algorithms with data locality and asynchronicity. Improving the data locality and asynchronicity of Krylov iterations is an interesting topic all on its own [165], but in Chapter 5 we focus on the preconditioner. The three important keywords in our approach are *hierarchical*, *approximate*, and *matrix-free*. The *hierarchy* provides a processor scale-independent $\mathcal{O}(N)$ arithmetic complexity and an $\mathcal{O}(\log P)$ communication complexity, where N is the number of unknowns and P is the number of processes. The *approximation* decreases the asymptotic constant of $\mathcal{O}(N)$ and $\mathcal{O}(\log P)$ by trading off the accuracy. These features are shared by methods like multigrid and \mathcal{H} -matrices. The third feature *matrix-free* will further decrease the asymptotic constant of the communication, but

will increase the asymptotic constant of the computation. Such approach will make more and more sense as the computation becomes cheaper compared to data movement on future architectures. The combination of these three features can be achieved by using the FMM as a preconditioner.

In Chapter 5, we showed that the fast multipole method can be successfully coupled to the boundary element method to give an effective preconditioner for the Poisson, Stokes, and Helmholtz problems. Additional algorithmic development, additional testing of implementations on emerging architectures, and exploiting preconditioning on more practical applications are necessary to more fully define the niche in which FMM is the preconditioner of choice.

An interesting application for future work is the solution of the 3-D Helmholtz acoustic problem with perfectly matched layer (PML) boundary conditions. This is a prerequisite for acoustic full-waveform inversion of marine and land seismic data. Another application is to use the FMM-based preconditioner for pressure corrections and other Laplace inversions that arise in multiphysics problems. In such problems, the pressure is determined at every node at every time step for 10,000–100,000 times per run. This accounts for 50%–80% of the CPU time, depending on the application. Another possibility is to extend the FMM preconditioner to other Green’s functions, like those of frequency-domain Maxwell’s equations that are used to describe electromagnetic and optical phenomena.

7.2.4 Benchmarking HLRA based methods

The topic of investigating the trade-off between analytic and algebraic hierarchical low-rank approximation methods is broad. In Chapter 6, we limited our investigation to the compute-memory trade-off in a comparison between highly optimized implementations of contrasting methods for some simple yet representative test cases.

Our comparisons are far from complete, and more thorough comparisons between

all the different methods are needed in order to be able to choose the appropriate method for a given pair of application and architecture. The limitation comes from the lack of highly optimized implementations of these methods that are openly available to us at the moment.

REFERENCES

- [1] TOP500.org, *TOP500 Supercomputer Site*, 2016. [Online]. Available: www.top500.org
- [2] H. C. Elman, D. J. Silvester, and A. J. Wathen, *Finite Elements and Fast Iterative Solvers: With applications in incompressible fluid dynamics*. Oxford: Oxford University Press, 2005.
- [3] R. M. Martin, *Electronic Structure: Basic Theory and Practical Methods*. Cambridge, UK: Cambridge University Press, 2004.
- [4] W. Hundsdorfer and J. G. Verwer, *Numerical Solution of Time-Dependent Advection-Diffusion-Reaction Equations*. Germany: Springer-Verlag, 2003.
- [5] A. L. Laird and M. B. Giles, “Preconditioned iterative solution of the 2d Helmholtz equation,” Oxford University Computing Laboratory, Tech. Rep. NA-02/12, 2002.
- [6] A. Arnold and M. Ehrhardt, “Discrete transparent boundary conditions for wide angle parabolic equations in underwater acoustics,” *Journal of Computational Physics*, vol. 145, no. 2, pp. 611 – 638, 1998. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0021999198960430>
- [7] G. J. Fix and S. P. Marin, “Variational methods for underwater acoustic problems,” *Journal of Computational Physics*, vol. 28, no. 2, pp. 253 – 270, 1978. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0021999178900372>

- [8] R.-E. Plessix and W. Mulder, “Frequency-domain finite-difference amplitude-preserving migration,” *Geophysical Journal International*, vol. 157, no. 3, pp. 975–987, 2004.
- [9] H. Urbach and R. Merkkx, “Finite element simulation of electromagnetic plane wave diffraction at gratings for arbitrary angles of incidence,” in *Mathematical and Numerical Aspects of Wave Propagation Phenomena, Proceedings of SIAM First Conference on Mathematical and Numerical Aspects of Wave Propagation*, 1991, pp. 89–99.
- [10] U. Trottenberg, C. Oosterlee, and A. Schüller, *Multigrid*. London: Academic Press, 2001.
- [11] L. Greengard and V. Rokhlin, “A fast algorithm for particle simulations,” *Journal of Computational Physics*, vol. 73, no. 2, pp. 325–348, 1987.
- [12] W. Hackbusch, “A sparse matrix arithmetic based on H-matrices, Part I: Introduction to H-matrices,” *Computing*, vol. 62, pp. 89–108, 1999.
- [13] K. Czechowski, C. McClanahan, C. Battaglini, K. Iyer, P.-K. Yeung, and R. Vuduc, “On the communication complexity of 3d FFTs and its implications for exascale,” in *Proceedings of the 26th ACM International Conference on Supercomputing*, 2012, pp. 205–214.
- [14] J. Jung, C. Kobayashi, T. Imamura, and Y. Sugita, “Parallel implementation of 3d FFT with volumetric decomposition schemes for efficient molecular dynamics simulations,” *Computer Physics Communications*, vol. 200, pp. 57 – 65, 2016. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0010465515004063>
- [15] I. Lashuk, A. Chandramowlishwaran, H. Langston, T.-A. Nguyen, R. Sampath, A. Shringarpure, R. Vuduc, L. Ying, D. Zorin, and G. Biros, “A massively parallel adaptive fast multipole method on heterogeneous architectures,” *Communications of the ACM*, vol. 55, no. 5, pp. 101–109, 2012.
- [16] G. H. Golub and C. F. van Loan, *Matrix Computations*. Baltimore: The John Hopkins University Press, 1996.

- [17] G. H. Golub and R. S. Varga, “Chebyshev semi iterative methods, successive overrelaxation iterative methods and second order Richardson iterative methods. Part I.” *Numerische Mathematik*, vol. 3, pp. 147–156, 1961.
- [18] M. R. Hestenes and E. Stiefel, “Methods of conjugate gradients for solving linear systems,” *Journal of Research of the National Bureau of Standards*, vol. 49, pp. 409–436, 1952.
- [19] C. C. Paige and M. A. Saunders, “Solution of sparse indefinite systems of linear equations,” *SIAM Journal on Numerical Analysis*, vol. 12, pp. 617–629, 1975.
- [20] Y. Saad and M. H. Schultz, “GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems,” *SIAM Journal on Scientific and Statistical Computing*, vol. 7, pp. 856–869, 1986.
- [21] W. Hackbusch, B. Khoromskij, and S. Sauter, “On H^2 -Matrices,” in *Lectures on Applied Mathematics*, H.-J. Bungartz, R. Hoppe, and C. Zenger, Eds. Springer Berlin Heidelberg, 2000, pp. 9–29. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-59709-1_2
- [22] H. Gahvari, A. H. Baker, M. Schulz, U. M. Yang, K. E. Jordan, and W. Gropp, “Modeling the performance of an algebraic multigrid cycle on HPC platforms,” in *ICS '11 Proceedings of the International Conference on Supercomputing*, 2011, pp. 172–181.
- [23] M. F. Adams, J. Brown, M. Knepley, and R. Samtaney, “A multigrid technique for data locality,” *SIAM Journal on Scientific Computing*, 2016.
- [24] A. Brandt, “Multi-level adaptive techniques (MLAT) for partial differential equations: Ideas and software,” in *Mathematical Software*, J. R. Rice, Ed. Academic Press, 1977, vol. III, pp. 277–318.
- [25] A. H. Baker, R. D. Falgout, T. Gamblin, T. V. Kolev, M. Schulz, and U. M. Yang, “Scaling algebraic multigrid solvers: On the road to exascale,” in *Competence in High Performance Computing*, C. Bischof, H.-G. Hegering, W. E. Nagel, and G. Wittum, Eds. Springer, 2012, pp. 215–226.

- [26] P. S. Vassilevski and U. M. Yang, “Reducing communication in algebraic multigrid using additive variants,” *Numerical Linear Algebra with Applications*, vol. 21, pp. 275–296, 2014.
- [27] S. Chandrasekaran, M. Gu, and T. Pals, “A fast ULV decomposition solver for hierarchically semiseparable representations,” *SIAM Journal on Matrix Analysis and Applications*, vol. 28, no. 3, pp. 603–622, 2006.
- [28] A. Aminfar, S. Ambikasaran, and E. Darve, “A fast block low-rank dense solver with applications to finite-element matrices,” *Journal of Computational Physics*, vol. 304, pp. 170 – 188, 2016. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0021999115006750>
- [29] K. L. Ho and L. Greengard, “A fast direct solver for structured linear systems by recursive skeletonization,” *SIAM Journal on Scientific Computing*, vol. 34, no. 5, pp. A2507–A2532, 2012.
- [30] C. T. Pan, “On the existence and computation of rank-revealing LU factorizations,” *Linear Algebra and its Applications*, vol. 316, pp. 199–222, 2000.
- [31] M. Gu and S. C. Eisenstat, “Efficient algorithms for computing a strong rank-revealing QR factorization,” *SIAM Journal on Scientific Computing*, vol. 17, no. 4, pp. 848–869, 1996.
- [32] W. Y. Kong, J. Bremer, and V. Rokhlin, “An adaptive fast direct solver for boundary integral equations in two dimensions,” *Applied and Computational Harmonic Analysis*, vol. 31, pp. 346–369, 2011.
- [33] L. Grasedyck and W. Hackbusch, “Construction and arithmetics of H-matrices,” *Computing*, vol. 70, pp. 295–334, 2003.
- [34] E. Liberty, F. Woolfe, P. G. Martinsson, V. Rokhlin, and M. Tygert, “Randomized algorithms for the low-rank approximation of matrices,” *PNAS*, vol. 104, no. 51, pp. 20 167–20 172, 2007.

- [35] S. Rjasanow, “Adaptive cross approximation of dense matrices,” in *International Association for Boundary Element Methods*, UT Austin, TX, USA, May 28-30 2002.
- [36] S. Börm, “Hybrid cross approximation of integral operators,” *Numerische Mathematik*, vol. 101, pp. 221–249, 2005.
- [37] A. Dutt, M. Gu, and V. Rokhlin, “Fast algorithms for polynomial interpolation integration and differentiation,” *SIAM Journal on Numerical Analysis*, vol. 33, no. 5, pp. 1689–1711, 1996.
- [38] L. Greengard, D. Gueyffier, P. G. Martinsson, and V. Rokhlin, “Fast direct solvers for integral equations in complex three dimensional domains,” *Acta Numerica*, vol. 18, pp. 243–275, 2009.
- [39] L. Grasedyck, W. Hackbusch, and R. Kriemann, “Performance of H-LU preconditioning for sparse matrices,” *Computational Methods in Applied Mathematics*, vol. 8, no. 4, pp. 336–349, 2008.
- [40] A. Gholami, D. Malhotra, H. Sundar, and G. 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*, vol. 38, no. 3, pp. C280–C306, 2016.
- [41] W. T. Rankin, “Efficient parallel implementations of multipole based N-body algorithm,” Ph.D. dissertation, Duke University, 1999.
- [42] J. Barnes and P. Hut, “ $O(N \log N)$ force-calculation algorithm,” *Nature*, vol. 324, pp. 446–449, 1986.
- [43] L. Greengard and R. V., “On the efficient implementation of the fast multipole algorithm,” Yale University, Research Report RR-602, 1988.
- [44] N. L. Gorn and D. V. Berkov, “Adaptation and performance of the fast multipole method for dipolar systems,” *Journal of Magnetism and Magnetic Materials*, vol. 272-276, pp. 698–700, 2004.

- [45] J. Carrier, L. Greengard, and V. Rokhlin, “A fast adaptive multipole algorithm for particle simulations,” *SIAM Journal on Scientific and Statistical Computing*, vol. 9, no. 4, pp. 669–686, 1988.
- [46] M. S. Warren and J. K. Salmon, “A portable parallel particle program,” *Computer Physics Communications*, vol. 87, pp. 266–290, 1995.
- [47] S.-H. Teng, “Provably good partitioning and load balancing algorithms for parallel adaptive N-body simulation,” *SIAM Journal on Scientific Computing*, vol. 19, no. 2, pp. 635–656, 1998.
- [48] W. Dehnen, “A hierarchical $O(N)$ force calculation algorithm,” *Journal of Computational Physics*, vol. 179, no. 1, pp. 27–42, 2002.
- [49] N. A. Gumerov and R. Duraiswami, “Fast multipole methods on graphics processors,” *Journal of Computational Physics*, vol. 227, pp. 8290–8313, 2008.
- [50] M. S. Warren and J. K. Salmon, “Astrophysical N-body simulation using hierarchical tree data structures,” in *Proceedings of the 1992 ACM/IEEE Conference on Supercomputing*, 1992, pp. 570–576.
- [51] R. Yokota, “An FMM based on dual tree traversal for many-core architectures,” *Journal of Algorithms and Computational Technology*, vol. 7, no. 3, pp. 301–324, 2013.
- [52] H. Ibeid, R. Yokota, and D. Keyes, “A performance model for the communication in fast multipole methods on high-performance computing platforms,” *International Journal of High Performance Computing Applications*, vol. 30, no. 4, pp. 423–437, 2016.
- [53] M. J. Clement and M. J. Quinn, “Symbolic performance prediction of scalable parallel programs,” in *Proceedings of the International Parallel Processing Symposium*, April 1995, pp. 635–639.
- [54] C. L. Mendes and D. A. Reed, “Integrated compilation and scalability analysis for parallel systems,” *International Conference on Parallel Architectures and Compilation Techniques (PACT’98)*, pp. 385–392, October 1998.

- [55] C. L. Mendes, “Performance scalability prediction on multicomputers,” Ph.D. dissertation, University of Illinois, Urbana-Champaign, May 1997.
- [56] A. Snavely, N. Wolter, and L. Carrington, “Modeling application performance by convolving machine signatures with application profiles,” in *Proceeding of the IEEE Workshop on Workload Characterization*, December 2001, pp. 149–156.
- [57] L. DeRose and D. A. Reed, “SvPablo: A multi-language, architecture-independent performance analysis system,” in *Proceeding of the International Conference on Parallel Processing*, August 1999, pp. 311–318.
- [58] W. D. Gropp, D. Kaushik, D. Keyes, and B. Smith, “Toward realistic performance bounds for implicit CFD codes,” in *Proceedings of Parallel CFD’99*, May 1999, pp. 23–26.
- [59] I. T. Foster and P. H. Worley, “Parallel algorithms for the spectral transform method,” *SIAM Journal on Scientific and Statistical Computing*, vol. 18, no. 3, pp. 806–837, 1997.
- [60] D. Kerbyson, H. Alme, A. Hoisie, F. Petrini, A. Wasserman, and M. Gittings, “Predictive performance and scalability modeling of a large-scale application,” in *Proceedings of the 2001 ACM/IEEE conference on Supercomputing*, 2001, pp. 1–12.
- [61] H. Gahvari, W. Gropp, K. E. Jordan, M. Schulz, and U. M. Yang, “Algebraic multigrid on a dragonfly network: First experience on a Cray XC30,” in *Proceeding of the 5th International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS14)*, November 2014.
- [62] P. H. Worley, “Performance evaluation of the IBM SP and the Compaq AlphaServer SC,” in *Proceeding of the ACM International Conference of Supercomputing 2000*, 2000, pp. 235–244.
- [63] J. M. Perez-Jorda and W. Yang, “On the scaling of multipole methods for particle-particle interactions,” *Chemical Physics Letters*, vol. 282, pp. 71–78, 1998.

- [64] P. Jetley, L. Wesolowski, F. Gioachin, L. V. Kale, and T. R. Quinn, “Scaling hierarchical N-body simulations on GPU clusters,” in *SC '10 Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, 2010, pp. 1–11.
- [65] A. Chandramowliswaran, S. Williams, L. Oliker, I. Lashuk, G. Biros, and R. Vuduc, “Optimizing and tuning the fast multipole method for state-of-the-art multicore architectures,” in *Proceedings of the International Parallel Distributed Processing Symposium (IPDPS)*, 2010, pp. 1–12.
- [66] I. Lashuk, A. Chandramowliswaran, H. Langston, T.-A. Nguyen, R. Sampath, A. Shringarpure, R. Vuduc, L. Ying, D. Zorin, and G. Biros, “A massively parallel adaptive fast multipole method on heterogeneous architectures,” in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, 2009, pp. 1–12.
- [67] I. Foster, *Designing and Building Parallel Programs*. Addison-Wesley, 1995.
- [68] P. Luszczek and J. Dongarra, “Introduction to the HPC Challenge Benchmark Suite,” University of Tennessee, Knoxville, Technical Report ICL-UT-05-01, March 2005.
- [69] M. Lee, N. Malaya, and R. D. Moser, “Petascale direct numerical simulation of turbulent channel flow on up to 768k cores,” in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, Denver, CO, USA, November 16-22 2013.
- [70] Y. Ohno, R. Yokota, H. Koyama, G. Morimoto, A. Hasegawa, G. Masumoto, N. Okimoto, Y. Hirano, H. Ibeid, T. Narumi, and M. Taiji, “Petascale molecular dynamics simulation using the fast multipole method on K computer,” *Computer Physics Communications*, vol. 185, pp. 2575–2585, 2014.
- [71] G. Rivas, F. Ferrone, and J. Herzfeld, “Life in a crowded world,” *EMBO reports*, vol. 5, no. 1, pp. 23–27, 2004. [Online]. Available: <http://embor.embopress.org/content/5/1/23>

- [72] N. A. Chebotareva, B. I. Kurganov, and N. B. Livanova, "Biochemical effects of molecular crowding," *Biochemistry (Moscow)*, vol. 69, no. 11, pp. 1239–1251, 2004. [Online]. Available: <http://dx.doi.org/10.1007/s10541-005-0070-y>
- [73] J. F. Watson and V. P. B. Jr., "Biologic activity of digoxin-specific antisera," *The Journal of Clinical Investigation*, vol. 51, no. 3, pp. 638–648, 3 1972. [Online]. Available: <http://www.jci.org/articles/view/106853>
- [74] N. A. Chebotareva, I. E. Andreeva, V. F. Makeeva, N. B. Livanova, and B. I. Kurganov, "Effect of molecular crowding on self-association of phosphorylase kinase and its interaction with phosphorylase *b* and glycogen," *Journal of Molecular Recognition*, vol. 17, no. 5, pp. 426–432, 2004.
- [75] T. Darden, D. York, and L. Pedersen, "Particle mesh Ewald: An $n \log(n)$ method for Ewald sums in large systems," *The Journal of Chemical Physics*, vol. 98, no. 12, 1993.
- [76] A. P. Minton, "Implications of macromolecular crowding for protein assembly," *Current Opinion in Structural Biology*, vol. 10, no. 1, pp. 34 – 39, 2000. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0959440X99000457>
- [77] D. Van Der Spoel, E. Lindahl, B. Hess, G. Groenhof, A. E. Mark, and H. J. Berendsen, "GROMACS: fast, flexible, and free," *Journal of Computational Chemistry*, vol. 26, no. 16, pp. 1701–1718, 2005.
- [78] J.-P. Ryckaert, G. Ciccotti, and H. J. Berendsen, "Numerical integration of the cartesian equations of motion of a system with constraints: molecular dynamics of n-alkanes," *Journal of Computational Physics*, vol. 23, no. 3, pp. 327 – 341, 1977. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/0021999177900985>
- [79] H. C. Andersen, "Rattle: A velocity version of the shake algorithm for molecular dynamics calculations," *Journal of Computational Physics*, vol. 52, no. 1, pp. 24 – 34, 1983. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/0021999183900141>

- [80] S. Miyamoto and P. A. Kollman, “Settle: An analytical version of the shake and rattle algorithm for rigid water models,” *Journal of Computational Chemistry*, vol. 13, no. 8, pp. 952–962, 1992. [Online]. Available: <http://dx.doi.org/10.1002/jcc.540130805>
- [81] Y. Shan, J. L. Klepeis, M. P. Eastwood, R. O. Dror, and D. E. Shaw, “Gaussian split Ewald: A fast Ewald mesh method for molecular simulation,” *The Journal of Chemical Physics*, vol. 122, no. 5, p. 054101, 2005.
- [82] P. H. Hünenberger and J. A. McCammon, “Ewald artifacts in computer simulations of ionic solvation and ion–ion interaction: a continuum electrostatics study,” *The Journal of Chemical Physics*, vol. 110, no. 4, pp. 1856–1872, 1999.
- [83] P. E. Smith and B. M. Pettitt, “Ewald artifacts in liquid state molecular dynamics simulations,” *The Journal of Chemical Physics*, vol. 105, no. 10, 1996.
- [84] F. Figueirido, G. S. Del Buono, and R. M. Levy, “On finite-size effects in computer simulations using the Ewald potential,” *The Journal of Chemical Physics*, vol. 103, no. 14, 1995.
- [85] X. Wu and B. R. Brooks, “Isotropic periodic sum: A method for the calculation of long-range interactions,” *The Journal of Chemical Physics*, vol. 122, no. 4, p. 044107, 2005.
- [86] D. Wolf, P. Keblinski, S. Phillpot, and J. Eggebrecht, “Exact method for the simulation of coulombic systems by spherically truncated, pairwise r^{-1} summation,” *The Journal of Chemical Physics*, vol. 110, no. 17, pp. 8254–8282, 1999.
- [87] I. Fukuda, Y. Yonezawa, and H. Nakamura, “Molecular dynamics scheme for precise estimation of electrostatic interaction via zero-dipole summation principle,” *The Journal of Chemical Physics*, vol. 134, no. 16, p. 164107, 2011.
- [88] D. F. Richards, J. N. Glosli, B. Chan, M. Dorr, E. W. Draeger, J.-L. Fattebert, W. D. Krauss, T. Spelce, F. H. Streitz, M. P. Surh, and J. A. Gunnels, “Beyond homogeneous decomposition: scaling long-range forces on massively parallel

systems,” in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. IEEE, 2009, pp. 1–12.

- [89] C. G. Lambert, T. A. Darden, and J. A. B. Jr., “A multipole-based algorithm for efficient calculation of forces and potentials in macroscopic periodic assemblies of particles,” *Journal of Computational Physics*, vol. 126, no. 2, pp. 274 – 285, 1996. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0021999196901370>
- [90] G. M. Amdahl, “Validity of the single processor approach to achieving large scale computing capabilities,” in *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*. ACM, 1967, pp. 483–485.
- [91] D. Sakakibara, A. Sasaki, T. Ikeya, J. Hamatsu, T. Hanashima, M. Mishima, M. Yoshimasu, N. Hayashi, T. Mikawa, M. Wälchli, B. Smith, M. Shirakawa, P. Güntert, and Y. Ito, “Protein structure determination in living cells by in-cell nmr spectroscopy,” *Nature*, vol. 458, no. 7234, pp. 102–105, 2009.
- [92] P. E. Stein, A. G. Leslie, J. T. Finch, and R. W. Carrell, “Crystal structure of uncleaved ovalbumin at 1.95 Å resolution,” *Journal of Molecular Biology*, vol. 221, no. 3, pp. 941–959, 1991.
- [93] D. A. Case, T. E. Cheatham, T. Darden, H. Gohlke, R. Luo, K. M. Merz, A. Onufriev, C. Simmerling, B. Wang, and R. J. Woods, “The Amber biomolecular simulation programs,” *Journal of Computational Chemistry*, vol. 26, no. 16, pp. 1668–1688, 2005.
- [94] V. Hornak, R. Abel, A. Okur, B. Strockbine, A. Roitberg, and C. Simmerling, “Comparison of multiple amber force fields and development of improved protein backbone parameters,” *Proteins: Structure, Function, and Bioinformatics*, vol. 65, no. 3, pp. 712–725, 2006.
- [95] W. L. Jorgensen, J. Chandrasekhar, J. D. Madura, R. W. Impey, and M. L. Klein, “Comparison of simple potential functions for simulating liquid water,” *The Journal of Chemical Physics*, vol. 79, no. 2, 1983.

- [96] S. Nos, “A unified formulation of the constant temperature molecular dynamics methods,” *The Journal of Chemical Physics*, vol. 81, no. 1, 1984.
- [97] W. G. Hoover, “Canonical dynamics: equilibrium phase-space distributions,” *Physical review A*, vol. 31, no. 3, p. 1695, 1985.
- [98] G. J. Martyna, M. E. Tuckerman, D. J. Tobias, and M. L. Klein, “Explicit reversible integrators for extended systems dynamics,” *Molecular Physics*, vol. 87, no. 5, pp. 1117–1157, 1996.
- [99] H. Ibeid, R. Yokota, J. Pestana, and D. E. Keyes, “Fast multipole preconditioners for sparse matrices arising from elliptic equations,” *arXiv:1308.3339v2*, 2014.
- [100] H. Ibeid, R. Yokota, and D. Keyes, “A matrix-free preconditioner for the Helmholtz equation based on the fast multipole method,” 2016, <http://arxiv.org/abs/1608.02461>.
- [101] S. R. Sambavaram, V. Sarin, A. Sameh, and A. Grama, “Multipole-based preconditioners for large sparse linear systems,” *Parallel Computing*, vol. 29, pp. 1261–1273, 2003.
- [102] M. Benzi, G. H. Golub, and J. Liesen, “Numerical solution of saddle point problems,” *Acta Numerica*, vol. 14, pp. 1–137, 2005.
- [103] A. Logg, K.-A. Mardal, and G. Wells, *Automated solution of differential equations by the finite element method: The FEniCS book*. Springer Science & Business Media, 2012, vol. 84.
- [104] H. A. van der Vorst, “Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems,” *SIAM J. Sci. Stat. Comput.*, vol. 13, no. 2, pp. 631–644, Mar. 1992. [Online]. Available: <http://dx.doi.org/10.1137/0913035>
- [105] P. Sonneveld, “CGS, a fast Lanczos-type solver for nonsymmetric linear systems,” *SIAM J. Sci. Stat. Comput.*, vol. 10, no. 1, pp. 36–52, Jan. 1989. [Online]. Available: <http://dx.doi.org/10.1137/0910004>

- [106] C. G. Broyden and M. T. Vespucci, *Krylov Solvers for Linear Algebraic Systems: Krylov Solvers*. Elsevier, 2004, vol. 11.
- [107] A. Greenbaum, *Iterative Methods for Solving Linear Systems*. USA: SIAM, 1997.
- [108] Y. Saad, *Iterative Methods for Sparse Linear Systems*. Philadelphia: SIAM, 2003.
- [109] M. Benzi, “Preconditioning techniques for large linear systems: A survey,” *Journal of Computational Physics*, vol. 182, no. 2, pp. 418–477, Nov. 2002. [Online]. Available: <http://dx.doi.org/10.1006/jcph.2002.7176>
- [110] J. Cahouet and J.-P. Chabard, “Some fast 3D finite element solvers for the generalized Stokes problem,” *International Journal for Numerical Methods in Fluids*, vol. 8, pp. 869–895, 1988.
- [111] A. J. Wathen, “Realistic eigenvalue bounds for the Galerkin mass matrix,” *IMA Journal of Numerical Analysis*, vol. 7, pp. 449–457, 1987.
- [112] A. Wathen and T. Rees, “Chebyshev semi-iteration in preconditioning for problems including the mass matrix,” *Electronic Transactions on Numerical Analysis*, vol. 34, pp. 125–135, 2009.
- [113] O. G. Ernst and M. J. Gander, “Why it is difficult to solve Helmholtz problems with classical iterative methods,” in *Numerical Analysis of Multiscale Problems*, ser. Lecture Notes in Computational Science and Engineering, I. G. Graham, T. Y. Hou, O. Lakkis, and R. Scheichl, Eds. Springer Berlin Heidelberg, 2012, vol. 83, pp. 325–363. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-22061-6_10
- [114] A. Brandt and I. Livshits, “Wave-ray multigrid method for standing wave equations.” *ETNA. Electronic Transactions on Numerical Analysis [electronic only]*, vol. 6, pp. 162–181, 1997. [Online]. Available: <http://eudml.org/doc/119506>

- [115] S. A. Sauter and Ch. Schwab, *Boundary Element Methods*. Heidelberg: Springer-Verlag, 2011.
- [116] S. J. Aarseth, “Dynamical evolution of clusters of galaxies, I,” *Monthly Notices of the Royal Astronomical Society*, vol. 126, p. 233, 1963.
- [117] I. Brunton, “Solving variable coefficient partial differential equations using the boundary element method,” Ph.D. dissertation, University of Auckland, 1996.
- [118] I. G. Graham, P. O. Lechner, and R. Scheichl, “Domain decomposition for multiscale PDEs,” *Numerische Mathematik*, vol. 106, pp. 589–626, 2007.
- [119] A. H.-D. Cheng, “Darcy’s flow with variable permeability: A boundary integral solution,” *Water Resources Research*, vol. 20, pp. 980–984, 1984.
- [120] D. L. Clements, “A boundary integral equation method for the numerical solution of a second order elliptic equation with variable coefficients,” *Journal of the Australian Mathematical Society*, vol. 22 (Series B), pp. 218–228, 1980.
- [121] U. Langer, G. Of, O. Steinbach, and W. Zulehner, “Inexact fast multipole boundary element tearing and interconnecting methods,” in *Domain Decomposition Methods in Science and Engineering XVI*. Springer Berlin Heidelberg, 2007.
- [122] L. J. Wardle and J. M. Crotty, “Two dimensional boundary integral equation analysis for non-homogeneous mining applications,” in *Recent Advances in Boundary Element Methods*. Pentch Press, 1978.
- [123] P. K. Banerjee, “Non-linear problems of potential flow,” in *Developments in Boundary Element Methods—I*. Applied Science, 1979.
- [124] P. Concus and G. H. Golub, “Use of fast direct methods for the efficient numerical solution of nonseparable elliptic equations,” *SIAM Journal on Numerical Analysis*, vol. 10, pp. 1103–1120, 1973.

- [125] H. Elman, A. Ramage, and D. Silvester, “Algorithm 866: IFISS, a Matlab toolbox for modelling incompressible flow,” *ACM Transactions on Mathematical Software*, vol. 33, pp. 2–14, 2007.
- [126] D. Silvester, H. Elman, and A. Ramage, “Incompressible Flow and Iterative Solver Software (IFISS) version 3.2,” May 2012, <http://www.manchester.ac.uk/ifiss>.
- [127] J. A. Meijerink and H. A. van der Vorst, “An iterative solution method for linear systems of which the coefficient matrix is a symmetric M -matrix,” *Mathematics of Computation*, vol. 31, pp. 148–162, 1977.
- [128] Y. Erlangga, C. Vuik, and C. Oosterlee, “On a class of preconditioners for solving the discrete Helmholtz equation,” in *Proceedings of The Sixth International Conference on Mathematical and Numerical Aspects of Wave Propagation Held at Jyväskylä, Finland, 30 June - 4 July, 2003*, G. Cohen, E. Heikkola, P. Joly, and P. Neittaanmäki, Eds. Berlin: Springer, 2003, pp. 788–793.
- [129] W. Hui and Q. Qinghua, “Some problems with the method of fundamental solution using radial basis functions,” *Acta Mechanica Solida Sinica*, vol. 20, no. 1, pp. 21 – 29, 2007. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0894916609601269>
- [130] S. Balay, J. Brown, , K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, “PETSc users manual,” Argonne National Laboratory, Tech. Rep. ANL-95/11 - Revision 3.4, 2013.
- [131] S. Balay, J. Brown, K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, “PETSc Web page,” 2013, <http://www.mcs.anl.gov/petsc>.
- [132] V. C. N. Collier, L. Dalcin, “PetIGA: High-performance isogeometric analysis,” *arxiv*, no. 1305.4452, 2013, <http://arxiv.org/abs/1305.4452>.

- [133] R. Yokota, H. Ibeid, and D. Keyes, “Fast multipole method as a matrix-free hierarchical low-rank approximation,” *CoRR*, vol. abs/1602.02244, 2016. [Online]. Available: <http://arxiv.org/abs/1602.02244>
- [134] W. Hackbusch, B. Khoromskij, and S. A. Sauter, “On H^2 -matrices,” in *Lectures on Applied Mathematics*, H. Bungartz, R. Hoppe, and C. Zenger, Eds. Springer-Verlag, 2000.
- [135] P. G. Martinsson and V. Rokhlin, “A fast direct solver for boundary integral equations in two dimensions,” *Journal of Computational Physics*, vol. 205, pp. 1–23, 2005.
- [136] S. Ambikasaran and E. Darve, “An $O(N \log N)$ fast direct solver for partial hierarchically semi-separable matrices,” *Journal of Scientific Computing*, vol. accepted, 2013.
- [137] —, “The inverse fast multipole method,” *arXiv:1407.1572v1*, 2014.
- [138] L. Ying, G. Biros, and D. Zorin, “A kernel-independent adaptive fast multipole algorithm in two and three dimensions,” *Journal of Computational Physics*, vol. 196, no. 2, pp. 591–626, 2004.
- [139] W. Fong and E. Darve, “The black-box fast multipole method,” *Journal of Computational Physics*, vol. 228, pp. 8712–8725, 2009.
- [140] L. A. Barba and R. Yokota, “How will the fast multipole method fare in the exascale era?” *SIAM News*, vol. 46, no. 6, pp. 1–3, 2013.
- [141] A. W. Appel, “An efficient program for many-body simulation,” *SIAM Journal on Scientific and Statistical Computing*, vol. 6, no. 1, pp. 85–103, 1985.
- [142] W. Hackbusch and Z. P. Nowak, “On the fast matrix multiplication in the boundary element method by panel clustering,” *Numerische Mathematik*, vol. 54, pp. 463–491, 1989.
- [143] J. Bédorf, E. Gaburov, M. S. Fujii, K. Nitadori, T. Ishiyama, and S. Portegies Zwart, “24.77 Pflops on a gravitational tree-code to simulate the Milky

Way galaxy with 18600 GPUs,” in *Proceedings of the 2014 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, 2014, pp. 1–12.

- [144] C. R. Anderson, “An implementation of the fast multipole method without multipoles,” *SIAM Journal on Scientific and Statistical Computing*, vol. 13, no. 4, pp. 923–947, 1992.
- [145] C. L. Berman, “Grid-multipole calculations,” *SIAM Journal on Scientific Computing*, vol. 16, no. 5, pp. 1082–1091, 1995.
- [146] J. Makino, “Yet another fast multipole method without multipoles – Pseudoparticle multipole method,” *Journal of Computational Physics*, vol. 151, no. 2, pp. 910–920, 1999.
- [147] P. Amestoy, C. Ashcraft, O. Boiteau, A. Buttari, J.-Y. L’Excellent, and C. Weisbecker, “Improving multifrontal methods by means of block low-rank representations,” *SIAM Journal on Scientific Computing*, vol. 37, no. 3, pp. A1451–A1474, 2015.
- [148] M. Bebendorf, *Hierarchical Matrices*, ser. Lecture Notes in Computational Science and Engineering. Springer, 2008, vol. 63.
- [149] S. Börm, “Construction of data-sparse H^2 -matrices by hierarchical compression,” *SIAM Journal on Scientific Computing*, vol. 31, no. 3, pp. 1820–1839, 2009.
- [150] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li, “Fast algorithms for hierarchically semiseparable matrices,” *Numerical Linear Algebra with Applications*, vol. 17, pp. 953–976, 2010.
- [151] F.-H. Rouet, X. S. Li, P. Ghysels, and A. Napov, “A distributed-memory package for dense hierarchically semi-separable matrix computations using randomization,” *ACM Transactions on Mathematical Software*, vol. 42, no. 4, pp. 27:1–27:35, Jun. 2016. [Online]. Available: <http://doi.acm.org/10.1145/2930660>

- [152] S. Le Borne, “Multilevel hierarchical matrices,” *SIAM Journal on Matrix Analysis and Applications*, vol. 28, no. 3, pp. 871–889, 2006.
- [153] S. Oliveira and Y. F., “An algebraic approach for H-matrix preconditioners,” *Computing*, vol. 80, pp. 169–188, 2007.
- [154] W. B. March, B. Xiao, and G. Biros, “ASKIT: Approximate skeletonization kernel-independent treecode in high dimensions,” *SIAM Journal on Scientific Computing*, vol. 37, no. 2, pp. A1089–A1110, 2015.
- [155] M. Izadi, “Hierarchical matrix techniques on massively parallel computers,” Ph.D. dissertation, Universität Leipzig, 2012.
- [156] K. Lessel, M. Hartman, and S. Chandrasekaran, “A fast memory efficient construction algorithm for hierarchically semi-separable representations,” *SIAM Journal on Matrix Analysis and Applications*, vol. 37, no. 1, pp. 338–353, 2016.
- [157] T. F. Chan, “Rank revealing QR factorizations,” *Linear Algebra and its Applications*, vol. 88/89, pp. 67–82, 1987.
- [158] Y. P. Hong and C. T. Pan, “Rank-revealing QR factorizations and the singular value decomposition,” *Mathematics of Computation*, vol. 58, no. 197, pp. 213–232, 1992.
- [159] S. Chandrasekaran and I. C. F. Ipsen, “On rank-revealing factorizations,” *SIAM Journal on Matrix Analysis and Applications*, vol. 15, no. 2, pp. 592–622, 1994.
- [160] T. F. Chan, “On the existence and computation of LU-factorizations with small pivots,” *Mathematics of Computation*, vol. 42, no. 166, pp. 535–547, 1984.
- [161] T.-M. Hwang, W.-W. Lin, and E. K. Yang, “Rank revealing LU factorizations,” *Linear Algebra and its Applications*, vol. 175, pp. 115–141, 1992.
- [162] T.-M. Hwang, W.-W. Lin, and D. Pierce, “Improved bound for rank revealing LU factorizations,” *Linear Algebra and its Applications*, vol. 261, no. 1, pp. 173–186, 1997.

- [163] L. Miranian and M. Gu, “Strong rank revealing LU factorizations,” *Linear Algebra and its Applications*, vol. 367, pp. 1–16, 2003.
- [164] R. Yokota, G. Turkiyyah, and D. Keyes, “Communication complexity of the fast multipole method and its algebraic variants,” *Supercomputing Frontiers and Innovations*, vol. 1, no. 1, pp. 63–84, 2014.
- [165] M. Hoemmen, “Communication-avoiding Krylov subspace methods,” Ph.D. dissertation, Berkeley, CA, USA, 2010, aAI3413388.

APPENDICES

Appendix A

Machines Description

In this thesis, we run our experiments on six principal machines described in this chapter: Titan, K computer, Mira, Shaheen II, Stampede, Piz Dora, and Shaheen I. A summary of the key features of each machine is provided in Table A.1.

A.1 Titan

A Cray XK7 system with 18,688 compute nodes each equipped with an AMD Opteron 6274 CPU and NVIDIA Kepler K20X GPU. The CPU has 16 cores running at 2.2GHz with 16 kB L1 cache, 2×4 MB L2 cache, and 8×2 MB L3 cache. The GPU has 15×64 cores running at 730MHz with $64 + 48$ kB L1 cache and 1.5 MB L2 cache. Each compute node has 32 GB of RAM with 51.2 memory bandwidth. The nodes are connected by a 3-D torus with 20 GB/s of injection bandwidth per node. We do not use any of the GPUs in the current study.

A.2 K computer

A Fujitsu system with 82,944 compute nodes each equipped with a SPARC64 VIIIfx CPU, an interconnect chip (ICC), and memory. A special rack houses 96 nodes, disk drives, and I/O units. The CPU has 8 cores running at 2GHz. Each compute node has 32 GB of RAM with 51.2 memory bandwidth. The nodes are connected by a

Torus FUsion (Tofu), a 6-D mesh/torus network.

A.3 Mira

An IBM BlueGene/Q system with 48 racks each contains 1024 Power A2 CPUs. The CPU has 16 + 1 cores running at 1.6GHz with 16 kB private L1 cache and 32 MB shared L2 cache. Each compute node has 16 GB RAM with 42.6 GB/s memory bandwidth. The nodes are connected by a 5-D torus network with 20 GB/s injection bandwidth per node.

A.4 Shaheen II

A Cray XC40 system with 6,174 compute nodes each equipped with two Intel Haswell CPUs (Intel®Xeon®E5-2698 v3). The CPU has 16 cores running at 2.3GHz. Each compute node has 128 GB of RAM running at 2,300 MHz. The nodes are connected by a dragonfly network using the Aries interconnect where the routers in each group are arranged as rows and columns of a rectangle, with all-to-all links across each row and column but not diagonally.

A.5 Stampede

A Dell Linux Cluster based on 6,400 Dell PowerEdge server nodes, each outfitted with two Intel Xeon E5 (Sandy Bridge) processors and an Intel Xeon Phi Coprocessor (MIC Architecture). The CPU has 8 cores running at 2.7GHz. The majority of the 6,400 nodes are configured with two Xeon E5-2680 processors and one Intel Xeon Phi SE10P Coprocessor (on a PCIe card). These compute nodes are configured with 32 GB of “host” memory with an additional 8 GB of memory on the Xeon Phi coprocessor card. A smaller number of compute nodes are configured with two Xeon Phi Coprocessors. The nodes are connected by an Infiniband FDR network.

A.6 Piz Dora

A Cray XC40 system with 1,256 compute nodes each equipped with two Intel Haswell CPUs (Intel®Xeon®E5-2690 v3). The CPU has 18 cores running at 2.1GHz. Out of the total, 1,192 nodes features 64 GB of RAM each, while the remaining 64 compute nodes have 128 GB of RAM each (fat nodes). The nodes are connected by a dragonfly network using the Aries interconnect.

A.7 Shaheen I

An IBM BlueGene/P system with 16 racks each contains 1024 PowerPC 450 CPUs. The CPU has 4 cores running at 850MHz with 32 kB private L1 cache and 8 MB shared L3 cache. Each compute node has 2 GB RAM with 13.6 GB/s memory bandwidth. The nodes are connected by 3-D torus network with 5.1 GB/s injection bandwidth per node.

Table A.1: Machines Description (as of November, 2016[1])

	Active	Manufacturer	Peak PFlop/s	Ranking ¹	Topology	Processor
Titan	2012	Cray Inc.	17.59	3	3D Torus	Opteron 6274 16C
K computer	2011	Fujitsu	10.51	7	Tofu interconnect	SPARC64 VIIIfx 8C
Mira	2012	IBM	8.59	9	5D Torus	Power BQC 16C
Shaheen II	2016	Cray Inc.	5.54	15	Aries interconnect	Xeon E5-2698v3 16C
Stampede	2012	Dell	5.17	17	Infiniband FDR	Xeon E5-2680 8C
Piz Dora	2014	Cray Inc.	1.41	71	Aries interconnect	Xeon E5-2695v4 18C

¹ TOP500: November, 2016[1]

Appendix B

Mathematical Supplements

Let Ω be an arbitrary domain that is bounded by a piecewise closed boundary Γ , and \mathbf{n} be the unit vector that is normal to Γ pointing outward.

B.1 Divergence Theorem

The Divergence theorem, also known as Gauss's theorem, states that the domain integral of the divergence of any differentiable function f inside Ω is equal to the flow rate of f across Γ

$$\int_{\Omega} \nabla \cdot f d\Omega = \int_{\Gamma} f \cdot \mathbf{n} d\Gamma. \quad (\text{B.1})$$

B.2 Dirac's Delta Function (δ)

By construction, Dirac's delta function is endowed with the following properties:

1. The Dirac's delta function vanishes everywhere except at the point \mathbf{x}_0 , where it becomes infinite

$$\delta(\mathbf{x} - \mathbf{x}_0) = \begin{cases} 0, & \mathbf{x} \neq \mathbf{x}_0, \\ \infty, & \mathbf{x} = \mathbf{x}_0. \end{cases}$$

2. The integral of the delta function over the domain Ω that contains the singular

point \mathbf{x}_0 is equal unity

$$\int_{\Omega} \delta(\mathbf{x} - \mathbf{x}_0) d\Omega = 1. \quad (\text{B.2})$$

3. The integral of the product of an arbitrary function $f(\mathbf{x})$ and the delta function over a domain Ω that contains the point \mathbf{x}_0 is equal to value of the function at the singular point

$$\int_{\Omega} \delta(\mathbf{x} - \mathbf{x}_0) f(\mathbf{x}) d\Omega = f(\mathbf{x}_0). \quad (\text{B.3})$$

B.3 Fundamental Solutions

In this section, we derive the fundamental solutions of the 2-D and 3-D Laplace and Helmholtz equations. Note that the Green functions G of these equations in the free space are equivalent to the fundamental solutions.

B.3.1 Laplace Equation

By definition, the Green's function, denoted by $G(\mathbf{x}, \mathbf{x}_0)$, satisfies the singularly forced Laplace equation

$$\nabla^2 G(\mathbf{x}, \mathbf{x}_0) = \delta(\mathbf{x} - \mathbf{x}_0). \quad (\text{B.4})$$

B.3.1.1 Fundamental solution in 2-D

We first look for the function $f(x, y)$ in the whole space \mathbb{R}^2 that satisfies (B.4). Using polar coordinates r and θ centered around $(0, 0)$, where $x = r \cos \theta$ and $y = r \sin \theta$, (B.4) can be written as

$$\frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\partial \Psi}{\partial r} \right) + \frac{1}{r^2} \frac{\partial^2 \Psi}{\partial \theta^2} = \delta(r), \quad (\text{B.5})$$

where $\Psi(r, \theta) = f(r \cos \theta, r \sin \theta)$.

To find the 2-D Laplace fundamental solution we first find the simplest function

that satisfies (B.4) in 2-D. We assume that Ψ is independent of θ which reduces (B.5) to the ordinary differential equation

$$\frac{d}{dr} \left(r \frac{d\Psi}{dr} \right) = 0, \quad r \neq 0. \quad (\text{B.6})$$

The above equation can be written as

$$\Psi'' + \frac{1}{r} \Psi' = 0. \quad (\text{B.7})$$

We integrate (B.7) as follow

$$\begin{aligned} \frac{\Psi''}{\Psi'} &= \frac{-1}{r}, \\ (\ln |\Psi'|)' &= \frac{-1}{r}, \\ \ln |\Psi'| &= -\ln r + a, \\ |\Psi'| &= e^{-\ln r} e^a, \\ \Psi' &= \frac{A}{r}. \end{aligned}$$

Which yields the general solution

$$\Psi(r) = A \ln(r) + B, \quad (\text{B.8})$$

for some constants a , A and B .

We can take $B = 0$ because constants will be differentiated away and shift the center of the polar coordinates from $(0, 0)$ to (x_0, y_0) as follows

$$f(x, y) = A \ln \sqrt{(x - x_0)^2 + (y - y_0)^2}. \quad (\text{B.9})$$

To find A , we integrate $f(x, y)$ over a disc of radius ϵ centered at $(0, 0)$

$$\int_{\Omega} \nabla^2 f(x, y) d\Omega = \int_{\Omega} \delta(x, y) d\Omega = 1. \quad (\text{B.10})$$

Using the Divergence Theorem (B.1)

$$\int_{\Gamma} \nabla f(x, y) \cdot \mathbf{n} d\Gamma = 1, \quad (\text{B.11})$$

where Γ is a circle of circumference $2\pi\epsilon$. Combining the previous equations gives

$$\int_{\Gamma} \nabla f(x, y) \cdot \mathbf{n} d\Gamma = 1 = \int_{\Gamma} \frac{A}{\epsilon} d\Gamma = 2\pi A. \quad (\text{B.12})$$

Therefore,

$$f(x, y) = \frac{1}{2\pi} \ln(r), \quad (\text{B.13})$$

where $r = \sqrt{(x - x_0)^2 + (y - y_0)^2}$.

We refer to the function $f(x, y)$ as the fundamental solution of the Laplacian in \mathbb{R}^2 . It is clear that for any point $\mathbf{x}_0 \in \mathbb{R}^2$, $\nabla^2 f(\mathbf{x}, \mathbf{x}_0) = \delta(\mathbf{x} - \mathbf{x}_0)$.

B.3.1.2 Fundamental solution in 3-D

Following the analysis in the previous section, we define a function $f(x, y, z)$ that satisfies (B.4) in \mathbb{R}^3 . Using the spherical coordinates r , θ , and Ψ centered around $(0, 0, 0)$, we define $\Psi(r, \theta, \Psi) = f(r \sin \Psi \cos \theta, r \sin \Psi \sin \theta, r \cos \Psi)$, where $r = \sqrt{x^2 + y^2 + z^2}$.

The 3-D Laplacian in polar coordinates can be written as

$$\frac{1}{r^2} \frac{\partial}{\partial r} \left(r^2 \frac{\partial \Psi}{\partial r} \right) + \frac{1}{r^2 \sin^2 \Psi} \frac{\partial^2 \Psi}{\partial \theta^2} + \frac{1}{r^2 \sin \Psi} \frac{\partial}{\partial \Psi} \left(\sin \Psi \frac{\partial \Psi}{\partial \Psi} \right) = \delta(r). \quad (\text{B.14})$$

Solving for the simplest function that satisfies (B.4) in 3-D given that Ψ depends on

the coordinate r only reduces (B.14) to the ordinary differential equation

$$\frac{d}{dr}\left(r^2 \frac{\partial \Psi}{\partial r}\right) = 0. \quad (\text{B.15})$$

Which can be rewritten as

$$\Psi'' + \frac{2}{r}\Psi' = 0. \quad (\text{B.16})$$

We integrate (B.16) twice as follow

$$\frac{\Psi''}{\Psi'} = \frac{-2}{r},$$

$$(\ln |\Psi'|)' = \frac{-2}{r},$$

$$\ln |\Psi'| = -2 \ln r + a,$$

$$|\Psi'| = e^{-2 \ln r} e^a,$$

$$\Psi' = \frac{b}{r^2}.$$

Which yields

$$\Psi(r) = \frac{A}{r} + B, \quad (\text{B.17})$$

for some constants a , b , A and B .

To find the constants, we set $B = 0$ because constants will be differentiated away and integrate $f(x, y, z)$ over a sphere of radius ϵ centered at the origin using (B.1), which gives

$$\int_{\Gamma} \nabla f(x, y) \cdot \mathbf{n} d\Gamma = 1 = \int_{\Gamma} \frac{A}{\epsilon^2} d\Gamma = -4\pi A. \quad (\text{B.18})$$

Therefore, the fundamental solution of the \mathbb{R}^3 Laplacian can be written as

$$f(x, y, z) = \frac{-1}{4\pi r}. \quad (\text{B.19})$$

B.3.2 Helmholtz Equation

The fundamental solution of the Helmholtz equation $G(\mathbf{x}, \mathbf{x}_0)$ is defined as

$$\nabla^2 G(\mathbf{x}, \mathbf{x}_0) + k^2 G(\mathbf{x}, \mathbf{x}_0) = \delta(\mathbf{x} - \mathbf{x}_0). \quad (\text{B.20})$$

B.3.2.1 Fundamental solution in 2-D

We first look for the function $f(x, y)$ that satisfies (B.20) in the whole space \mathbb{R}^2 . By using the Laplacian in the polar coordinates (B.5), (B.20) can be written as

$$\frac{1}{r} \frac{d}{dr} \left(r \frac{d\Psi}{dr} \right) + k^2 \Psi = \delta(r). \quad (\text{B.21})$$

Equation (B.21) can be rewritten as

$$\frac{1}{r} \frac{d}{dr} \left(r \frac{d\Psi}{dr} \right) + k^2 \Psi = 0. \quad (\text{B.22})$$

The above equation is known as the Bessel's differential equation. Equation (B.22) admits the solution

$$\Psi(r) = Y_0(kr) + iJ_0(kr), \quad (\text{B.23})$$

where J_0 and Y_0 are the zeroth order Bessel functions of the first and second kinds, respectively. Equation B.23 is known as the Hankel function of the first kind $H_0^1(kr)$.

The Bessel functions J_0 and Y_0 are given by

$$J_0(x) = \sum_{m=0}^{\infty} \frac{(-1)^m x^{2m}}{4^m (m!)^m}, \quad (\text{B.24})$$

and

$$Y_0(x) = \frac{2}{\pi} \left(\ln\left(\frac{x}{2}\right) + \gamma \right) J_0(x) - \frac{2}{\pi} \sum_{m=0}^{\infty} \frac{(-1)^m x^{2m}}{4^m (m!)^2}, \quad (\text{B.25})$$

where γ is the Euler constant defined by

$$\gamma = \lim_{n \rightarrow \infty} \left(-\ln(n) + 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \right). \quad (\text{B.26})$$

Therefore, the fundamental solution of the 2-D Helmholtz equation can be written as

$$\Psi(r) = AH_0^1(kr). \quad (\text{B.27})$$

The constant A is determined by substituting (B.27) into (B.20), and then integrating (B.20) over a disc of radius ϵ centered at $(0, 0)$, as

$$\int_{\Gamma} \nabla f(x, y) d\Gamma + \int_{\Omega} k^2 f(x, y) d\Omega = 1. \quad (\text{B.28})$$

By taking the limit of $r \rightarrow 0$, we obtain $A = i/4$. Therefore, the fundamental solution of the 2-D Helmholtz equation can be written as

$$f(x, y) = \frac{i}{4} H_0^1(kr). \quad (\text{B.29})$$

B.3.2.2 Fundamental solution in 3-D

We define a function $f(x, y, z)$ that satisfies (B.20) in \mathbb{R}^3 . By using the 3-D Laplacian in the polar coordinates (B.14), (B.20) can be written as

$$\frac{1}{r^2} \frac{d}{dr} \left(r^2 \frac{d\Psi}{dr} \right) + k^2 \Psi = \delta(r). \quad (\text{B.30})$$

By multiplying both sides by r , (B.30) can be rewritten as

$$\frac{d^2}{dr^2} (r\Psi) + k^2 (r\Psi) = 0. \quad (\text{B.31})$$

Equation B.31 is solved to obtain

$$\Psi(r) = A \frac{e^{-ikr}}{r}. \quad (\text{B.32})$$

The arbitrary constant A is determined by substituting (B.32) into (B.20), and then integrating (B.20) within the small circle including the origin, as

$$\begin{aligned} \int_{\Omega} \nabla^2 f(x, y, z) d\Omega &= \int_{\Gamma} \nabla f(x, y, z) d\Gamma \\ &= 4\pi r^2 \hat{r} \cdot \nabla f(x, y, z) \\ &= 4\pi A r^2 \left(\frac{e^{-ikr}}{r^2} + ik \frac{e^{-ikr}}{r} \right) \end{aligned}$$

$$\int_{\Omega} k^2 f(x, y, z) d\Omega = 4\pi k^2 A \left(-\frac{1}{ik} r e^{-ikr} + \frac{1}{k^2} (e^{-ikr} - 1) \right)$$

$$\int_{\Omega} \delta d\Omega = 1.$$

By taking the limit of $r \rightarrow 0$, we get $A = 1/4\pi$. Therefore, the fundamental solution of the 3-D Helmholtz equation can be written as

$$f(x, y, z) = \frac{1}{4\pi r} e^{ikr}. \quad (\text{B.33})$$

Appendix C

Boundary Element Method

C.1 Weighted Residual Methods

Let us consider the Poisson equation of the form

$$\nabla^2 \tilde{u} = f. \quad (\text{C.1})$$

The exact solution \tilde{u} can be discretized by

$$\tilde{u} = \sum_i u_i \phi_i, \quad (\text{C.2})$$

where u_i and ϕ_i are the nodal values and basis functions at node i , respectively. Similarly, the spatial coordinates \mathbf{x} can be expressed by a superposition of shape functions ψ

$$\tilde{\mathbf{x}} = \sum_i \mathbf{x}_i \psi_i. \quad (\text{C.3})$$

With this in mind, a broad range of methods can be formulated as weighted residual methods

$$\int_{\Omega} \underbrace{(\nabla^2 u - f)}_R W d\Omega = 0, \quad (\text{C.4})$$

where R and W are the residual and weight, respectively. The integral is over the domain Ω , which represents a volume in 3-D and surface in 2-D problems. Depending

on what is used for the weighting function W , this can turn into many different types of numerical methods as shown in Table C.1.

The weighting function for least squares method could be understood better by substituting it into (C.4) to obtain

$$\frac{\partial}{\partial u_i} \int_{\Omega} R^2 d\Omega = 0, \quad (\text{C.5})$$

which now looks like a minimization problem for the squared residual.

C.2 Weak Form

Equation (C.4) can be transformed to yield powerful numerical methods. We will use two mathematical concepts in the following analysis. The Gauss divergence theorem (B.1) and integration by parts

$$\int f'(x)g(x) = \int (f(x)g(x))' - \int f(x)g'(x). \quad (\text{C.6})$$

Moving f in (C.4) to the right hand side gives

$$\int_{\Omega} (\nabla^2 u)W d\Omega = \int_{\Omega} fW d\Omega. \quad (\text{C.7})$$

Table C.1: Weighted Residual Methods

$W_i = \delta(\mathbf{x} - \mathbf{x}_i)$	Collocation Method
$W_i = \frac{\partial R}{\partial u_i}$	Least Squares Method
$W_i = \mathbf{x}^i$	Method of Moments
$W_i = \phi_i$	Galerkin Method
$W_i = G(\mathbf{x} - \mathbf{x}_i)$	Green's Function Solution

Then, by separating the divergence operator on the left-hand side we obtain

$$\int_{\Omega} \nabla \cdot (\nabla u) W d\Omega = \int_{\Omega} f W d\Omega. \quad (\text{C.8})$$

Integrating by parts, by treating ∇u as $f(x)$ and W as $g(x)$ in (C.6), changes the left-hand side of (C.8) to

$$\int_{\Omega} \nabla \cdot (\nabla u W) d\Omega - \int_{\Omega} (\nabla u) \cdot \nabla W d\Omega = \int_{\Omega} f W d\Omega. \quad (\text{C.9})$$

Then, we apply Gauss's theorem (B.1) to the first term on the left-hand side

$$\int_{\Gamma} \mathbf{n} \cdot (\nabla u W) d\Gamma - \int_{\Omega} (\nabla u) \cdot \nabla W d\Omega = \int_{\Omega} f W d\Omega. \quad (\text{C.10})$$

Since $\mathbf{n} \cdot \nabla = \partial/\partial n$, we have

$$\int_{\Gamma} \frac{\partial u}{\partial n} W d\Gamma - \int_{\Omega} (\nabla u) \cdot \nabla W d\Omega = \int_{\Omega} f W d\Omega. \quad (\text{C.11})$$

This is the weak form of (C.4). When basis functions are used as the weights $W = \phi$ in the weak form, this becomes the finite element method.

C.3 Inverse Form

Applying integration by parts to the second term in (C.11), treating u as $f(x)$ and ∇W as $g(x)$ in (C.6), gives

$$\int_{\Gamma} \frac{\partial u}{\partial n} W d\Gamma - \int_{\Omega} \nabla \cdot (u \nabla W) d\Omega + \int_{\Omega} u (\nabla^2 W) d\Omega = \int_{\Omega} f W d\Omega. \quad (\text{C.12})$$

Using the divergence theorem on the second term gives

$$\int_{\Gamma} \frac{\partial u}{\partial n} W d\Gamma - \int_{\Gamma} u \frac{\partial W}{\partial n} d\Gamma + \int_{\Omega} u (\nabla^2 W) d\Omega = \int_{\Omega} f W d\Omega. \quad (\text{C.13})$$

When the Green's function is used as weights $W = G$, this becomes the boundary element method

$$\int_{\Gamma} \frac{\partial u}{\partial n} G d\Gamma - \int_{\Gamma} u \frac{\partial G}{\partial n} d\Gamma + \int_{\Omega} u (\nabla^2 G) d\Omega = \int_{\Omega} f G d\Omega. \quad (\text{C.14})$$

The Green's function has the property $\nabla^2 G = \delta$, so the third term becomes

$$\int_{\Omega} u (\nabla^2 G) d\Omega = \int_{\Omega} u \delta d\Omega. \quad (\text{C.15})$$

Inside the domain $\int_{\Omega} u \delta d\Omega = u$, and at the boundary $\int_{\Gamma} u \delta d\Gamma = 1/2u$. Thus, on the boundary we have

$$\int_{\Gamma} \frac{\partial u}{\partial n} G d\Gamma - \int_{\Gamma} u \left(\frac{\partial G}{\partial n} - \frac{1}{2} \delta \right) d\Gamma = \int_{\Omega} f G d\Omega, \quad (\text{C.16})$$

and inside the domain we have

$$u = \int_{\Gamma} u \frac{\partial G}{\partial n} d\Gamma - \int_{\Gamma} \frac{\partial u}{\partial n} G d\Gamma + \int_{\Omega} f G d\Omega. \quad (\text{C.17})$$

C.4 BEM Matrices

Solving (C.16) and (C.17) allows one to solve the Poisson equation (C.1) for Dirichlet, Neumann, and mixed boundary conditions. For Dirichlet boundary conditions, u is known on the boundary and the second term in (C.16) moves to the right-hand side,

and can be written in the following matrix form

$$N_{\Gamma} \left\{ \begin{array}{c} \overbrace{\left[\begin{array}{ccc} \ddots & & \\ & G(r_{ij}) & \\ & & \ddots \end{array} \right]}^{N_{\Gamma}} \underbrace{\left[\begin{array}{c} \vdots \\ \frac{\partial u_j}{\partial n} \\ \vdots \end{array} \right]}_{\text{unknown}} \end{array} \right. = \begin{array}{c} \overbrace{\left[\begin{array}{ccc} \ddots & & \\ & \frac{\partial G(r_{ij})}{\partial n} - \frac{1}{2} \delta_{ij} & \\ & & \ddots \end{array} \right]}^{N_{\Gamma}} \left[\begin{array}{c} \vdots \\ u_j \\ \vdots \end{array} \right] + \overbrace{\left[\begin{array}{ccc} \ddots & & \\ & G(r_{ij}) & \\ & & \ddots \end{array} \right]}^{N_{\Omega}} \left[\begin{array}{c} \vdots \\ f_j \\ \vdots \end{array} \right], \end{array}$$

where N_{Γ} and N_{Ω} are the number of boundary nodes and internal nodes, respectively. The G matrix on the left-hand side and the $(\partial G/\partial n - \delta/2)$ matrix on the right-hand side are $N_{\Gamma} \times N_{\Gamma}$ matrices, while the G matrix on the right hand side is a $N_{\Gamma} \times N_{\Omega}$ matrix. The vector $\partial u/\partial n$ on the left-hand side is the only unknown in this equation, and can be obtained by solving the linear system. For Neumann boundary conditions, $\partial u/\partial n$ is known and the first term in (C.16) moves to the right-hand side

$$N_{\Gamma} \left\{ \begin{array}{c} \overbrace{\left[\begin{array}{ccc} \ddots & & \\ & \frac{\partial G(r_{ij})}{\partial n} - \frac{1}{2} \delta_{ij} & \\ & & \ddots \end{array} \right]}^{N_{\Gamma}} \underbrace{\left[\begin{array}{c} \vdots \\ u_j \\ \vdots \end{array} \right]}_{\text{unknown}} \end{array} \right. = \begin{array}{c} \overbrace{\left[\begin{array}{ccc} \ddots & & \\ & G(r_{ij}) & \\ & & \ddots \end{array} \right]}^{N_{\Gamma}} \left[\begin{array}{c} \vdots \\ \frac{\partial u_j}{\partial n} \\ \vdots \end{array} \right] - \overbrace{\left[\begin{array}{ccc} \ddots & & \\ & G(r_{ij}) & \\ & & \ddots \end{array} \right]}^{N_{\Omega}} \left[\begin{array}{c} \vdots \\ f_j \\ \vdots \end{array} \right]. \end{array}$$

For this case, one solves the linear system for vector u . For mixed boundary conditions, we move the unknown part of vectors u and $\partial u/\partial n$ to the left-hand side, and the known part to the right-hand side. The corresponding matrix entries in G and $(\partial G/\partial n - \delta/2)$ must also be moved. This results in the following system of equations

$$N_{\Gamma} \left\{ \begin{array}{c} \overbrace{\left[\begin{array}{cc|c} \ddots & & \\ G(r_{ij}) & \frac{1}{2} \delta_{ij} - \frac{\partial G(r_{ij})}{\partial n} & \\ & & \ddots \end{array} \right]}^{N_{\Gamma}} \underbrace{\left[\begin{array}{c} \vdots \\ \frac{\partial u_j}{\partial n} \\ u_j \\ \vdots \end{array} \right]}_{\text{unknown}} \end{array} \right. = \begin{array}{c} \overbrace{\left[\begin{array}{cc|c} \ddots & & \\ \frac{\partial G(r_{ij})}{\partial n} - \frac{1}{2} \delta_{ij} & -G(r_{ij}) & \\ & & \ddots \end{array} \right]}^{N_{\Gamma}} \left[\begin{array}{c} \vdots \\ \frac{u_j}{\partial n} \\ \vdots \end{array} \right] + \overbrace{\left[\begin{array}{ccc} \ddots & & \\ & G(r_{ij}) & \\ & & \ddots \end{array} \right]}^{N_{\Omega}} \left[\begin{array}{c} \vdots \\ f_j \\ \vdots \end{array} \right]. \end{array}$$

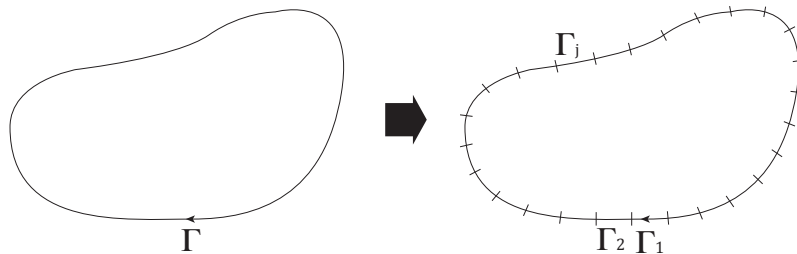


Figure C.1: Discretization of global integral into a sum of piecewise local integrals

The size of the vectors and matrices remain the same as the previous cases, but the entries are mixed according to the knowns and unknowns. In any case, the values of u and $\partial u/\partial n$ will be known everywhere on the boundary after solving the linear system. With this information it becomes possible to solve (C.17), which can also be expressed in the following matrix form

$$N_{\Omega} \begin{Bmatrix} \left[\begin{array}{c} \vdots \\ u_i \\ \vdots \end{array} \right] \\ \end{Bmatrix} = \overbrace{\begin{bmatrix} \ddots & & \\ & \frac{\partial G(r_{ij})}{\partial n} & \\ & & \ddots \end{bmatrix}}^{N_{\Gamma}} \begin{Bmatrix} \left[\begin{array}{c} \vdots \\ u_j \\ \vdots \end{array} \right] \\ \end{Bmatrix} - \overbrace{\begin{bmatrix} \ddots & & \\ & G(r_{ij}) & \\ & & \ddots \end{bmatrix}}^{N_{\Gamma}} \begin{Bmatrix} \left[\begin{array}{c} \vdots \\ \frac{\partial u_j}{\partial n} \\ \vdots \end{array} \right] \\ \end{Bmatrix} + \overbrace{\begin{bmatrix} \ddots & & \\ & G(r_{ij}) & \\ & & \ddots \end{bmatrix}}^{N_{\Omega}} \begin{Bmatrix} \left[\begin{array}{c} \vdots \\ f_j \\ \vdots \end{array} \right] \\ \end{Bmatrix}.$$

On the left-hand side, we have the values u at the internal points in the domain Ω , which is precisely the solution to the Poisson equation. The first two matrices on the right hand side have size $N_{\Omega} \times N_{\Gamma}$, while the last matrix is a $N_{\Omega} \times N_{\Omega}$ matrix. A classical N -body problem simply solves for this last term on the right hand side, and is a special case of this more general framework that can be extended to Dirichlet/Neumann boundary conditions for arbitrary geometries.

An important aspect when constructing these BEM matrices is how the continuous integrals in (C.16) and (C.17) are transformed into the discrete sums to form the linear systems shown above. This process will be explained in detail in the following section.

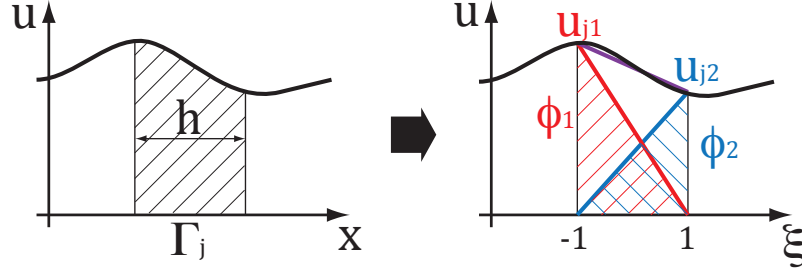


Figure C.2: Local integral by superposition of basis functions

C.5 Discretization

Discretization of all domain/boundary integrals in (C.16) and (C.17) is done in three steps:

1. Break the global integral into a sum of piecewise local integrals over each element as shown in Figure C.1. For example, the integration of the second term on the right-hand side of (C.17) can be expressed as

$$\int_{\Gamma} u \frac{\partial G}{\partial n} d\Gamma = \sum_j \int_{\Gamma_j} u \frac{\partial G}{\partial n} d\Gamma_j. \quad (\text{C.18})$$

The piecewise integration is still performed analytically.

2. Break the local integrals into the sum of contributions from the basis functions of each node that belongs to the element. Figure C.2 shows an example for the case of a linear and continuous basis function, where integration of a piecewise element Γ_i can be obtained from

$$\begin{aligned} \int_{\Gamma_j} u \frac{\partial G}{\partial n} d\Gamma_j &= \int_{\Gamma_j} u_{j1} \phi_1(x) \frac{\partial G}{\partial n} d\Gamma_j + \int_{\Gamma_j} u_{j2} \phi_2(x) \frac{\partial G}{\partial n} d\Gamma_j \\ &= \int_{-1}^1 u_{j1} \phi_1(\xi) \frac{\partial G}{\partial n} |J_j| d\xi + \int_{-1}^1 u_{j2} \phi_2(\xi) \frac{\partial G}{\partial n} |J_j| d\xi, \end{aligned} \quad (\text{C.19})$$

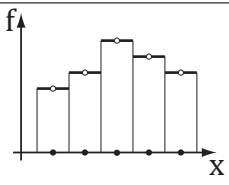
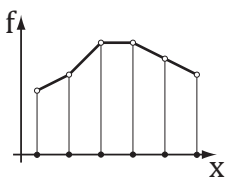
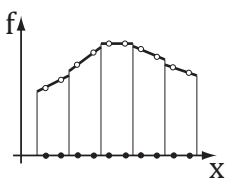
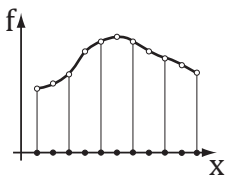
where $\phi_1(\xi) = (1 - \xi)/2$, $\phi_2(\xi) = (1 + \xi)/2$, and $J_j = \partial \mathbf{x}_j / \partial \xi$ is the Jacobian of the j th element. For this simple 1-D example $|J_j| = h/2$, where h is the length

of the element. It can be thought of as a scaling factor for the parameterized integration. Examples of other basis functions are shown in Table C.2. The figures on the left show the numerical values of u , which the basis functions reproduce. The vertical lines represent the boundaries of each element, and the black dots represent the location of the nodes, while the white dots represent the value of u at each of these nodes. The first row of the table is for constant (0^{th} order) elements. Constant elements are always discontinuous at the element boundaries. The second row is for linear continuous elements, which was shown in the example in Figure C.2. One can also construct linear discontinuous elements by using the function in the third row of Table C.2. Finally, the bottom row depicts a quadratic continuous element, where each element has three nodes. Continuous elements have nodes at the edges of the element, while discontinuous elements have all the nodes away from the edges of the element. To account for these various basis functions, (C.19) can be generalized to

$$\int_{\Gamma_j} u \frac{\partial G}{\partial n} d\Gamma_j = |J_j| \sum_k u_{jk} \int_{-1}^1 \phi_k(\xi) \frac{\partial G}{\partial n} d\xi, \quad (\text{C.20})$$

where the index k sums over all nodes in the element. Since $|J_j|$ is constant within each element, it can be moved outside of the summation and integration. Also, u_{jk} is not a function of ξ so it can be moved outside of the integration.

Table C.2: Different basis functions

	Order	Continuous	Function
	0	no	$\phi_1 = 1$
	1	yes	$\phi_1 = (1 - \xi)/2$ $\phi_2 = (1 + \xi)/2$
	1	no	$\phi_1 = 1/2 - \xi$ $\phi_2 = 1/2 + \xi$
	2	yes	$\phi_1 = \xi(\xi - 1)/2$ $\phi_2 = \xi(\xi + 1)/2$ $\phi_3 = (1 - \xi)(1 + \xi)$

3. Integrate over each basis function using Gaussian quadrature. Equation (C.20) is not completely discretized since it still requires analytical integration over the parametrized space ξ . In special cases where the basis function ϕ is of low order, and the Green's function G is simple (such as 2-D Laplace), the integration can be performed analytically. However, a more general solution to this problem is provided through numerical integration using Gaussian quadrature

$$\int_{-1}^1 \phi_k(\xi) \frac{\partial G}{\partial n} d\xi = \sum_l \phi_k(\xi_l) \frac{\partial G_{jkl}}{\partial n} w_l, \quad (\text{C.21})$$

where ξ_l and w_l are the parametrized coordinates and weights of the quadratures, respectively. The Green's function is a function of the location of the quadrature points, which depends on j , k , and l , and is therefore noted as G_{jkl} . By combining (C.18), (C.20), and (C.21) we have

$$\int_{\Gamma} u \frac{\partial G}{\partial n} d\Gamma = \sum_j |J_j| \sum_k u_{jk} \sum_l \phi_k(\xi_l) \frac{\partial G_{jkl}}{\partial n} w_l, \quad (\text{C.22})$$

where the indices j , k , and l correspond to the elements, nodes, and quadrature points, respectively.

C.6 Matrix Construction

The discretization shown in Section C.5 takes an element centric approach, where we first loop over the elements and then loop over the nodes that belong to this element. From the perspective of discretization, this is natural since the basis functions and quadratures are defined per element. Conversely, the matrices in Section C.4 take a node centric approach, where each matrix entry represents the contribution from each node. This is also quite natural since the data (x, y, z, u) are stored on the nodes and not the elements, and the matrix operates on this data directly. For

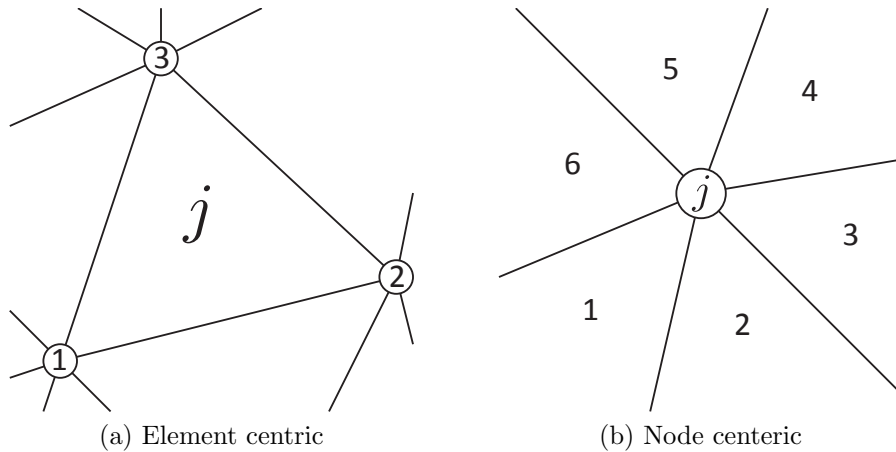


Figure C.3: Element centric and node centric approaches for BEM discretization

constant basis functions or discontinuous elements (see Table C.2) nodes are not shared between elements, so being element centric or node centric does not make a big difference. One can simply loop over the elements and then the nodes that belong to the element, and this will always be equivalent to looping over all of the nodes directly. However, for continuous elements, nodes are shared between different elements and the relation between looping over elements and looping over nodes becomes slightly more complicated.

Let us consider the case of 2-D triangular linear elements as shown in Figure C.3. The element centric approach for (C.22) will loop over the elements j and then over the three nodes k and then the quadrature points l . In order to retrieve u_{jk} , one must map the k^{th} node in the j^{th} element to the actual index of the node in vector \mathbf{u} . Although such mapping is easy to do, the element centric approach results in irregular data access patterns, since the shared nodes will be visited multiple times by different elements. The fact that the data are stored on the nodes is unchangeable, and a logical solution to this problem is to bring the compute to the data. This can be achieved by taking the node centric approach shown in Figure C.3. In this approach, one first loops over the nodes, then loops over the elements that contain this node and calculate the contribution from the corresponding basis function. Finally, the

innermost loop goes over the quadrature points in a similar fashion to the element centric case. We may redefine the indices in (C.22) accordingly to obtain

$$\int_{\Gamma} u \frac{\partial G}{\partial n} d\Gamma = \sum_j u_j \sum_k |J_{jk}| \sum_l \phi_{jk}(\xi_l) \frac{\partial G_{jkl}}{\partial n} w_l, \quad (\text{C.23})$$

where the indices j , k , and l correspond to the nodes, elements, and quadrature points, respectively. In this case, one must map the k^{th} element of the j^{th} node to the actual index of the element to obtain J_{jk} . Furthermore, it is necessary to keep track of whether the node is the first, second, or third node in that element, in order to obtain the correct basis function ϕ_{jk} . With the node centric approach one can easily construct the matrix from

$$\int_{\Gamma} u \frac{\partial G}{\partial n} d\Gamma = N_{\Gamma} \overbrace{\left[\begin{array}{c} \ddots \\ \sum_k |J_{jk}| \sum_l \phi_{jk}(\xi_l) \frac{\partial G_{jkl}}{\partial n} w_l \\ \ddots \end{array} \right]}^{N_{\Gamma}} \begin{bmatrix} \vdots \\ u_j \\ \vdots \end{bmatrix}. \quad (\text{C.24})$$

Note that this matrix is square ($N_{\Gamma} \times N_{\Gamma}$) and defines the relation between source nodes to target nodes. The complication of having elements and quadratures has been absorbed into the matrix elements themselves. This becomes a classical N -body problem where the point values at the nodes (x, y, z, u) can be used directly. There is no need to construct special data types for elements or quadratures to be fed in to the N -body solver. In the case of hierarchical N -body solvers, one simply needs to modify the $P2P$ and $P2M$ kernels to take into account the two loops over the elements and quadratures. The same logic can be applied to all other terms in (C.16) and (C.17).

Appendix D

Papers Accepted, Submitted, and Under Preparation

- M. AbdulJabbar, **H. Ibeid**, R. Yokota, and D. Keys. Simultaneously minimizing and Balancing Communication in Distributed Hierarchical N-body Methods. In preparation.
- **H. Ibeid**, R. Yokota, and D. Keys. A Matrix-free Preconditioner for the Helmholtz Equation based on the Fast Multipole Method. In preparation.
- **H. Ibeid**, R. Yokota, J. Pestana, and D. Keys. Fast Multipole Preconditioners for Sparse Matrices Arising from Elliptic Equations. *Computing and Visualization in Science*, submitted.
- **H. Ibeid**, R. Yokota, and D. Keys. A performance model for the communication in fast multipole methods on high-performance computing platforms. *International Journal of High Performance Computing Applications (IJHPCA)*, 2016.
- R. Yokota, **H. Ibeid**, and D. Keys. Fast Multipole Method as a Matrix-Free Hierarchical Low-Rank Approximation. In *Proc. International Workshop on Eigenvalue Problems: Algorithms; Software and Applications, in Petascale Computing (EPASA 2015)*, Lecture Notes in Computer Science, to appear.
- Y. Ohno, R. Yokota, H. Koyama, G. Morimoto, A. Hasegawa, G. Masumoto, N. Okimoto, Y. Hirano, **H. Ibeid**, T. Narumi, and M. Taiji. Petascale molecular dynamics simulation using the fast multipole method on K computer. *Computer Physics Communications*, Vol. 185, No. 10, pp. 2575-2585 (2014).

- **H. Ibeid**, D. Kaushik, D. Keyes, and H. Ltaief. Toward Accelerating the Matrix Inversion Computation of Symmetric Positive-Definite Matrices on Heterogenous GPU-Based Systems. *Student Research Symposium, The annual IEEE International Conference on High Performance Computing (HiPC)*, 2011.