HIGH PERFORMANCE COMPUTATIONAL CHEMISTRY: BRIDGING QUANTUM MECHANICS, MOLECULAR DYNAMICS, AND COARSE-GRAINED MODELS

by

DAVID M. OZOG

A DISSERTATION

Presented to the Department of Computer and Information Science
and the Graduate School of the University of Oregon
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

March 2017

DISSERTATION APPROVAL PAGE

Student: David M. Ozog

Title: High Performance Computational Chemistry: Bridging Quantum Mechanics, Molecular Dynamics, and Coarse-Grained Models

This dissertation has been accepted and approved in partial fulfillment of the requirements for the Doctor of Philosophy degree in the Department of Computer and Information Science by:

Dr. Allen D. Malony	Chair and Advisor
Dr. Hank R. Childs	Core Member
Dr. Boyana R. Norris	Core Member
Dr. Wibe A. de Jong	Core Member
Dr. Marina G. Guenza	Institutional Representative

and

Scott L. Pratt	Dean of the Graduate School

Original approval signatures are on file with the University of Oregon Graduate School.

Degree awarded March 2017

DISSERTATION ABSTRACT

David M. Ozog

Doctor of Philosophy

Department of Computer and Information Science

March 2017

Title: High Performance Computational Chemistry: Bridging Quantum Mechanics, Molecular Dynamics, and Coarse-Grained Models

The past several decades have witnessed tremendous strides in the capabilities of computational chemistry simulations, driven in large part by the extensive parallelism offered by powerful computer clusters and scalable programming methods in high performance computing (HPC). However, such massively parallel simulations increasingly require more complicated software to achieve good performance across the vastly diverse ecosystem of modern heterogeneous computer systems. Furthermore, advanced "multi-resolution" methods for modeling atoms and molecules continue to evolve, and scientific software developers struggle to keep up with the hardships involved with building, scaling, and maintaining these coupled code systems.

This dissertation describes these challenges facing the computational chemistry community in detail, along with recent solutions and techniques that circumvent some primary obstacles. In particular, I describe several projects and classify them by the 3 primary models used to simulate atoms and molecules: quantum mechanics (QM), molecular mechanics (MM), and coarse-grained (CG) models. Initially, the projects investigate methods for scaling simulations to larger and more relevant chemical applications *within the same resolution model* of

either QM, MM, or CG. However, the grand challenge lies in effectively *bridging* these scales, both spatially and temporally, to study richer chemical models that go beyond single-scale physics and toward hybrid QM/MM/CG models. This dissertation concludes with an analysis of the state of the art in multiscale computational chemistry, with an eye toward improving developer productivity on upcoming computer architectures, in which we require productive software environments, enhanced support for coupled scientific workflows, useful abstractions to aid with data transfer, adaptive runtime systems, and extreme scalability.

This dissertation includes previously published and co-authored material, as well as unpublished co-authored material.

CURRICULUM VITAE

NAME OF AUTHOR: David M. Ozog

GRADUATE AND UNDERGRADUATE SCHOOLS ATTENDED:

University of Oregon, Eugene, OR
Whitman College, Walla Walla, WA

DEGREES AWARDED:

Doctor of Philosophy, Computer and Information Science, 2017, University of Oregon
Master of Science, Computer and Information Science, 2013, University of Oregon
Master of Science, Chemistry, 2009, University of Oregon
Bachelor of Arts, Physics, 2007, Whitman College
Bachelor of Arts, Applied Mathematics, 2007, Whitman College

AREAS OF SPECIAL INTEREST:

Computational Science
High Performance Computing
Molecular Modeling

PROFESSIONAL EXPERIENCE:

Computational Science Graduate Fellow, Department of Energy / Krell Institute, Advisor: Allen Malony, 2013-present

Research Affiliate, Lawrence Berkeley National Laboratory, Advisor: Kathy Yelick, June 2015 - September 2015

Research Affiliate, Argonne National Laboratory, Advisor: Andrew Siegel, June 2014 - September 2014

Research Assistant, Argonne National Laboratory, Advisor: Pavan Balaji, Jeff Hammond, July 2012 - December 2012

Graduate Research Fellow, University of Oregon - NeuroInformatics Center, Advisor: Allen Malony, January 2011 - September 2013

Web Developer, Concentric Sky, November 2009 - December 2010

Graduate Research Fellow, University of Oregon - Department of Chemistry, August 2008 - June 2009

GRANTS, AWARDS AND HONORS:

Juilfs Scholarship, University of Oregon, 2015

Upsilon Pi Epsilon Computer Science Honors Society, 2015

Student travel grant award for IPDPS, 2015

Department of Energy Computational Science Graduate Fellowship, 2013

Travel sponsorship for the Advanced Computational Software Collection Workshop, 2012

PUBLICATIONS:

D. Ozog, J. Frye, R. Gowers, A. Malony, M. Guenza, "Computational Performance Analysis of a Scientific Workflow for Coarse-Grained Polymer Simulations", (in submission), 2016.

D. Ozog, A. Kamil, Y. Zheng, P. Hargrove, J. Hammond, A. Malony, W. de Jong, K. Yelick, "A Hartree-Fock Application using UPC^{++} and the New DArray Library", *International Parallel and Distributed Processing Symposium (IPDPS)*, Chicago, IL, May 23-27, 2016.

D. Ozog, A. Malony, M. Guenza, "The UA↔CG Workflow: High Performance Molecular Dynamics of Coarse-Grained Polymers", *Parallel, Distributed and Network-Based Processing (PDP)*, Heraklion Crete, Greece Feb. 17-19, 2016.

D. Ozog, J. McCarty, G. Gossett, M. Guenza, A. Malony, "Fast Equilibration of Coarse-Grained Polymeric Liquids", *Journal of Computational Science*, presented at the *International Conference on Computational Science (ICCS)* in Reykjavik, Iceland, June 2, 2015.

D. Ozog, A. Siegel, A. Malony, "A Performance Analysis of SIMD Algorithms for Monte Carlo Simulations of Nuclear Reactor Cores", *International Parallel and Distributed Processing Symposium (IPDPS)*, Hyderabad, India, May 25-29, 2015.

A. Salman, A. Malony, S. Turovets, V. Volkov, D. Ozog, D. Tucker, "Concurrency in Electrical Neuroinformatics: Parallel Computation for Studying the Volume Conduction of Brain Electrical Fields in Human Head Tissues", *Concurrency and Computation Practice and Experience*, July 2015.

D. Ozog, A. Malony, A. Siegel, "Full-Core PWR Transport Simulations On Xeon Phi Clusters", *Joint International Conference on Mathematics and Computation, Supercomputing in Nuclear Applications and the Monte Carlo Method (M&C+SNA+MC)*, Nashville, TN, April 19-23, 2015.

D. Ozog, A. Malony, J. Hammond, Pavan Balaji, "WorkQ: A Many-Core Producer/Consumer Execution Model Applied to PGAS Computations". *International Conference on Parallel and Distributed Systems (ICPADS)*, Hsinchu, Taiwan, Dec. 16-19, 2014.

D. Ozog, J. Hammond, J. Dinan, P. Balaji, S. Shende, and A. Malony. "Inspector-Executor Load Balancing Algorithms for Block-Sparse Tensor Contractions". *International Conference on Parallel Processing (ICPP)*, Lyon, France, Oct. 14, 2013.

A. Salman, A. Malony, S. Turovets, V. Volkov, D. Ozog, D. Tucker, "Next-generation Human Brain Neuroimaging and the Role of High-Performance Computing", *International Conference on High Performance Computing and Simulation (HPCS)*, July 2013.

ACKNOWLEDGEMENTS

I thank Professor Allen Malony for guiding me under his research wing and introducing me to the power of using parallel programming to solve large-scale computational science problems. I also thank the members of the Performance Research Lab who went above and beyond the call of duty to teach me a broad set of skills in performance analysis: Sameer Shende, Kevin Huck, Wyatt Spear, and Scott Biersdorff. Special thanks are due to Professor Marina Guenza, who took much time out of her busy schedule to teach me about the intricacies of coarse-grained polymer chemistry. I also thank the scientists that advised and guided me in my internships, laboratory practicum, and job search: Pavan Balaji, Jeff Hammond, James Dinan, Andrew Siegel, Ron Rahaman, Kathy Yelick, Yili Zheng, Amir Kamil, Eric Roman, and Hank Childs. Finally, a big thanks to my other colleagues for their unbridled support, particularly Chad Wood, Daniel Ellsworth, Nicholas Chaimov, Robert Lim, and last but certainly not least, Charlotte Wise. This research was supported by the Department of Energy Computational Science Graduate Fellowship (CSGF) program, and I thank the many people involved with the CSGF for their kindness and encouragement.

I dedicate this to my family: my father taught me that it doesn't matter where you go, but what you do when you get there; my mother taught me to keep my wits about me; and my brother taught me to love programming in Linux.

TABLE OF CONTENTS

LIST OF FIGURES

Figure Page

LIST OF TABLES

Table Page

CHAPTER I

INTRODUCTION, BACKGROUND, AND MOTIVATION

Computational chemistry codes comprise some of the most computationally expensive scientific applications of all time, which explains why they are often at the forefront of high performance computing capabilities, both in terms of hardware [1, 2] and software [3, 4]. These codes make use of novel and advanced algorithms, programming models, modern computer architectures, and even data mining and machine learning techniques [5, 6, 7, 8, 9]. For example, density function theory codes scale to more than 6,000,000 threads [10], large-scale tensor contractions incorporate advanced communication optimal algorithms [11], Hartree-Fock implementations use several novel programming paradigms [12], and molecular dynamics simulations can cross many orders of magnitude in space and time [13]. These simulations are computationally expensive for many reasons, the primary of which is their considerable algorithmic complexity. The most accurate models of atoms and molecules must incorporate *quantum mechanics (QM)*, and such methods can scale up to $\mathcal{O}(n^8)$ in practice, where n corresponds to the number of basis functions used to represent the system. QM algorithms are some of the most computationally expensive of all practical scientific simulations, and they are intractable for all but the smallest of chemical systems. For example, the largest calculations that include coupled cluster theory simulate only a few dozen atoms. On the other hand, various forms of density functional theory are capable of modeling many thousands of atoms, and classical models can track *billions* of atoms [14]. In fact, many chemical models exist, each achieving a different level of accuracy without incurring the extreme cost of full QM treatment. QM may not be

necessary to understand dynamic, thermodynamic, and structural properties of a chemical system, so molecular mechanics (MM) can be used instead. Furthermore, if MM requires tracking too many particles and/or too long of time-scales, then coarse graining (CG) molecular components can still capture relevant physics. One goal of this dissertation is to describe these different models *individually*, highlighting important optimizations made, so that simulations can effectively apply modern high performance computing (HPC) techniques. However, the grand challenge lies in effectively *bridging* these scales, both spatially and temporally, to study richer chemical models that go beyond single-scale physics. Therefore, this dissertation describes the state of the art in existing approaches for multi-scale chemistry simulations, the drawbacks of such implementations, and how best to improve the quality of this software.

The development of multiscale and multiresolution modeling software is important for many computational physics and chemistry simulations. For example, when modeling a biological system, certain regions must contain relatively high detail (such as the active binding site of an enzyme), while other regions require much less detail (such as the water solvent that surrounds the enzyme). Of particular interest are methods that bridge the electronic structure of relatively small regions, described by QM, with surrounding parts described by MM. In fact, the scientists who invented these methods won the Nobel prize in chemistry in 2013 [15]. QM/MM methods have also inspired adding a third level of *coarse-grained* resolution with QM/MM/CG methods in very recent research [16, 17]. Figure 1 shows an example of a QM/MM/CG system in which the enzyme is modeled with QM, several proteins modeled with MM, and the water solvent modeled with CG. The ultimate challenge here is accomplishing

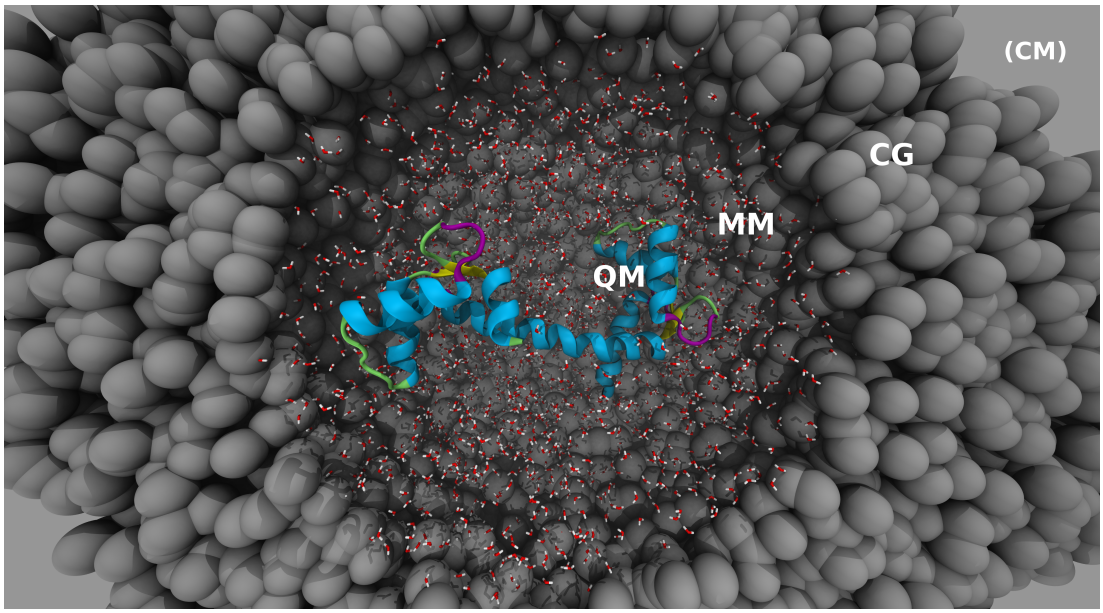


FIGURE 1. An example of a QM/MM/CG system in which the calmodulin protein is modeled with QM, a water layer with MM, and the water solvent with CG (and the outer region possibly modelled with continuum mechanics).

adaptive resolution, which allows for different regions of the simulation domain to be modeled with more or less granularity, depending on the desired level of accuracy. Because of its novelty, adaptive resolution computational software generally lacks thorough performance analyses and efficient application of modern optimizations and HPC techniques. This dissertation describes the state of the art in multiscale computational chemistry, with an eye towards improving developer productivity on upcoming exascale architectures, where we require productive software environments, enhanced support for coupled scientific workflows, adaptive and introspective runtime systems, resilience to hardware failures, and extreme scalability.

This dissertation is organized into five chapters:

- Chapter I: Introduction, Motivation, and Background
- Chapter II: Quantum Chemistry models

- Chapter III: Molecular Mechanics models
- Chapter IV: QM/MM and Beyond
- Chapter V: Conclusion and Future Work

Chapter I provides an in-depth survey of existing methods within QM, MM, and CG modeling that utilize HPC to solve large-scale chemistry problems. The high-level organization of this chapter is as follows:

- Section 1.1: Challenges at extreme scale
- Section 1.2: Quantum Chemistry models
- Section 1.3: Molecular Mechanics models
- Section 1.4: Coarse-Grained models
- Section 1.5: Multiscale and multiresolution techniques
- Section 1.6: Relevant applications

Chapters II and III discuss several research projects within the domains of QM, MM, and CG. Finally, Chapter IV discusses ongoing work in multi-resolution paradigms of QM/MM, MM/CG, and QM/MM/CG.

This dissertation includes prose, figures, and tables from previously published conference and journal proceedings. Chapters I and IV are completely unpublished with no co-authorship, other than that associated with the guidance of Prof. Allen Malony and Prof. Marina Guenza. Chapters II and III, however, contain content from several past paper publications. Section 2.1 includes content on Inspector/Executor load balancing from ICPP 2013 [18]. This work was a collaboration with scientists at Argonne National Laboratory: Jeff Hammond, James Dinan, and Pavan Balaji; and the University of Oregon: Sameer Shende and Allen Malony. Section 2.2 describes the WorkQ system, first presented at ICPADS 2014 [19]. This work resulted in a Directed Research Project with Prof. Malony

and collaborators Dr. Hammond and Dr. Balaji. Section 2.3 includes content about UPC++ and Hartree-Fock from IPDPS 2016 [20]. This work was a collaboration with scientists at Lawrence Berkeley National Laboratory: Kathy Yelick, Wibe de Jong, Yili Zheng, and Amir Kamil; as well as Prof. Malony and Dr. Hammond.

Chapter III contains previously published content that is part of an ongoing collaboration with the Guenza Lab at the University of Oregon. Section 3.1 includes published material from ICCS 2015 [21] and The Journal of Computational Science 2015 [22]. Section 3.2 includes published content from PDP 2016 [23]. Finally, section 3.3 includes new material that is so far unpublished.

1.1 Challenges at Extreme Scale

For several decades, computational scientists have come to rely on the steady increase of CPU clock rates for improving the performance of their application codes. *Moore's Law*, the economic observation that the number of transistors in dense integrated circuit dies roughly doubles every two years, appears to hold into 2015, but there remains another serious problem. *Dennard Scaling*, the observation that power density remains roughly constant as clock rate increases and die size shrinks, has unequivocally ended, which also threatens the end of Moore's Law. Post-petascale machines will struggle to overcome this power wall without drastic adaptations to how computational codes utilize the features of evolving architectures. This will involve efficient load balancing in the face of millions of cores, simultaneous multithreading, vectorizing hardware units, unconventional non-uniform memory hierarchies, and new programming models [24].

Rapidly changing programming environments and the need to scale to millions of cores have lead to a crisis in computational science, wherein codes

often scale poorly, software engineering practices lack agility and rigor, and software is difficult to maintain and extend due to increasing complexity [25]. The computational science community and culture requires a radical shift towards improving software productivity to sufficiently adapt to extreme-scale computing platforms [25, 24, 26, 27, 28]. Common themes arise in these reports as to what is most important for effectively navigating extreme-scale requirements:

- Component or modular software
- Software productivity metrics
- Software methodology and architecture
- Software development infrastructure
- Scientific workflows (often tightly coupled)
- Verification and validation
- Effective development productivity of teams
- Support for legacy codes
- Multiphysics/multiscale component coupling
- Runtime feedback and control

The sections below consider how these topics are addressed in computational chemistry applications. While these requirements are common to most major computational science software projects, computational chemistry codes are no exception. One goal of this dissertation is to address how these requirements are satisfied by previous research and recent advancements in chemistry simulation software, along with promising ideas for future improvements. The following sections will cover advances in computational chemistry applications, highlighting when any of the above requirements are well-supported.

1.2 HPC in Quantum Chemistry

Quantum Chemistry (QC) is the study of atoms and molecules using a quantum mechanical model. In quantum mechanics, energy is quantized, particles obey the Heisenberg uncertainty principle, and wave-particle duality always holds [29]. A thorough introduction to this topic is beyond the scope of this dissertation, so instead, we concern ourselves with the application of QC to the specific domain described in the next introductory Section 1.2.1.. We then focus on applications that utilize parallel computing to push the limits of QC simulations, in terms of their size and higher-order methods that improve accuracy. Section 1.2.2. highlights the fact that there are several categories of QC, each with a different fundamental approach, and the subsequent subsections describe these categories in more detail. Section 1.2.3. describes the dominant parallel programming models used in QC, with a focus on the PGAS model because of its applicability and its wide-spread acceptance in QC codes. Finally, Section 1.2.4. presents and describes some of the more influential algorithms and methods used in QC codes.

1.2.1. Introduction and Basics

The goal of QC calculations is to approximately solve the many-body time-independent Schrödinger equation, $\mathbf{H}\,|\psi\rangle \;=\; E\,|\psi\rangle$, where $\mathbf{H}$ is the Hamiltonian operator, which extracts the sum of all kinetic and potential energies, E, from the wavefunction, $|\psi\rangle$. Here we make the standard set of assumptions: the Born-Oppenheimer approximation (in which neutrons are fixed but electrons move freely), Slater determinant wavefunctions (that easily satisfy the anti-symmetry principle), and non-relativistic conditions. After these approximations, our focus

resides only on the electronic terms of the Hamiltonian and the wavefunction: $\mathbf{H}_{elec}\,|\psi_{elec}\rangle = E\,|\psi_{elec}\rangle$.

The molecular orbitals that express the electronic wavefunction $|\psi_{elec}\rangle$ consist of a sum of *basis functions* from set $\{\phi_j\}$. We desire $\{\phi_j\}$ to be complete, but this is not practical, since it generally requires an infinite number of functions. We therefore truncate to a finite n value large enough to balance the trade-off between accuracy and computational cost:

$$|\psi_i\rangle = \sum_{j=1}^{n} c_{ij}\,|\phi_j\rangle \tag{1.1}$$

Typically, each basis function is composed of one or more *Gaussian primitive* functions centered at the nucleus location. As we will see in Section 1.2.2..1, 6-dimensional integrals containing these primitives are the dominant computational component of QC applications. However, because Gaussian integrals have convenient analytical solutions, the complexity of a single integral is in practice $\mathcal{O}(K^4)$, where K is the number of Gaussians used to represent the basis functions [30, 31].

1.2.2. Overview of the Most Popular QC Methods

Broadly speaking, QC methods fall into two categories: *ab initio*, and everything else. The phrase *ab initio* means "from first principles", which in computational chemistry means that these methods converge to the exact solution of the Schrödinger equation as the collection of basis functions tends towards completeness. The most popular classes of *ab initio* methods are Hartree-Fock, post-Hartree-Fock, and multireference methods. Subsection 1.2.2..1 covers the Hartree-Fock method, and subsection 1.2.2..2 covers various post-Hartree-Fock

methods; but other sources [32, 33] better describe the multireference methods, which are quite similar to post-Hartree-Fock.

Another QC method that some consider *ab initio* is density functional theory (DFT), discussed in subsection 1.2.2..3. DFT is the most widely used of the QC methods, and it has many interesting HPC applications, a few of which we consider below. Finally, we briefly consider semi-empirical methods in subsection 1.2.2..4, and quantum Monte Carlo in subsection 1.2.2..5.

1.2.2..1 Hartree-Fock and the Self-Consistent Field Method

The Hartree-Fock (HF) method is the most fundamental of the QC methods, because it is the fundamental starting point for approximately solving the Schrödinger equation. HF attempts to determine the c_{ij} values that best minimize the ground state energy, in accordance with the variational principle. The *ground state* is the configuration of the molecule, along with all its electrons, that exhibits the lowest possible energy. All other states are called excited states. In computational chemistry, the *variational principle* states that the energy of an approximate wave function is always too high; therefore, the best wavefunction is the one that minimizes the ground state energy. This principle is the foundation for all iterative QC methods: The degree of energy minimization determines the relative quality of different QC results.

By utilizing a numerical technique for iteratively converging the energy, each subsequent iteration becomes more and more consistent with the field that is imposed by the input molecule and basis set. The method is accordingly called the self-consistent field (SCF) method, and Algorithm 1 shows its *de facto* procedure in pseudocode, with the goal of solving the generalized eigenvalue problem $\mathbf{FC} = \mathbf{SC}\epsilon$,

Algorithm 1 The SCF Procedure

Inputs:

1) A molecule (nuclear coordinates, atomic numbers, and N electrons)
2) A set of basis functions $\{\phi_\mu\}$

Outputs:

Final energy E, Fock matrix $\mathbf{F}$, density matrix $\mathbf{D}$, coefficient matrix $\mathbf{C}$

1: Compute overlap terms $S_{\mu\nu}$, core Hamiltonian terms $H_{\mu\nu}^{core}$, and 2-electron integrals $(\mu\nu|\lambda\sigma)$.
2: Diagonalize the overlap matrix $\mathbf{S} = \mathbf{U}\mathbf{s}\mathbf{U}^\dagger$ and obtain $\mathbf{X} = \mathbf{U}\mathbf{s}^{1/2}$.
3: Guess the initial density matrix $\mathbf{D}$.
4: **while** E not yet converged **do**
5: Calculate $\mathbf{F}$ from $H_{\mu\nu}$, $\mathbf{D}$, and $(\mu\nu|\lambda\sigma)$.
6: Transform $\mathbf{F}$ via $\mathbf{F}' = \mathbf{X}^\dagger\mathbf{F}\mathbf{X}$.
7: $E = \sum_{\mu,\nu} D_{\mu\nu}(H_{\mu\nu}^{core} + F_{\mu\nu})$
8: Diagonalize $\mathbf{F}' = \mathbf{C}'\epsilon\mathbf{C}'^\dagger$.
9: $\mathbf{C} = \mathbf{X}\mathbf{C}'$
10: Form $\mathbf{D}$ from $\mathbf{C}$ by $D_{\mu\nu} = 2\sum_i^{N/2} C_{\mu i}C_{\nu i}^*$.
11: **end while**

where $\mathbf{F}$ is the Fock matrix (defined below in Eqn. 1.2), $\mathbf{C}$ is the matrix composed of expansion coefficients from Eqn. 1.1, $\mathbf{S}$ is the matrix composed of *overlap integrals* taken over all space

$$S_{ij} = \int \phi_i^*(\mathbf{r})\phi_j(\mathbf{r})d\mathbf{r} = \langle\phi_i|\phi_j\rangle\,,$$

and ϵ is the energy eigenvalue. Many SCF iterations are required for energy convergence, so steps 1-3 of Algorithm 1 cost much less than steps 5-10. Step 5 normally comprises a majority of the execution time in Hartree-Fock codes, because each element of the Fock matrix requires computing several two-electron integrals:

$$F_{ij} = H_{ij}^{\text{core}} + \sum_{\lambda\sigma}(2(\mu\nu|\lambda\sigma) - (\mu\lambda|\nu\sigma)) \tag{1.2}$$

The two-electron integrals on the right-hand side are plentiful and expensive to compute [34]. They take this form:

$$(\mu\nu|\lambda\sigma) = \iint \phi_\mu^*(\mathbf{r}_1)\phi_\nu(\mathbf{r}_1)r_{12}^{-1}\phi_\lambda^*(\mathbf{r}_2)\phi_\sigma(\mathbf{r}_2)d\mathbf{r}_1 d\mathbf{r}_2 \tag{1.3}$$

As mentioned, there are formally $\mathcal{O}(n^4)$ integrals to compute within the basis set. However, from the definition in 2.2, we see there are only $\sim n^4/8$ unique integrals by permutational symmetry, and the number of non-zero integrals is asymptotically $\mathcal{O}(n^2)$ when Schwartz screening is employed.

While the two-electron integrals comprise the most computation time in SCF, communication and synchronization overheads can very well dominate, particularly at large scale. Inherent communication costs arise from the parallel decomposition of HF codes, leading to load imbalance and synchronization delays. Parallel HF applications also exhibit highly diverse accesses across distributed process memory spaces. As such, HF is well-suited for a programming model that emphasizes lightweight and one-sided communication within a single global address space. This is the subject of Section 1.2.3..1.

Nearly all quantum chemistry codes implement HF, and many contain parallel versions. QC codes with parallel versions include NWChem, Q-Chem, GAMMESS, Gaussian, Psi, CFOUR, GPAW, MOLPRO, ACES III, Quantum ESPRESSO, MPQC and many more. Many consider the most scalable code to be NWChem [3, 4], but there remain serious scalability issues due to the inherent load imbalance and the difficulty in exploiting data locality at scale. The load imbalance comes from the fact that we must bundle the 2-electron integrals into what are called *shell quartets* in order to reuse a lot of the intermediate integral

values. Formally, a shell quartet is defined as:

$$(MN|PQ) = \{(ij|kl) \text{ s.t. } \begin{array}{l} i \in \text{shell } M, \\ j \in \text{shell } N, \\ k \in \text{shell } P, \\ l \in \text{shell } Q\} \end{array}$$

where "shells" refer to the common notion of electrons that have the same principal quantum number, as one learns in their first year of general chemistry. Because this bundling is an *essential* optimization, it introduces more load imbalance to the problem, because some quartets are simply more expensive than others, particularly after screening out quartets that are close to zero. Formally, a two-electron integral is screened if it satisfies this condition:

$$\sqrt{(ij|ij)(kl|kl)} < \tau$$

where τ is the desired screening threshold value.

Optimizing for locality is also very difficult, because tiled accesses across the 4-dimensional space of two-electron integral permutations are wide-spread, the data is sparse, re-distribution is expensive, and the locality patterns strongly depend on the input molecular geometry.

1.2.2..2 Post-Hartree-Fock Methods

Post-Hartree-Fock (PHF) methods improve on the Hartree Fock approximation by including the electron correlation effects that HF mostly ignores

(except for parallel-spin electrons [35], the discussion of which is beyond the scope of this dissertation). In HF, electron correlation is treated in an average way, whereas realistic models must account for the fact that each electron feels instantaneous Coulombic repulsions from each of the other electrons. PFH methods still invoke the Born-Oppenheimer approximation under non-relativistic conditions, but now include more than a single Slater determinant.

The primary PHF methods are configuration interaction (CI), coupled cluster (CC), Møller-Plesset perturbation theory, and multi-reference methods; but there exist many others. Here, we focus on a brief introduction and overview of CI and CC, although it is difficult to appreciate without a full treatment, as in [36].

PHF methods often utilize a notation for describing excited determinants with respect to the reference HF wavefunction, $|\Psi_{\mathrm{HF}}\rangle$. Formally, HF assumes that each electron is described by an independent one-electron Hamiltonian:

$$h_i = -\frac{1}{2}\nabla_i^2 - \sum_{k=1}^{M} \frac{Z_k}{r_{ik}}$$

such that $h_i |\psi_i\rangle = \varepsilon_i |\psi_i\rangle$. Because the total Hamiltonian is assumed separable, the many-electron solution becomes $|\Psi_{\mathrm{HF}}\rangle = |\psi_1\psi_2\cdots\psi_N\rangle$. On the other hand, PHF methods strive to discover the *exact* wavefunction:

$$|\Psi\rangle = c_0 |\Psi_{\mathrm{HF}}\rangle + \sum_{i}^{\text{occ.}} \sum_{r}^{\text{vir.}} c_i^r |\Psi_i^r\rangle + \sum_{i<j}^{\text{occ.}} \sum_{r<s}^{\text{vir.}} c_{ij}^{rs} \left|\Psi_{ij}^{rs}\right\rangle + \cdots \tag{1.4}$$

where the notation $|\Psi_i^r\rangle$ refers to a determinant in which electron i is in excited (or *virtual*) state r and $\left|\Psi_{jk}^{st}\right\rangle$ refers to a doubly-excited determinant in which electron j is in excited state s, and electron k is in excited state t. Notice that if we include all possible configuration states, then we have exactly solved this

form of the electronic Schrödinger equation. This accomplishment is referred to as *full CI*, and it is never accomplished in practice. However, given a truncated form of Eqn. 1.4 (for example, including the first two terms is referred to as CISD), we apply the linear variational method to form the matrix representation of the Hamiltonian, then find the eigenvalues of the matrix. Specifically, we solve the eigenvalue problem:

$$\mathbf{Hc} = E\mathbf{Sc}$$

where $\mathbf{c}$ is now the column-vector of coefficients for our wavefunction in the desired basis set, $\mathbf{S}$ is the overlap matrix, and $\mathbf{H}$ is the CI Hamiltonian.

CC theory, on the other hand, assumes an exponential ansatz operator applied to the reference wavefunction:

$$|\Psi_{\mathrm{CC}}\rangle = e^{\mathbf{T}} |\Psi_{\mathrm{HF}}\rangle$$

where the $\mathbf{T}$ operator definition is:

$$\mathbf{T} = \mathbf{T}_1 + \mathbf{T}_2 + \mathbf{T}_3 + ...$$

$$\mathbf{T}_n = \frac{1}{(n!)^2} \sum_{\substack{a_1 ... a_n \\ i_1 ... i_n}} t_{i_1 ... i_n}^{a_1 ... a_n} a_{a_1}^\dagger ... a_{a_n}^\dagger a_{i_n} ... a_{i_1}$$

Each $t_{i_1 ... i_n}^{a_1 ... a_n}$ term is a tensor with rank $2n$ that represents the set of amplitudes for all possible excitations of n electrons to excited states in virtual orbitals. The terms a_i and $a_a^\dagger$ are the creation and annihilation operators, respectively, that act on electron states. The creation operator adds a row (electron) and column

(orbital) to the Slater determinant, whereas the annihilation operator removes a row and column. By far, the most computationally expensive component of CC codes is the computation of tensor contractions to determine the $\mathbf{T}_n$ terms. This is the object of much advanced research, and we discuss specific accomplishments in Sections 1.2.4..3, 1.2.4..4, and, 1.2.4..5.

1.2.2..3 Density Functional Theory

Density Functional Theory (DFT) is the most widely used QC method (based on the number of citations throughout the history of *Physical Reviews* [37]). It also happens to be the most prevalent method used for large scale simulations, parallel performance studies, and applications in the chemical industry [38]. DFT has the distinct advantage of being able to ignore the complicated features of the electronic wavefunction, while still accounting for correlation effects that Hartree-Fock ignores. It is strongly based on the notion of the electron density, which is defined as the integral over the spin (s_i) and spatial ($\mathbf{r}_i$) coordinates:

$$\rho(\mathbf{r}) = N \sum_{s_1} \cdots \sum_{s_N} \int d\mathbf{r}_2 \cdots \int d\mathbf{r}_N |\Psi|^2$$

where Ψ is actually a function of $\Psi(\mathbf{r}, s_1, \mathbf{r}_2, s_2, \cdots, \mathbf{r}_N, s_N)$. Unlike Ψ, the electron density ρ is a physical observable that can be measured experimentally, and its integral over all space is conveniently the total number of electrons:

$$N = \int \rho(\mathbf{r}) d\mathbf{r}$$

The goal of most DFT methods is to solve the so-called Kohn-Sham (KS) equations (in similar form to the Schrödinger eigenvalue equations) [33] by

representing the total energy of the system as a functional. For example, Thomas and Fermi produced the first DFT model [39] by writing the kinetic energy as:

$$T_{TF}[\rho(\mathbf{r})] = \frac{3}{10}(3\pi^2)^{2/3} \int \rho^{5/3}(\mathbf{r}) d\mathbf{r} \tag{1.5}$$

This notation $F[\rho(\mathbf{r})]$ implies that F is a **functional**, since it takes ρ, which itself is a function of $\mathbf{r}$, as its argument. In a pivotal result, Hohenburg and Kohn famously proved [40] that some functional of total energy, $E[\rho(\mathbf{r})]$, must exist which represents the many-body ground state both *exactly* and *uniquely*. However, this proof specifies neither the form of the functional, nor how to obtain it. Accordingly, KS-DFT research often consists of exploring different forms of these functionals, and classifying their applicability to certain chemical systems.

Solving the orbital coefficients using KS-DFT is very similar to HF [41], except instead of the F_{ij} elements composing the Fock matrix as in Eqn. 1.2, we have

$$F_{ij} = H_{ij}^{\text{core}} + G_{ij}^{J} + \alpha G_{ij}^{K} + \beta G_{ij}^{\text{X-DFT}} + \gamma G_{ij}^{\text{C-DFT}}$$

where the first three terms on the right hand side are the same as in HF, but the final two more difficult terms consist of functionals of the energy, such as in Eqn. 1.5. Here, the constants α, β, and γ can enable spanning the limits of HF and DFT and any hybrid mixture of the two. Because of DFT's computational similarity, it shares the same bottlenecks: calculating all the two-electron integrals, forming the Fock and Density matrices,

1.2.2..4 Semi-Empirical Methods

Semi-empirical methods differ from ab-initio by incorporating empirical observations to account for errors made by assumptions and approximations inherent to solving the Schrödinger equation in practice. For example, including empirical constants in terms of the secular determinant may better approximate resonance integrals [33]. A popular semi-empirical method is the complete neglect of differential overlap (CNDO), which adopts a series of conventions and parameterizations to drastically simplify the application of HF theory. Specifically, CNDO only considers valence electrons and makes the zero differential overlap approximation, which ignores certain integrals [32], reducing the algorithmic complexity from $\mathcal{O}(n^4/8) \sim \mathcal{O}(n^4)$ to $\mathcal{O}(n(n+1)/2) \sim \mathcal{O}(n^2)$.

At this point it is important to note that methods in machine learning [8, 42] often show less error than certain semi-empirical methods. If a semi-empirical does not offer insight from a parameterization, then it may be a better choice to use a machine learning method to improve accuracy.

1.2.2..5 Quantum Monte Carlo

Quantum Monte Carlo (QMC) is briefly considered here for the sake of completeness, but also because of its extreme scalability [43] and promising results on GPUs [44]. There are many different QMC schemes, but within the limited scope of this dissertation it suffices to consider those that fall under the variational Monte Carlo category. In short, this QMC method takes the variational principle (defined in 1.2.2..1) one step further to evaluate the two-electron integrals numerically. Because the energy depends on a given set of variational parameters,

QMC utilizes mathematical optimization to determine the ground state energy by selecting the best set of parameters.

1.2.3. Parallel Programming Models in QC

QC codes have extraordinary demands in term of memory requirements. For instance, consider the well-known hierarchy of CC methods that provides increasing accuracy at increased computational cost [36]:

$$\cdots < CCSD < CCSD(T) < CCSDT$$
$$< CCSDT(Q) < CCSDTQ < \cdots .$$

Here, S refers to truncation at the singles term, D to the doubles term, T to triples, and Q to quadruples (terms with parentheses refer to *perturbation* terms). The simplest CC method that is generally useful is CCSD, has a computational cost of $\mathcal{O}(n^6)$ and storage cost of $\mathcal{O}(n^4)$, where n is again the number of basis functions. CCSD(T) is a very good approximation to the full CCSDT method, which requires $\mathcal{O}(n^8)$ computation and $\mathcal{O}(n^6)$ storage. The addition of quadruples provides chemical accuracy, albeit at great computational cost. CCSDTQ requires $\mathcal{O}(n^{10})$ computation and $\mathcal{O}(n^8)$ storage. Needless to say, these memory requirements quickly become prohibitive for molecular systems of moderate size. For even just a few water molecules with a reasonable basis set such as cc-pvdz, the application can easily consume dozens of GBs, necessitating some form of distributed memory management. Typically, distributed codes distribute the Fock matrix in some sort of block cyclic or irregular block cyclic fashion.

Furthermore, due to the nature of the entire collection of two-electron integrals, processes that own a particular block of the Fock matrix require access to blocks owned by *other* processes during distributed computation. Treating such communication algorithms with point-to-point message passing is intractable and requires high synchronization overhead. QM codes therefore require efficient *one-sided* message passing capabilities in which a process can get or put data into the memory of a remote process *without the explicit receiving communication call, or even participation of the CPU on the remote process*. Nowadays, many modern network interconnects such as InfiniBand, PAMI (on Blue Gene/Q), iWARP, Cray Gemini/Aries, support such remote direct memory access (RDMA) capabilities. These operations are necessary, not only for tiled accesses to compute elements of the Fock and density matrices, but also to support *nxtval* like dynamic load balancing, task assignment, and work stealing [45, 18, 46].

The extraordinary demands of QC simulations require the support and development of novel and productive parallel programming models and paradigms. Most QC codes run on a single compute node, but the most popular frameworks, such as GAMESS, ACE III, NWChem, and Gaussian have parallel implementations. Most parallel HF codes listed in section 1.2.2..1 use MPI for distributed message passing, but GAMESS, NWChem, GTFock do not (necessarily). Interestingly, GAMESS uses the distributed data interface (DDI) [47] for message passing, which originated as an interface to support one-sided messaging. At the time of DDI's inception, the MPI-2 specification included one-sided memory window instantiations (`MPI_WIN_CREATE`) and subsequent put and get operations (`MPI_PUT`/`MPI_GET`) but these functions were apparently not fully offered by any vendor. The DDI programming model emphasizes three types of memory:

replicated data (one copy of small matrices one each CPU), distributed data (for large matrices or tensors spread across the cluster), and node-replicated (for data to be stored once per compute node).

1.2.3..1 PGAS in Quantum Chemistry

The algorithmic characteristics and resource requirements of HF (and post-HF) methods clearly motivate the use of distributed computation. HF tasks are independent and free to be computed by any available processor. Also, simulating a molecule of moderate size has considerable memory requirements that can easily exceed the memory space of a single compute node. However, at a high level of abstraction, *indexing into distributed HF data structures need not be different than indexing shared memory structures*. This programming abstraction is the basis of the partitioned global address space (PGAS) model for distributing and interacting with data. In computational chemistry, this model is advantageous for interacting with arrays and tensors productively and efficiently.

The PGAS model utilizes one-sided communication semantics, allowing a process to access remote data with a single communication routine. Remote memory access (RMA) is particularly advantageous in HF applications for three primary reasons. First, HF exhibits highly irregular access patterns due to the intrinsic structure of molecules and the necessary removal of integrals from the Fock matrix in a procedure called "screening". Second, there is a need for efficient atomic accumulate operations to update tiles of the global matrices without the explicit participation of the target process. Finally, dynamic load balancing in HF is usually controlled by either a single atomic counter [3], or many counters [48, 18], both of which require a fast one-sided fetch-and-add implementation.

The NWChem software toolkit [49] paved the way towards the ubiquity of PGAS in computational chemistry using the Global Arrays (GA) library and the underlying ARMCI messaging infrastructure [3, 4]. GTFock followed suit, also using GA for tiled accesses across the irregularly distributed Fock and density matrices. The GA model is applicable to many applications, including ghost cells and distributed linear algebra, but GA's primary use today is for quantum chemistry. GA is limited to a C and Fortran API, but does have Python and C++ wrapper interfaces. There exist many other PGAS runtimes, including but not limited to: UPC++, Titanium, X10, Chapel, Co-Array Fortran, and UPC.

1.2.4. Key Algorithmic Improvements

This section highlights some key algorithmic improvements made to QC codes, particularly in the area of parallel computing and HPC. Parallel strong scaling is crucial for QC codes, but the inherent load imbalance and communication bound nature of the calculations require innovative optimization approaches. The following sections describe several such optimizations.

1.2.4..1 Load Balancing: Static Partitioning + Work Stealing

Section 1.2.2..1 describes why the primary hindrance to scalability in QC codes is often due to load imbalance, and much research tackles this problem [50, 51, 18]. Although individual two-electron integrals do not possess drastic differences in execution time, the crucial issue is that *bundles* of shell quartets can vary *greatly* in computational cost. It is necessary to designate shell quartets as task units in HF codes because it enables the reuse of intermediate quantities shared by

basis functions within a quartet [52, 48]. The goal is to assign these task units to processors with minimal overhead and a schedule that reduces the makespan.

NWChem balances load with a centralized dynamic scheduler, which is controlled by a single global counter control referred to as `nxtval` (for "next value"). Each task has a unique ID, and a call to `nxtval` *fetches* the current ID of an uncomputed task, then atomically *adds* 1 to the counter so the next task gets executed. Once the counter reaches the total number of tasks, all work is done. For this reason, the performance of RMA fetch-and-add operations is very important for the scalability of computational chemistry codes like NWChem and GTFock. This has motivated the implementation and analysis of hardware-supported fetch-and-ops on modern interconnects, such as Cray Aries using the GNI/DMAPP interfaces [45].

The `nxtval` scheme exhibits measurable performance degradation caused by network contention, but it can be alleviated by an informed static partitioning of tasks and the addition of atomic counters to every process or compute node. This strategy is called Inspector/Executor load balancing, which shows substantial speedup in NWChem [18]. The GTFock project takes this notion one step further by following the static partitioning phase with *work stealing* [48, 46]. During the local phase, each process only accesses the local memory counter; however, during the work stealing phase, the process accesses other counters remotely. As illustrated in Algorithm 2, each process in GTFock begins an SCF iteration by prefetching the necessary tiles from the global density matrix and storing them into a local buffer. After all local tasks are computed, the global Fock matrix is updated. Then, each process probes the `nxtval` counters of remote processes, looking for work to steal. This algorithm results in many more *local* fetch-and-adds than *remote*, which has

Algorithm 2 Load balance and work stealing

```
1: Determine total number of tasks (after screening).
2: Statically partition tasks across process grid.
3: Prefetch data from DArrays.
4: while a task remains in local queue     /* fetch_add local integer */
5:     compute task
6: end while
7: update Fock matrix DArray via accumulate
8: for Every process p
9:     while a task remains in p's queue    /* fetch_add remote integer */
10:        get remote blocks
11:        compute task
12:    end while
13:    update Fock matrix DArray via accumulate
14: end for
```

important performance implications for how the operations take place. GTFock shows good Xeon Phi performance using OpenMP and vectorized two-electron integral calculation with the OptErd library [46]. The UPC++ PGAS extension for C++ [53] also improves performance by up to 20% when using the new DArray library [20], as illustrated in Fig. 2.

1.2.4..2 Diagonalization vs. Purification

One of the computational bottlenecks of QM codes is the diagonalization of the Fock matrix, which is approximately $\mathcal{O}(n^3)$. Most parallel QM implementations make use of a parallel linear algebra library such as ScaLAPACK, which is a reasonable choice in terms of scalability. However, an alternative "diagonalization-free" technique called *purification* [54] claims to achieve linear scaling by exploiting the fact that elements of the density matrix are short range in coordinate space. This means that matrix elements, $\rho_{ij} \to 0$, as their pairwise distances, $R_{ij} \to \infty$. By truncating elements beyond a certain cutoff distance, this method may achieve linear scaling with system size.

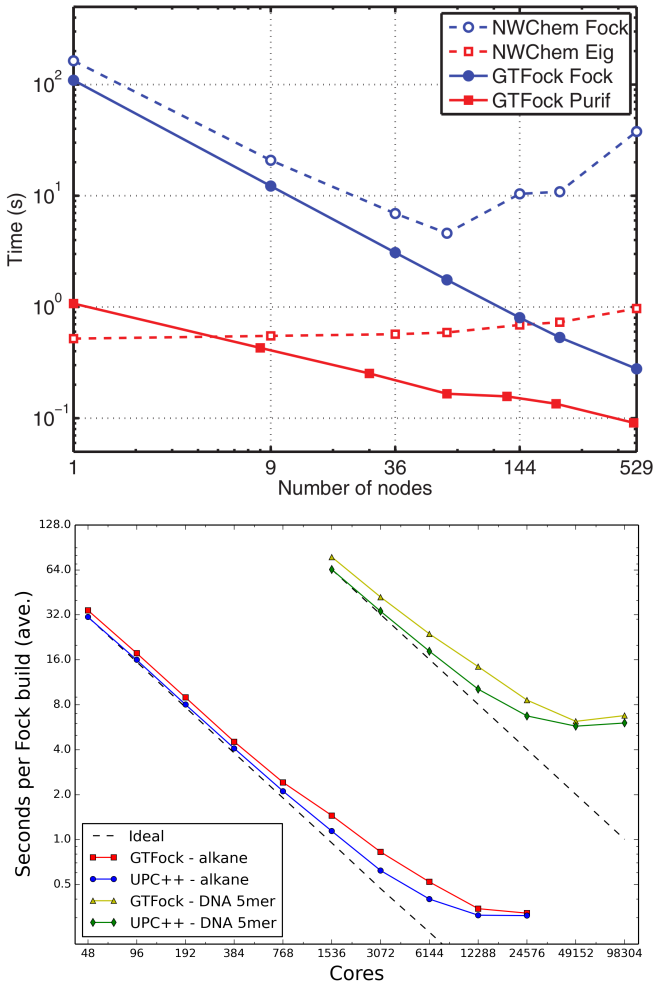


FIGURE 2. GTFock improves NWChem's performance of Hartree-Fock calculations. (The top plot is from [46]).

GTFock and the UPC++ HF implementation described in the previous subsection both make use of the purification technique, but more needs to be done to compare its performance relative to standard diagonalization methods. For instance, we know that GTFock SCF iterations take far less time than NWChem iterations, but we do not yet know about convergence properties of the estimated ground state energy between the two approaches. For instance, the performance per iteration of purification may be better than diagonalization, but the convergence may be slow enough to deem these improvements fruitless.

Future work should consider such convergence behavior in the context of overall application performance.

1.2.4..3 Coupled Cluster and Tensor Contractions

For higher level QM methods such as coupled cluster, the tensor contraction is the most important operation related to achieving scalable performance. The tensor contraction itself is essentially a matrix multiplication, except between tensors, which are multidimensional arrays. In fact, tensor contractions can be reduced to matrix multiplication by a series of index reordering operations. Most of the load imbalance in NWChem arises from such a decomposition of multidimensional tensor contractions into a large collection of matrix multiplication operations. Interesting work by Solomonik et al. [55] considers a cyclic distribution of tensor data with the Cyclops Tensor Framework (CTF), which preserves symmetry and drastically reduces inherent load imbalance showing significant speedups over ScaLAPACK and NWChem. Further work proves a communication *optimal* algorithm on the 5D torus of BG/Q for dense tensor contractions. The RRR framework (for Reduction, Recursive broadcast, and Rotation) shows significant speedups over CTF for certain tensor contractions, such as:

$$C[i,j,m,n] = A[i,j,m,k] \times B[k,n]$$

but not for others, such as:

$$C[i,j,m,n] = A[i,j,k,l] \times B[k,l,m,n]$$

RRR generates several algorithms based on different iteration space mappings and compatible input distributions [11]. The time to generate these algorithms is not reported in this work, and it would be interesting to see how the chosen algorithm depends on tensor size, network topology, and amount of memory per compute node.

1.2.4..4 DAG Dependencies/Scheduling

Not only are tasks within a single tensor contraction of coupled cluster calculations independent, but so too are *some* tensor contractions independent of each other. Recent work has developed directed acyclic graphs (DAGs) that represent the dependencies between the tensor contractions in CCSD calculations [11]. Using this DAG as a task graph, the tensor contractions are grouped into several pools (see Fig. 3) and executed with fewer global barrier synchronization costs compared to NWChem and coupled cluster. Subsequent work takes this notion one step further to construct a dataflow-based execution by breaking the CC kernels down into a large collection of fine-grained tasks with explicitly defined dependencies [56]. This work uses the PaRSEC dataflow framework directly on the Tensor Contraction Engine of NWChem.

1.2.4..5 Accelerator applications

Implementation and performance studies of QM methods on accelerated architecture, such as GPUs and the Intel Xeon Phi Many Integrated Core (MICs) have been quite prevalent the past several years. There are too many to describe in great detail here, so this section mentions only a few with impactful results.

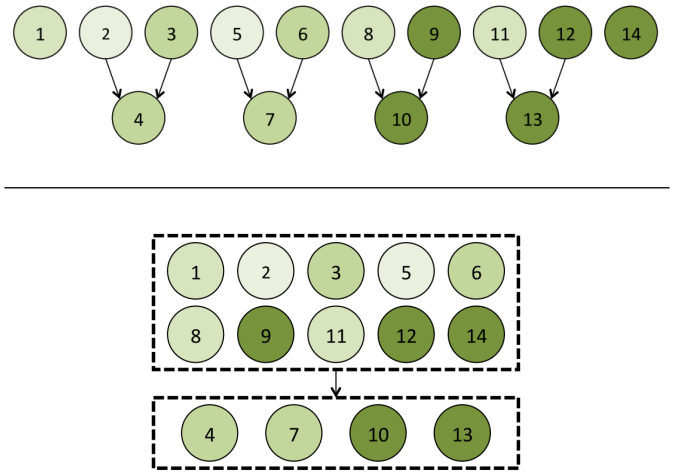


FIGURE 3. An example of scheduling DAG dependencies across tensor contractions in coupled cluster doubles (CCD). This image is from [51].

Much work has considered running two-electron integrals on GPUs [31, 57, 58, 59] with satisfactory results. Now, many QM frameworks enable two-electrons to be run on GPUs including ABINIT, ACES III, ADF, BigDFT, CP2K, GAMESS, GAMESS-UK, GPAW, LATTE, LSDalton, LSMS, MOLCAS, MOPAC2012, NWChem, OCTOPUS, PEtot, QUICK, Q-Chem, QMCPack, Quantum Espresso/PWscf, QUICK, TeraChem, and VASP [60]. DePrince and Hammond implemented the first coupled clusters code on GPUs [61], showing a 4-5 speedup relative to a multithreaded CPU in 2010. In their implementation, the entire SCF iteration takes place on the GPU, but only CCD was implemented. Now, there is a CCSD(T) GPU implementation in NWChem [62].

On the Xeon Phi, Apra et al. optimize CCSD(T) code for the MIC using offload mode [63]. OptErd (the electron repulsion integral library used by GTFock) also makes use of MICs in offload mode [46]. Shan et al. tune the TEXAS integral module, used by NWChem to run more efficiently on multiple architectures, including the Xeon Phi [34]. They also implement MIC optimizations for CCSD(T)

in *native* mode [64], which may be more useful for future Knights Landing and Knights Hill architectures.

Other recent work focuses on enhancing rapid development of QM codes by generating code to run on several novel architectures. For example, Titov et al. use metaprogramming to generate SCF code for the GPU and MIC using either CUDA, OpenCL, or OpenMP based parallelism [65]. Also, the KPPA project by Linford et al. [66] automatically generates C or Fortran90 code to simulate chemical kinetic systems, and the same approach could likely be useful for QM codes as well.

1.2.4..6 Scientific Workflows in QM

This section briefly highlights work in scientific workflows applied directly to QC. In particular, the MoSGrid project [67, 68, 69] has established a science gateway using web-based services for end users to design complex workflows and meta-workflows. MosGrid has considered 3 interesting general workflow use cases in QC. Fig. 4 shows the workflow diagrams for 1) high-throughput analysis of X-ray crystallography files, 2) transition state (TS) analysis and 3) parameter sweeps. While these use cases are extremely interesting in terms of automation, more needs to be done to establish how such workflows and *collections* of workflows should most efficiently be scheduled, and how resources should be allocated, to accomplish such workflows. Workflow management is clearly becoming more important in modern scientific endeavors, and as we will see in following sections, this is particularly true for multiscale and multiresolution computational chemistry.

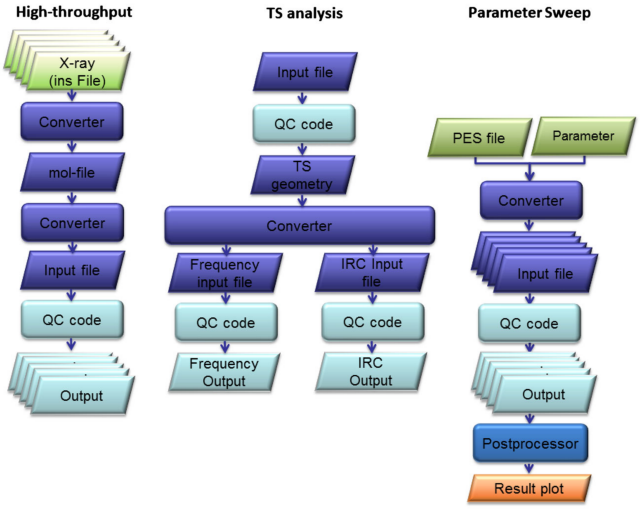


FIGURE 4. 3 QM workflow use cases supported by MosGrid. This diagram is from [69].

1.3 HPC in Molecular Mechanics/Dynamics

This section highlights the primary aspects of molecular dynamics simulations, and how they have evolved alongside developments in HPC. We consider the most scalable and extendible molecular dynamics software frameworks and how they do and do not address the challenges of extreme scale computing listed in Section 1.1.

1.3.1. Introduction and Basics

Molecular Dynamics (MD) simulation is a type of N-body simulation in which the particles of interest are atoms and molecules. MD simulations differ greatly from quantum mechanical simulations: Instead of solving the Schrödinger equation, they solve Newton's equations of motion, which model the classical physics of particle-particle interactions. While MD mostly ignores quantum electronic properties of atoms, it is still able to capture relevant thermodynamics, dipoles, and some reaction pathways/mechanisms. The most important component

of accurate MD simulations is the *potential function* (sometimes called the force field or the interatomic potential), which describes how atoms will interact based on their positions. Incidentally, the evaluation of the potential function is also the most expensive computational component of MD simulations. Potentials must account for all the different forms of energy within molecular systems, both bonded (bond stretching, valence angle bending, and dihedral torsions) and non-bonded (electrostatics and van der Waals forces).

The potential energy function of most MD simulations includes the following contributions:

$$U_{\text{total}} = U_{\text{bond}} + U_{\text{angle}} + U_{\text{dihedral}} + U_{\text{VDW}} + U_{\text{Coulomb}} \tag{1.6}$$

U_{bond} corresponds to the bond stretching (as in the top 2-atom molecule in Fig. 5), which in the simplest non-rigid case is modeled as a harmonic spring:

$$U_{\text{bond}} = \sum_{\text{bond } i} k_i^{\text{bond}} (r_i - r_{i(eq)})^2 \tag{1.7}$$

Similarly, U_{angle} is the bond angle term (as in the 3-atom molecule in Fig. 5), which is also often modeled harmonically:

$$U_{\text{angle}} = \sum_{\text{angle } i} k_i^{\text{angle}} (\theta_i - \theta_{i(eq)})^2 \tag{1.8}$$

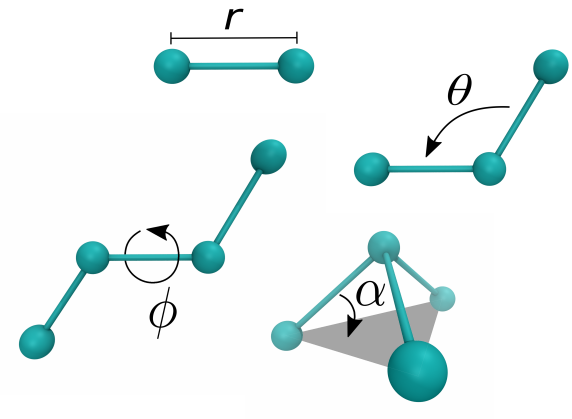


FIGURE 5. Diagram showing the components of a potential $U(r)$ in MD simulations.

$U_{dihedral}$ corresponds to the torsional interactions between 4 atoms (see angle ϕ in Fig. 5) and is usually more sinusoidal in nature:

$$U_{\text{dihedral}} = \sum_{\text{dihedral } i} \begin{cases} k_i^{\text{dihedral}}[1 + \cos(n_i\phi_i - \gamma_i)], n_i \neq 0 \\ k_i^{\text{dihedral}}(0_i - \gamma_i)^2, n = 0 \end{cases}$$

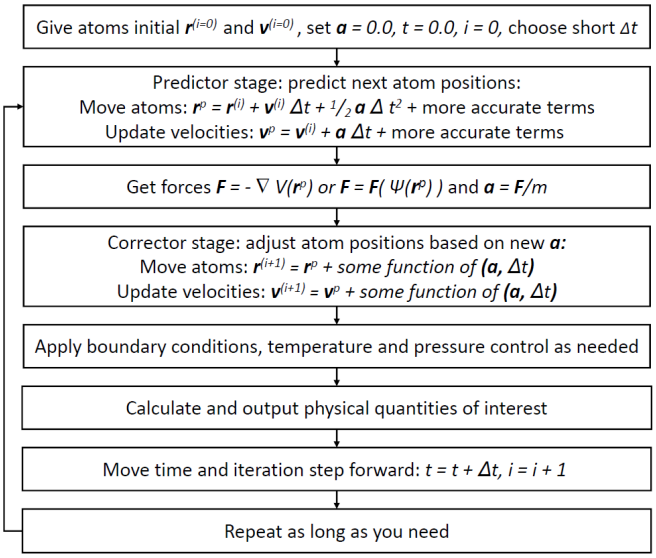
U_{VDW} is the van der Waals term, which accounts for dipole-related forces, and is usually described by some sort Lennard-Jones potential:

$$U_{\text{VDW}} = \sum_i \sum_{j>i} 4\varepsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right] \tag{1.9}$$

Finally, U_{Coulomb} accounts for the more long-range electrostatic interactions, which is almost always described by the standard potential energy between two point charges:

$$U_{\text{Coulomb}} = \sum_i \sum_{j>i} \frac{q_i q_j}{4\pi\varepsilon_0 r_{ij}} \tag{1.10}$$

Algorithm 3 The standard MD simulation algorithm [70].



An important concept in MD simulation is the enforcement of periodic boundary conditions (PBCs). Normally, an MD simulation consists of a simulation cube of specified length L (but some software supports more advanced shapes such as parallelepipeds). What happens if a particle leaves the simulation box? PBCs suggest that the particle should re-enter at the opposite face of the cube. In this way, a simulation of a unit cell with PBCs is equivalent to simulating an infinite system with a repeating unit cell, as illustrated in Fig. 6. The PBC technique upholds thermodynamic consistency, and is good for spatial load balancing, which we discuss in Section 1.3.3..1.

The basic MD algorithm is shown in Algorithm 3. In summary, we first apply an integrator at each time step to determine the movements of atoms. Then we satisfy any necessary boundary or thermodynamic conditions, calculate any desired quantities, and ultimately move on to the next timestep.

In MD software implementations, the naïve approach is to calculate the particle interactions from Eqn. 1.6 for all unique pairs. This approach is $\mathcal{O}(n^2)$,

which is prohibitive for systems of moderate size. However, it is standard practice to exploit the short-ranged nature of certain components of the total potential energy function. For instance, the first 4 terms of Eqn. 1.6 are for very short-ranged bonded atoms. Furthermore, the Lennard-Jones potential from Eqn. 1.9 quickly drops to zero as r increases. By imposing a *cutoff* distance outside of which to neglect any short-range particle interactions, we can drastically reduce the complexity of the simulation to $\sim \mathcal{O}(n)$. Unfortunately, the Coulombic component of the potential is long-ranged by nature, because it drops of as $1/r$. However, methods using Ewald summations can exploit the power of the fast Fourier transform (FFT) to very quickly calculate this long-range potential, which we consider in more detail in Section 1.3.3..2.

Finally, it is standard for each particle to have a *neighbor list*, which is queried during each simulation timestep to efficiently track necessary particle interactions [71] within a given cutoff distance. This has been deemed the most sensitive and important parameter for performance of MD simulations [14]. It is even more important for parallel implementations, because we must update the neighbor list as particles move outside of the relevant cutoff region. This involves frequent communication, particularly when the number of processes is high, the dynamics are fast, and/or the cutoff distance is large. We consider these issues in more detail in Section 1.3.3..1.

1.3.2. MD Force Fields

The previous section presented very simple forms for the potential energy components in Eqns. 1.7, 1.8, 1.9, and 1.10. Together, these simple functions comprise a *force field* for MD simulation, which can also be thought of as a

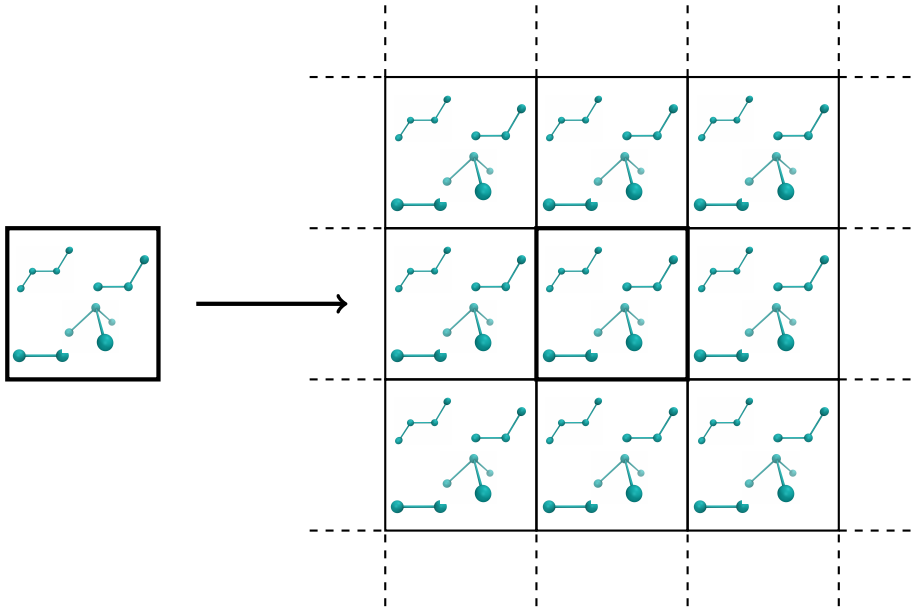


FIGURE 6. An example of periodic boundary conditions.

software implementation for evaluating the potential energy function from Eqn. 1.6. There are *many* such force field implementations, and each is relevant to specific chemical systems of interest. Perhaps the primary challenge of MD simulation studies is that there is currently no general purpose force field - it is necessary to choose the best fit for a given chemical system.

Just as DFT has a very large number of functionals available, so too are there a large number of force fields. The primary categories are classical force fields, polarizable force fields, coarse-grained force fields, and water models. For MD, the more popular options include AMBER, CHARMM, GROMOS, and UFF (the latter being an attempt at a *general* force field) [33].

The total combinatorial space of atoms, molecules, chemical systems, simulation techniques, and force fields is dauntingly large. Interesting work considers high-throughput computational materials design to alleviate the burden of exploring the entire space [5, 8, 72, 73]. These projects combine MD and QM

methods with data mining and large-scale database management to generate and analyze enormous amounts of simulation data towards the discovery and development of novel materials. However, little work considers machine learning techniques for choosing appropriate force fields given an input chemical system to minimize error, despite the clear similarity to other applications [74]. This may be promising future work.

1.3.3. High Performance MD Software Frameworks

The diversity of MD simulation software frameworks is dauntingly diverse, however, they can generally be grouped into two different camps. Many MD simulation frameworks focus primarily on atomistic modeling, such as NAMD [75], AMBER [76], DL_POLY [76], CHARMM [77], and Simpatico [78]. Such frameworks require subtle improvements to force field parameters, and high-level features such as workflow support, but in general the algorithmic developments are well-established [79]. On the other hand, mesoscopic simulation frameworks that model soft matter are relatively less established, because their parameters are more dependent on the chemical system of interest, which may require more complex parameter tuning, and many different modeling methods exist, some of which are more applicable to a particular system setup. The latter type of framework includes LAMMPS, GROMACS, and ESPResSo^{++}. These simulation frameworks are the subject of the following three sections.

1.3.3..1 LAMMPS and Parallel Decomposition Schemes

LAMMPS is a classical molecular dynamics (MD) code which emphasizes parallel performance and scalability. It deploys distributed-memory message-passing

parallelism with MPI using a spatial decomposition of the simulation domain. It has GPU (CUDA and OpenCL) and OpenMP capabilities, and also runs on the Intel Xeon Phi. LAMMPS runs from an input script, which has advantages, such as a low barrier to entry, but also disadvantages, because the input script language is unique and difficult to extend. LAMMPS supports modeling atoms, coarse-grained particles, united-atom polymers or organic molecules, all-atom polymers, organic molecules, proteins, DNA, metals, granular materials, coarse-grained mesoscale models, point dipole particles, and more (including hybrid combinations).

Pivotal work in comparing spatial-decomposition, force-decomposition, and atom-decomposition performance [80].

Atom decomposition: With N total atoms and P total processors, the *atom decomposition* simply assigns N/P atoms to each processor. Throughout the simulation, each processor calculates forces and updates the positions only of the atoms which they are assigned. This decomposition implies that a processor will compute all interactions associated with a given atom. This is generally considered the simplest decomposition scheme, but it may require extra communication because each process requires the coordinates of atoms from many other processors. However, if the all-to-all operation is well-implemented on a given network, then this operation is reasonably efficient, particularly when accounting for Newton's 3^{rd} law to reduce computation in half. Load balance in atom decomposition is good if the row blocks in F have the same number of zeros. This is the case if atomic density is relatively uniform throughout the simulation box. If not, then it helps to randomly permute the number schemes to be less influenced by geometric arrangement.

Force decomposition: In the force matrix F, each element F_{ij} represents the force that atom i feels due to atom j. F is quite sparse because of the fast $1/r^d$ dropoff (d is a positive integer ≥ 2) of short-range potentials combined with the practical use of a pairwise cutoff. Also, it is easy to see by Newton's 3$^{\text{rd}}$ law that F is a symmetric matrix. With the atom decomposition described above, each processor is assigned N/P rows of the F matrix (or equivalently N/P columns). On the other hand, the *force decomposition* technique distributes F into smaller blocks, each $N/\sqrt{P} \times N/\sqrt{P}$. This has the advantage of improving communication costs to $\mathcal{O}(N/\sqrt{P})$ compared to $\mathcal{O}(N)$ in atom decomposition [80]. However, load balance may be worse in force decomposition because the blocks must be uniformly sparse in order for processors to do equal amounts of work. Even if atom density is uniform, geometrically ordered atom identifiers create diagonal bands in F. As in atom decomposition, a random permutation of the numbering scheme helps to reduce load imbalance.

Spatial decomposition: Spatial decomposition involves subdividing the simulation domain into separate 3D boxes, and assigning atoms within a box to a particular processor. As atoms move through the simulation domain, they are reassigned to the appropriate processor based on the spatial decomposition. While it may seem that spatial decomposition may suffer from poor load balance for simulations with spatially-varying density distributions, there are now several options in LAMMPS to improve the load balance. For instance, the `balance` command applies a recursive multisection algorithm to adjust the subdomain sizes in each dimension of the simulation box. Furthermore, when using multi-threading in LAMMPS, an atom decomposition is used for on-node parallelism, whereas spatial decomposition is used for off-node parallelism.

In summary, there are three practical ways to decompose MD simulations, and LAMMPS developers were among the first to study and analyze these alternatives; furthermore, their findings influenced many other future implementations. Figure 7 shows their results comparing the three decomposition schemes using MPI on a Cray machine [80]. We see that the best choice of decomposition scheme depends on both the scale and the chosen cutoff. This phenomenon was relevant on the outdated Cray T3D, and is certainly relevant today with the diversification of on-node architectures/memory hierarchies, network topologies, and programming models. Many recent exploratory MD applications (some on the latest many-core architectures [81]) deploy strict domain decomposition [82] based on such pivotal work as this. However, it is important to keep in mind that the other alternatives (or hybrid schemes) may be better suited to extreme scale systems where concepts such as minimizing communication and exploiting vectorization are paramount for scalable performance.

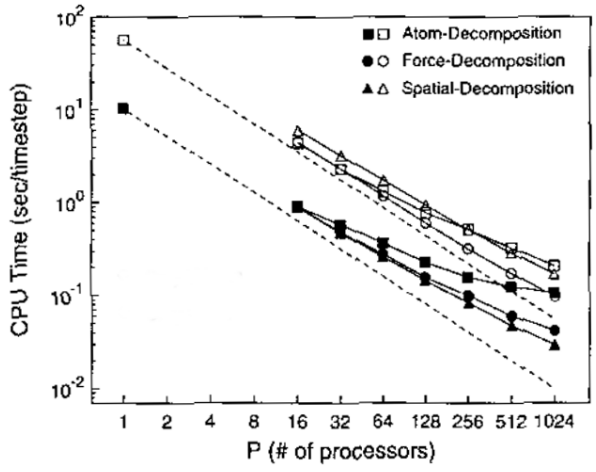


FIGURE 7. Comparison of the 3 MD decompositions for two different cutoff lengths. Although these results are dated, the concepts are important for understanding the consequences of domain decomposition techniques when porting MD codes to modern hardware systems using productive programming models. This figure is from [80].

1.3.3..2 GROMACS and Ewald Summation Techniques

This section highlights the unique aspects of the GROMACS MD framework that improve its performance and scalability. In order to understand, a brief introduction to Ewald summations and particle-mesh Ewald (PME) is necessary (a more thorough introduction is found in [83]). The long-range component of the potential energy function is shown in Eqn. 1.10. The goal for this electrostatic interaction is to solve Poisson's equation given the function for U_{Coulomb}. Ewald summation involves adding a Gaussian function to each point charge in the system.In short, we do this because the Gaussian representation has an *exact* solution to Poisson's equation in reciprocal space. This means that if we transform the spatial coordinates into frequency space using Fourier techniques, then we can trivially solve Poisson's equation across the entire simulation domain. The FFT accomplishes this feat by imposing a domain that is a power of 2 units long in each spatial dimension, then enforcing the point charges to be centers on the nearest grid point.

Unfortunately, FFT scalability does not keep up with the scalability of the short-range component of MD simulations using a given number of processes. However, GROMACS 4.5 was the first framework to solve this problem, and now NWChem utilizes the same technique [84]. This method uses a "pencil" decomposition of reciprocal space, as displayed in Fig. 8. Here, a subset of the total MPI application processes (implemented via MPI communicators) participate in the FFT calculation. At the beginning of each timestep, the direct-space nodes (top of Fig. 8) send coordinate and charge data to the FFT nodes (bottom of Fig. 8). Some number of direct-space nodes (usually 3-4) map onto a single reciprocal-space node. Limiting the computation of the FFT to a smaller number

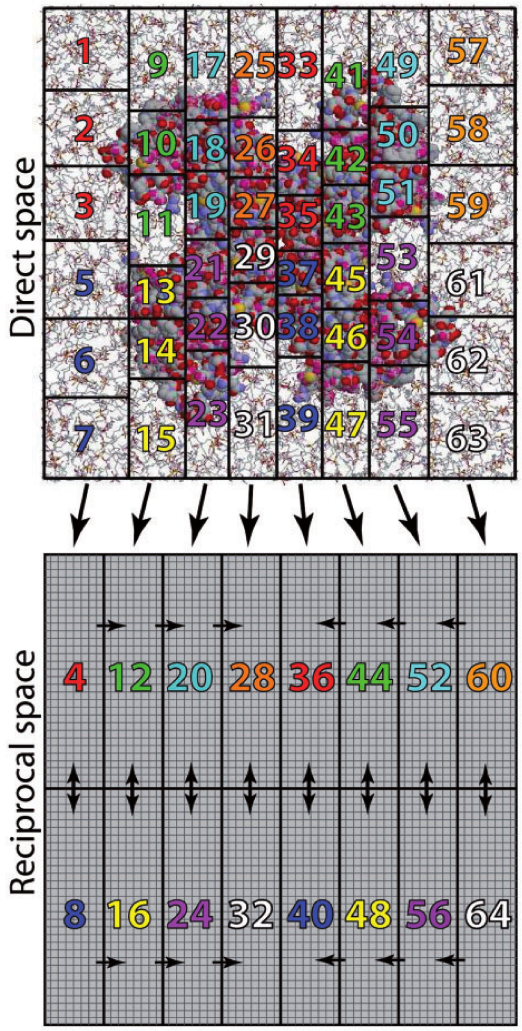


FIGURE 8. GROMACS 3D spatial decomposition combined with 2D decomposition in reciprocal space. This diagram is from [85].

of nodes significantly improves parallel scaling [85], likely due to the communication boundedness of 3D FFTs.

Modern GROMACS (version 5.1) utilizes hierarchical levels of parallelism, as shown in Fig. 9. First, many MD simulations depend on collections of similar instances, which together make up a statistically significant *ensemble*. GROMACS supports tools for easily constructing such ensembles and deploying them on a computer cluster. Second, each simulation within an ensemble is spatially

domain decomposed and dynamically load balances over MPI. Third, non-bonded interactions are handled on GPUs because of the short-ranged calculations compatibility with simultaneous multithreading (SIMT). Finally, CPU cores apply SIMD operations to parallelize cluster interactions kernels or bonded interactions, which now makes heavy use of OpenMP pragmas. This grouping of different components onto the best suited device within the heterogeneous architecture is what makes GROMACS a front-runner in MD simulation frameworks, especially in light of the push towards exascale capability.

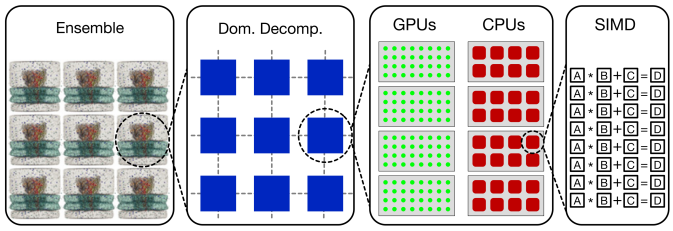


FIGURE 9. The hierarchical levels of parallelism in GROMACS. This diagram is from [84].

1.3.3..3 NAMD

Here, we briefly consider the NAnaoscale Molecular Dynamics (NAMD) framework, which has historically prioritized the strong scaling capabilities of MD simulations more than other frameworks [75]. It's capabilities are similar to that of GROMACS and LAMMPS, but it runs over a more interesting parallel runtime, Charm++. In Charm++, C++ objects called *chares* represent migratable tasks that execute asynchronously and in a one-sided manner. Users are encouraged to "over-decompose" their tasks into chares, because a finer granularity leads to better load balance and adaptability. The runtime of Charm++ manages chares

with a system of queues on each compute node with the goal of hiding latency and promoting asynchronous execution.

NAMD deploys a spatial decomposition into so-called "patches" that fill the simulation box. Unlike other common approaches, the number of patches in NAMD is unrelated to the total number of processes. Charm++ encourages this separation of problem decomposition from hardware resources, because it allows for adaptability in the execution. Furthermore, the assignment of patches to processors can be changed between iterations based on measurements made by the Charm++ runtime. The occasional redistribution of tasks at runtime gives NAMD a key scalability advantage in terms of load balance.

We finally briefly mention some of NAMD's auxiliary accomplishments. First, NAMD researchers explored the implications of multi-lingual programming at scale [86], which led to a rich and flexible software architecture, shown pictorially in Fig. 10 (though this architecture is now out of date). Second, FPGA researchers compiled NAMD with ROCCC, which resulted in an impressive speed-up of the critical region that computes non-bonded interactions [87]. They focused on generating hardware that maximizes parallelism in the FPGA circuit while minimizing the number of off-chip memory accesses.

1.3.3..4 ESPResSo^{++}

ESPResSo^{++} is a re-write of the popular ESPResSo MD software toolkit, which primarily supports simulation of soft matter physics and physical chemistry. Unlike alternatives such as LAMMPS and GROMACS, ESPResSo^{++} boldly prioritizes software *extensibility* over all other concerns during development of MD software, which often (but not always) includes performance [79].

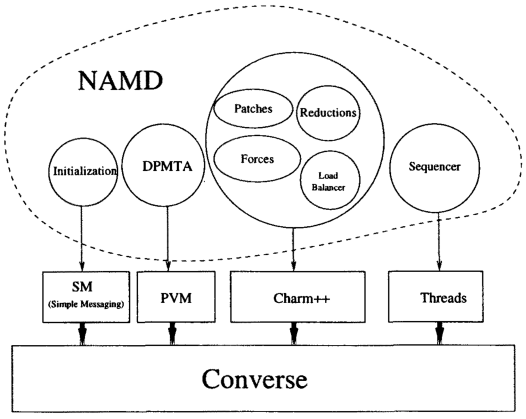


FIGURE 10. NAMD's language architecture. This diagram is from [86].

ESPResSo^{++} supports the extensibility of MD simulations by enhancing readability, maintainability, expandability, and verifiability by means of a well-defined coupling between two software interfaces. First, ESPRessSo^{++} consists of a high-level Python-based interface used for 1) importing relevant modules and packages, 2) setting up the chemical system and its interactions, 3) simulation of the system dynamics, and 4) analysis. ESPResSo^{++} also consists of a lower-level C^{++} interface to the simulation engines and various computational tools. Development of new modules must follow a specific protocol for exposing necessary C++ classes to the Python user interface using the Boost.Python library.

This design is quite useful for writing adaptive simulations with in-situ analysis capabilities. For example, to visualize a plot of the pressure every 100 timesteps using a Python library is trivial:

```
integrator = espresso.integrator.VelocityVerlet(system)
...
for i in range(10):
   integrator.run(100)
   P = espresso.analysis.Pressure(system).compute()
   matplotlib.plot(P, len(P))
```

While this is certainly possible with LAMMPS using the following lines in an input script:

```
thermo_style custom pressure
thermo 100
run 10000
```

it is far more difficult to do something as simple as display a plot every 100 timesteps without resorting to custom interprocess communication (which is not clearly possible without changing LAMMPS source code). Not only is this trivial for ESPResSo^{++}, one can go a step further and visualize simulation progress on-the-fly by sending data over a socket to VMD server:

```
# VMD initialization
sock = espresso.tools.vmd.connect(system)
...
# Send current positions to VMD
espresso.tools.vmd.imd_positions(system, sock)
```

While this is certainly possible with LAMMPS using the following lines in an input script:

```
thermo_style custom pressure
thermo 100
run 10000
```

it is far more difficult to do something in LAMMPS as simple as display a plot every 100 timesteps. This workflow-oriented capability is inherently included in the ESPResSo^{++} software design. The parallel programming model for ESPResSo^{++} is unique in that all communication is done through MPI, but user programs *do not apply an SPMD model*. This unexpected architecture is shown in Figure 11, where

a program using the Python scripting interface invokes a controller that launches MPI4Py parallel programs [88]. Furthermore, processes can either communicate through the Python layer of abstraction, or through the C++ layer, as shown in Figure 11. using the Boost.MPI library, ESPResSo^{++} has been shown to be scalable up to 1024 cores, as shown in Figure 12. While the performance is not as good as LAMMPS at scale, its prioritization of extensibility have allowed for more advanced features such as adaptive resolution, and support for various course grained potentials. Profiles suggest that the performance differences between ESPResSo^{++} and LAMMPS are due to optimizations made within the neighbor list construction and the communication of particle data [79]. In this author's opinion, more needs to be done to quantify the communication overheads. For instance, there are may be overheads associated with using Boost.MPI to send serialized object data instead of sending MPI derived types, as LAMMPS does.

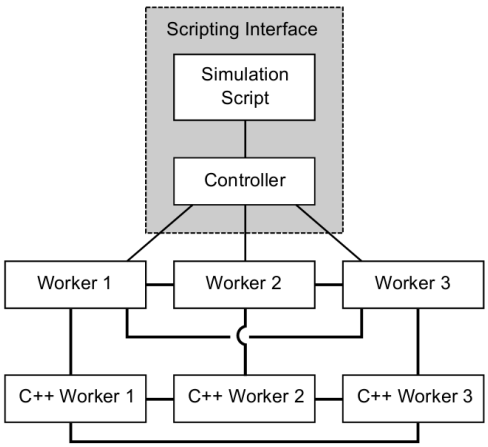


FIGURE 11. The parallel execution model of ESPResSo^{++}. This diagram is from [79].

One reason we emphasize ESPresSo^{++} in this section is because it does a good job addressing most of the extreme-scale requirements listed in Section 1.1. It emphasizes a modular software infrastructure (sometimes sacrificing absolute best

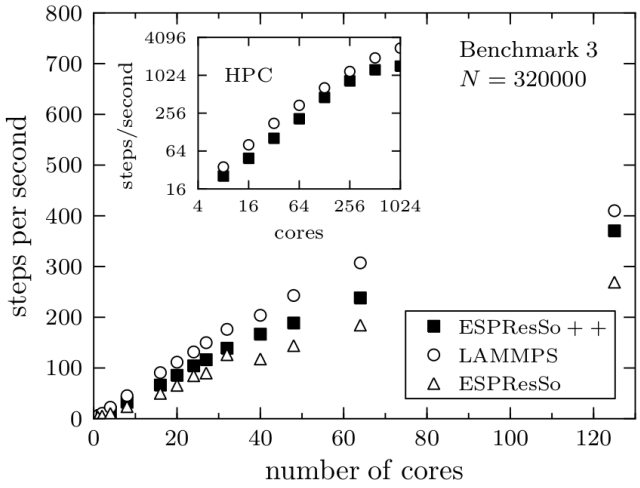


FIGURE 12. ESPResSo^{++} strong scaling compared to LAMMPS. These results are from [79].

performance), enables scientific workflows as an integral part of the software design, and supports multiphysics/multiscale component coupling in the form of adaptive resolution.

1.3.4. Other Noteworthy MD Accomplishments in HPC

This section briefly mentions other noteworthy accomplishments in MD software implementations that utilize HPC. First of all, most key MD frameworks are GPU-accelerated, including ACEMD, AMBER, BAND, CHARMM, DESMOND, ESPResso, Folding@Home, GPUgrid.net, GROMACS, HALMD, HOOMD-Blue, LAMMPS, Lattice Microbes, mdcore, MELD, miniMD, NAMD, OpenMM, PolyFTS, SOP-GPU and more [60]. Other codes, such as ESPResSo^{++} are currently porting and optimizing GPU kernels.

On MICs, miniMD has implemented and published on very effective vectorization techniques that run on the Xeon Phi [81]. LAMMPS, GROMACS, and NAMD also have support for MIC [84], but more optimizations need to be

made to compete with the CPU implementation and to balance work between CPU and devices. Incorporating vectorization into full-fledged MD simulation frameworks on *socket-attached* MIC architectures such as Knights Landing and Knights Hill is a very promising area for future work.

Custom alternative architectures also show *remarkable* potential for MD simulation. For instance, researches have ported NAMD benchmarks to an FPGA and report 800x improvement for single-precision performance and 150x improvement for double precision over a conventional x86 architecture [87]. Also, Anton [1, 2], a special purpose supercomputer designed from the ground up to be optimized to MD simulations, shows impressive performance improvements over general purpose machines, with speedups of up to 180x.

1.4 Coarse-Grained Molecular Simulations

On the largest modern supercomputers, molecular dynamics (MD) simulations of polymer systems contain billions of atoms and span roughly a few nanoseconds of simulation time per week of execution time. Unfortunately, most macromolecular processes of interest contain many orders of magnitude more particles and often bridge microsecond or even millisecond timescales or longer. These include phenomena like microphase separation transition in block-copolymers [89], phase separation in polymer blends and composite materials [90], polymer crystallization, and glass formation and aging [91] to mention just a few. Despite our pervasive access to massive computing power, full *united-atom (UA)* simulations do not come close to representing real-world polymer systems (see Figure 45), because they are too computationally expensive and slow. Simply put, we require new approximation methods that capture the relevant physics and chemistry while requiring fewer

computational resources. The most promising approach is the *coarse-graining (CG)* method, in which groups of atoms are represented as one collective unit. CG has proven to be valuable for eliminating unnecessary degrees of freedom and tackling the scaling complexity of larger problems [92]. The key issue is how to simultaneously maintain solution accuracy and high performance.

The following sections describe some popular approaches for conducting accurate CG simulations. They fall into two broad categories: numerically based and theoretically based. Section 1.4.1. considers a new software framework called VOTCA that provides an interface to several numerical CG approaches. Section 1.4.2. considers a more theoretically driven CG technique called integral equation coarse-graining that relies on principles in statistical mechanics to derive an analytical potential that describes CG models of polymer melts.

1.4.1. VOTCA

There exist several different techniques for applying CG models to molecular systems, such as iterative Boltzmann inversion, force-matching, and inverse Monte Carlo. We begin our discussion of CG techniques by considering the Versatile Object-Oriented Toolkit for Coarse-Graining Applications (VOTCA), because it provides a unified framework that implements many different CG methods and allows their direct comparison [93]. The momentous publication that describes VOTCA [93] provides a nice comparison of these numerically-driven CG techniques, and they are the subjects of the following three sections.

1.4.1..1 Boltzmann Inversion

Boltzmann inversion (BI) is considered to be the simplest method for deriving a CG potential, because it simply involves inverting the distribution functions of a coarse-grained system. Specifically, the bond length, bond angle, and torsion angle distributions are sampled from a simulation trajectory, then inverted to derive the desired potential functions. For example, sampled bond lengths may be fitted to a Gaussian:

$$p(r) = \frac{A}{\omega\sqrt{\pi/2}} \exp\left[\frac{-2(r - r_{eq})^2}{\omega^2}\right]$$

Now, exploiting the fact that a canonical ensemble obeys the Boltzmann distribution between independent degrees of freedom:

$$p(r) = Z^{-1} \exp[-\beta U(r)]$$

(where Z is the standard partition function), this can be inverted to derive the desired harmonic potential:

$$U(r) = -k\beta T \ln[p(r)] = K_r(r - r_{eq})^2.$$

Here, Z becomes an irrelevant additive constant to the CG potential [94].

This simple approach has disadvantages. First, assumes independent degrees of freedom:

$$\begin{aligned} P(r, \theta, \phi) &= \exp[-\beta U(r, \theta, \phi)] \\ P(r, \theta, \phi) &= P_r(r) P_\theta(\theta) P_\phi(\phi) \end{aligned}$$

which may not be true for some systems; however, it *is* true, then BI is an *exact* CG representation of the potential. Another problem is that we require smoothing $U(q)$ to provide a continuous force, which can be accomplished with extrapolation. Finally, we require an atomistic reference system to accomplish BI, be we would prefer a CG model that is independent of any full atomistic simulation.

1.4.1..2 Iterative Boltzmann Inversion

Iterative Boltzmann Inversion (IBI) is very similar to Boltzmann Inversion from the previous section except that the non-bonded CG potential is iteratively updated until it matches the corresponding radial distribution function (RDF) of the atomistic representation:

$$U_{i+1}(r) = U_i(r) - \alpha k_B T \ln \left[\frac{g_i(r)}{g_t(r)} \right]$$

where α is a scaling factor (to reduce large deviations between iterations), $g_i(r)$ is the RDF of the i^{th} iteration and $g_t(r)$ is the target RDF. The RDF is an extremely important quantity in MD simulation studies, because if it is known, it can be used to derive thermodynamic quantities. Fig. 13 shows a graphical representation of an RDF for a liquid chemical system.

1.4.1..3 Inverse Monte Carlo and Force Matching

Two other popular numerical methods for deriving CG potentials are Inverse Monte Carlo (IMC) and Force Matching (FM). An in depth description of these techniques is unnecessary for this discussion, but it suffices to say that IMC is quite similar to IBI, except that it is based on more rigorous thermodynamic arguments.

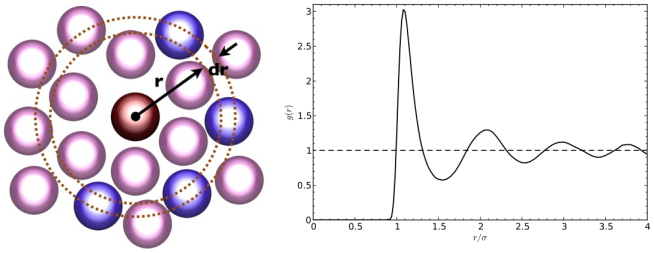


FIGURE 13. An illustration of the radial distribution function, $g(r)$. The left image shows a liquid with "solvation shells" surrounding a central atom. The $g(r)$ function on the right shows the statistical likelihood of a solvation shell at a given distance r, which decreases to zero at large distances.

IMC has advantages over IBI in that it shows better and faster convergence, but also is more expensive computationally.

The FM approach is quite different from IMC and IBI. It is non-iterative like BI, but instead of generating distribution functions from reference simulations, FM generates *force functions* with the goal of matching CG units to atomistic units. It is non-iterative, and therefore less computationally expensive.

1.4.1..4 Iterative Workflow

VOTCA enables the direct comparison of the above CG methods (and potentially many more) by providing a modular interface to a workflow template for generating CG potentials. Fig. 14 shows this iterative workflow. Such capabilities are important for computational chemists because they allow for controlled verification, validation, and testing of new techniques across several different development teams of MD software. Sections below will present a large variety of multiscale and multiresolution simulation methods that would also benefit from a VOTCA-like simulation framework for comparing workflows.
One current drawback of VOTCA, however, is that it is currently focused on numerically-driven techniques for deriving CG potential and force functions, despite

the fact that more consistent techniques exist in analytically-driven techniques for deriving CG potentials. This approach is the subject of Section 1.4.2. below.

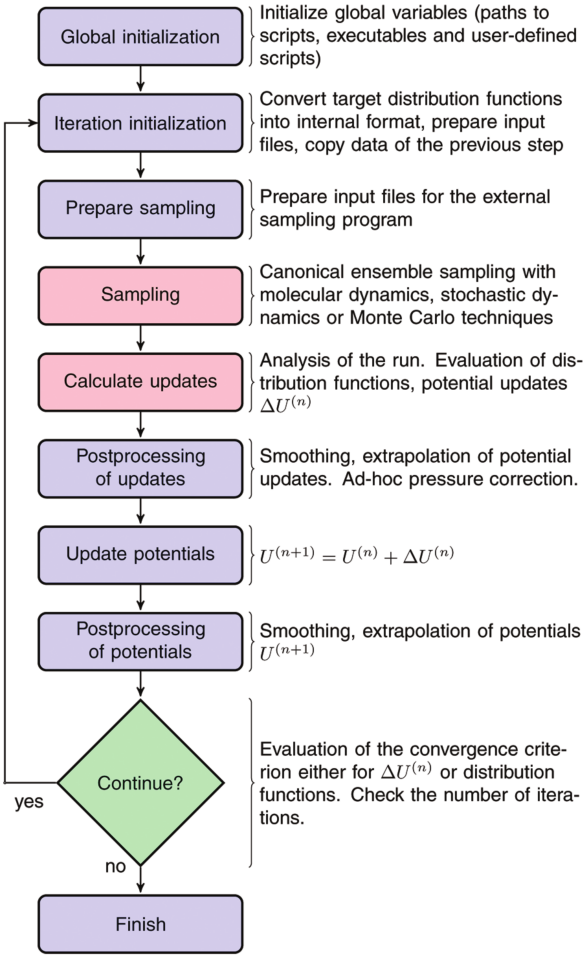


FIGURE 14. VOTCA's iterative workflow. The diagram is from [93].

1.4.2. Integral Equation Coarse Graining Theory

The *Integral Equation Coarse-Grained (IE-CG)* model by Guenza and coworkers [96, 97, 98, 99, 100, 101] adopts an analytically-derived potential and dramatically improves spatial and temporal scaling of polymer simulations, while accurately preserving thermodynamic quantities and bulk properties [102, 103, 104].

Several numerical techniques and force fields exist for performing coarse-grained simulations [105, 106, 107]. However, these methods generally preserve either structure or fully preserve thermodynamics, but not both. As a result, only a small level of coarse-graining is typically adopted to limit the errors in the simulated structure and thermodynamics. In contrast, the IECG model adopts the analytical approach offered by statistical mechanics theory, because it recovers crucial structural and thermodynamic quantities such as the equation of state, excess free energy, and pressure, while enabling a much higher level of coarse-graining and the corresponding gains in computational performance.

Although CG polymer physics is a mature field, little has been done to analyze the performance benefits of CG versus UA representations. While it is clear that CG will exhibit computational gains, does it strong scale to as many processors as the corresponding UA simulation? Likely not, because CG tracks far fewer overall particles, sometimes by orders of magnitude. Also, the scalability of CG simulations likely depends on the granularity factor, e.g, the number of UA coordinates a CG unit represents. Despite the purpose of CG research to improve computational efficiency of MD simulations, the relevant literature lacks measurements that quantify expected computational performance of various CG techniques, and how they scale across different supercomputer architectures. Furthermore, CG related parameters may be chosen to give the

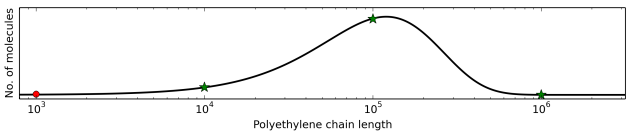


FIGURE 15. A representation of the average polyethylene chain length determined by chromatography experiments [95]. Most studies are limited to very short chain lengths ($\leq$ 1000) due to the prohibitive cost of UA simulations, but recent work freely explores the realistic systems with 10^4 to 10^6 monomers per chain [23].

absolute performance, but do they also give the best accuracy? Two primary goals of this work are to quantify expected performance of IE-CG simulations at large scale, and to evaluate the trade-off between granularity, scalability, and accuracy.

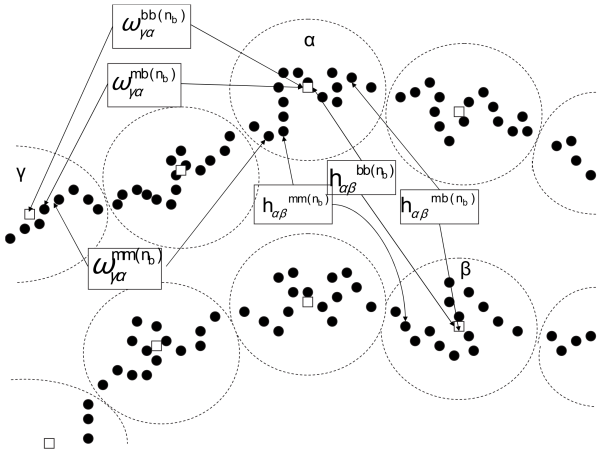


FIGURE 16. A representation of the different block averaged components of the IECG equations. The diagram is from [97].

When considering homopolymers, the IECG model represents polymer chains as collections of monomer *blocks* that consist of many monomers and interact via soft long-range potentials. If each chain has N monomers, we say that there are n_b blocks per chain with N_b monomers per block. The most important quantity used to derive the CG potential is the *time correlation function*, which is essentially a measure of the pairwise influence between particles as a function of distance. They are related by the radial distribution function, $g(r)$ which is defined as:

$$g(r) = \frac{1}{\rho}\left\langle \frac{1}{N}\sum_{i}^{n}\sum_{j\neq i}^{n}\delta(\vec{r}-\vec{r}_{ij})\right\rangle$$

the total correlation function relates to $g(r)$ by:

$$h(r) = g(r) - 1.$$

The Polymer Reference Inter Site Model (PRISM) site-averaged Ornstein-Zernike equation relates the relevant time correlation functions in Fourier Space:

$$\hat{h}^{mm}(k) = \hat{\omega}^{mm}(k)\hat{c}^{mm}(k)[\hat{\omega}^{mm}(k) + \rho\hat{h}^{mm}(k)]$$

from which we derive the CG potential. These equations contain a large number of terms, so they are neglected here, but can be found in the relevant literature [96, 97, 98, 99, 100, 101]. Using this solution for the potential, CG simulations in LAMMPS have been shown to capture the relevant thermodynamics while drastically reducing the number of degrees of freedom [102, 103, 104]. Furthermore, using the analytical approach, there are far fewer tunable parameters than the numerical approaches such as Boltzmann inversion. Finally, numerical approaches often require a atomistic reference system, which to some degree defeats the purpose of gaining efficiency with CG. Assuming an informed value of the radius of gyration and the direct correlation function limit as $k \to 0$, called c_0 [103], the IECG potential only requires such reference systems for validation studies.

1.5 Multiscale / Multiresolution Chemistry

One of the grand challenges of modern molecular simulation is bridging length and time scales in physically meaningful and consistent ways [108]. Consider a biomolecular system such as a collection of eukaryotic cells. From a low-level chemical point of view, each cell consists of many different biopolymers such as DNA, proteins, sugars, enzymes, and upwards of 90% liquid water. From a higher-level biological point of view, intercellular communication (between cells) plays an important role in synaptic transmission, hormone secretion, and other

signaling events, even at long distances. Currently, MD simulations are capable of modeling a few biopolymers in a water bath, and accurate QM simulations can reasonably simulate regions with a few dozen atoms. CG methods are capable of simulating much larger regions, but methods normally consist of repeating units of the same atomic or molecular type. Continuum mechanics goes one step further, and models materials as a continuous mass rather than a system of discrete particles. For example, the water within a cell can potentially be modeled with methods in hydrodynamics and fluid mechanics. Furthermore, it is known that biological processes occur over extremely long timescales, whereas MD simulations can reasonably only reach the order of microseconds.

In order to effectively and accurately model and simulate such complex systems, we require tractable and robust methods for bridging the QM, MM, CG, and continuum scales, both spatially and temporally, while still exhibiting the correct behavior at the edges of multiscale regions. This research area is expansive, consists of many different facets, and even contains cross-discipline similarities and analogies between topics in chemistry and physics [109, 110]. However, the following sections limit the scope of mutliscale and multiresolution methods to particle-based computational chemistry, with a particular focus on available software frameworks and methodologies, which are currently somewhat lacking possibly due to the difficulty of the problem.

1.5.1. QM/MM

QM/MM simulations consist of a relatively small region accurately modeled by QM methods, with the remaining regions modeled more efficiently by MM methods. Fig. 17 shows a simple example of a QM/MM hybrid simulation region.

In this example, we presume the inner region is interesting because it contains some chemical reaction, which QM models well. The outer region, on the other hand, may contain no chemical reactions, but we still desire consistent thermodynamics and inclusion of long-range effects on the QM region. There are three types of interactions in this hybrid system: interactions between atoms in the QM region, interactions between atoms in the MM region, and interactions between QM and MM atoms. Sections I.1.2 and I.1.3 of this dissertation described the QM and MM interactions (respectively) in detail, and their evaluation within QM/MM simulations is no different. However, the interactions between the QM and MM regions are more complicated, and researchers have proposed several approaches for handling them.

One simple approach is called *subtractive QM/MM coupling*. The idea is fairly simple, and is represented by this equation:

$$V_{\mathrm{QM/MM}} = V_{\mathrm{MM}}(\mathrm{MM} + \mathrm{QM}) + V_{\mathrm{QM}}(\mathrm{QM}) - V_{\mathrm{MM}}(\mathrm{QM}) \tag{1.11}$$

Here, we claim that the total potential energy of the QM/MM system, $V_{\mathrm{QM/MM}}$, involves adding two terms and subtracting another term. Fig. 18 shows a representative diagram of the subtractive QM/MM coupling method. The first term on the right hand side of Eqn. 1.11 corresponds to the sum of the QM and the MM regions evaluated at the MM level, as shown in the third box from the left of Fig. 18. The second term on the right hand side corresponds to the QM region evaluated at the QM level, as shown in the second box from the left of Fig. 18. Finally, the subtracted term corresponds to the QM region evaluated at the MM level, as shown in the rightmost box of Fig. 18. The most popular implementation of this approach is the ONIOM method [111], which is available in NWChem and

Gaussian. Besides the subtractive QM/MM coupling scheme, there is also additive QM/MM coupling:

$$V_{\mathrm{QM/MM}} = V_{\mathrm{MM}}(\mathrm{MM}) + V_{\mathrm{QM}}(\mathrm{QM}) + V_{\mathrm{QM-MM}}(\mathrm{QM} + \mathrm{MM}) \tag{1.12}$$

in which the third term on the right hand side considers the interactions between QM and MM atoms explicitly. Another popular, but more sophisticated, approach is capping bonds at the QM/MM boundary. This approach cuts bonds crossing the QM/MM region boundary, and replaces them with something like link atoms or localized orbitals [112]. The specifics of this approach bring about complications that are beyond the scope of this dissertation, but are covered elsewhere [113, 114, 112, 109, 110].

Simulation frameworks that support QM/MM through various packages include NWChem, GROMACS, QuanPol, GAMESS (both US and UK version), and AMBER. The method has become so important for computational chemists that the inventors of QM/MM, Warshel and Levitt, won 2013 Nobel Prize in Chemistry [115]. Despite its importance, there is a dearth of research that considers the parallel performance of these workflows. This is puzzling given the fact that it usually relies on the tight coupling of different simulation modules, even between different frameworks [116]. For instance, Fig. 19 shows the overall control flow of a QM/MM simulation. At a glance, it is clear that this directed acyclic graph (DAG) of dependencies offers opportunity to exploit parallelism, but popular QM/MM frameworks do not support such dynamic execution. Future dissertation work could consider the performance of these methods in the context of scientific workflow management and how to efficiently share computational resources in the face of tight application coupling.

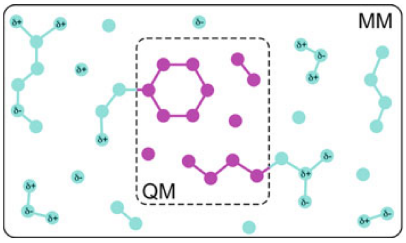


FIGURE 17. QM/MM concept. Diagram is from [114].

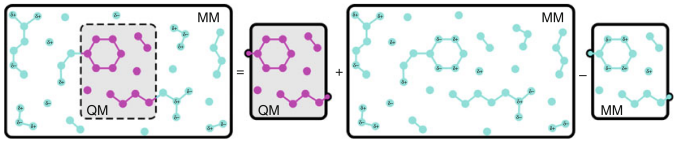


FIGURE 18. QM/MM subtractive coupling. Diagram is from [114].

1.5.2. MM/CG

The MM/CG model is analogous to the QM/MM model, except that MM represents a smaller region more accurately, and CG models the remaining regions more efficiently. Such methods are particularly important for multiscale simulations in which there are large regions with well-verified CG models, such as the solvent regions around active proteins. Interestingly, the first CG model of a globular protein was introduced in a 1975 paper by Warshel and Levitt [117], the recipients of the 2013 Nobel Prize in Chemistry. Even more impressive, this was the first multiscale MM/CG simulation, because side-chains of the protein were treated with atom-level detail. Since then, MM/CG has been explored in much more sophisticated scenarios [118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 13, 79, 129]. Most notably, methods in *Adaptive Resolution Schemes (AdResS)* consider ways to specify regions having different granularities. AdResS is the subject of the following subsection 1.5.2..1.

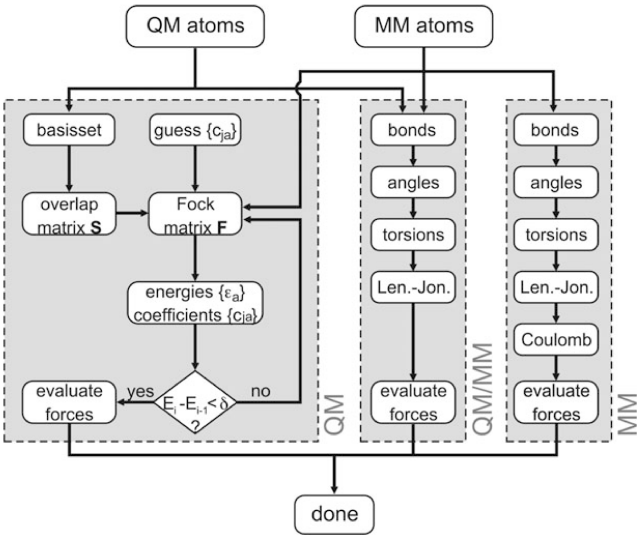


FIGURE 19. QM/MM software flow diagram. Diagram is from [114].

In terms of software implementations, MM/CG is less ubiquitous than QM/MM, but is currently available in GROMACS (however, documentation is limited and support is constrained to certain systems such as those in which only the water solvent is coarse-grained). Other tools, such as the ChemShell environment [116] do apparently allow for flexible MM/CG setups [16], but again, thorough documentation is currently limited to QM/MM procedures. Finally, recent work by Ozog et al. enables switching between MM and CG representations in an automated way [22], which is a considerable step towards AdResS and MM/CG in the LAMMPS framework.

1.5.2..1 Adaptive Resolution (AdResS)

Many QM/MM studies do not consider dynamical systems in which atoms freely move between the QM and MM regions. For MM/CG simulations however, this may be a more important requirement, because we often coarse grain in

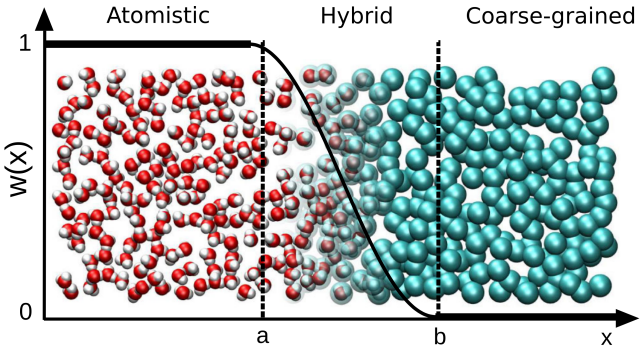


FIGURE 20. A simple AdResS simulation box with the switching function, $w(x)$, overlaid.

order to bridge longer timescales. The adaptive resolution, or *AdResS*, technique addresses the following requirements:

1. Specification of MM and CG regions, analogous to QM/MM from Fig. 17; however, simpler AdResS experiments consider the scenario in Fig. 20.

2. Free exchange of atoms and molecules from the MM region to the CG region and vice versa.

3. Dynamics should occur under thermodynamic equilibrium, i.e., at the same temperature, density, and pressure throughout the simulation box.

L.D. Site and others have led several explorations [108, 118, 119, 120, 130, 123, 126, 128, 13, 129] of an AdResS technique with a hybrid region between MM and CG domains where the derived forces couple in the following manner:

$$\mathbf{F}_{\alpha\beta} = w(R_\alpha)w(R_\beta)\mathbf{F}^{atom}_{\alpha\beta} + [1 - w(R_\alpha)w(R_\beta)]\mathbf{F}^{cm}_{\alpha\beta} \tag{1.13}$$

This equation describes the force between two molecules as a function of their positions (in the x dimension). Here, α and β refer to two molecules, and the positions of their center of mass are X_α and X_β, respectively. $\mathbf{F}^{atom}_{\alpha\beta}$ is the force

derived from the atomistic (MM) potential and $\mathbf{F}_{\alpha\beta}^{cm}$ is the force derived from the CG potential. The function $w(x)$ is a smooth switching function that is 0 in the CG region and 1 in the MM region. With this force description, atoms moving in the hybrid region slowly lose (or gain) degrees of freedom. ESPResSo^{++}, from section 1.3.3..4 and GROMACS+MARTINI use this technique for doing AdResS simulations.

The model from Eqn. 1.13 is quite simple, and it clearly satisfies requirements 1 and 2 from above. However, to satisfy requirement 3, the authors must make non-trivial adjustments. For instance, an external source of heat must be introduced to each degree of freedom to assure that the temperature is constant through the hybrid region [108]. Also, an external force is added to assure thermodynamic equilibrium [131]. Interestingly, the IECG work discussed in Section 1.4.2. does not seem to require such adjustments for different levels of granularity. While Eqn. 1.13 may not be compatible with IECG in its form above, future work should consider whether something like a gradual (or sudden) change in *granularity* (for example, within a polymer melt) can be used for AdResS. Not only might this trivially satisfy requirement 3 without external forces/thermostats, but it is possible computation will be more efficient when using the analytically derived IECG potential.

1.5.2..2 Backmapping

Defining a new representation of the polymer as a chain of soft colloidal particles greatly reduces the amount of information to be collected and controlled, which speeds up the simulation. It is well known that modeling fewer colloidal particles with an appropriate potential decreases the degrees of freedom and

computational requirements by an amount proportional to the granularity [92]. The representation of a polymer as a chain of soft blobs also allows the chains to more easily cross each other, decreasing the required time for the simulation to find the equilibrium structure. However, the information on the molecular local scale needs to be restored at the end of the simulation to account for properties on the UA scale. In a nutshell, it is important to alternate between the CG representation (which speeds up the simulation) and the UA representation (which conserves the local scale information). By quickly switching back and forth from UA to CG, we open doors to new studies of polymeric systems while maintaining simulation accuracy and efficiency. While optimizing the computational performance of CG codes is important, without a way to incorporate UA-level detail, CG efficiency has relatively less value. The goal is to develop an integrated approach for conducting simulations that exploit both CG efficiency and UA accuracy.

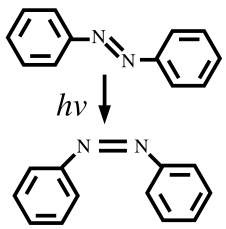
In homopolymer systems, transforming from the UA representation to the CG representation is straightforward: for each subchain having N_b monomers, the new soft sphere coordinate is simply the center of mass of the subchain. On the other hand, the *reverse* procedure of mapping from the CG representation to the UA representation is not generally well-defined. This transformation of a CG model into a UA model is a popular research topic, commonly referred to as the *backmapping* problem (See section 3.2). For our homopolymer system, the backmapping problem is simply stated as follows: given a collection of CG soft sphere chains coordinates, insert monomer chains in such a way that we would recover the original CG configuration if we were to coarse-grain the system again.

It is easy to see that solutions to backmapping problems are not unique, because there are many different UA configurations that could map back to a given

CG configuration. Much backmapping work focuses on biomolecules [132], but relatively little work has explored homopolymers. However, efficient backmapping procedures in polymer simulations are imperative for developing a full-fledged adaptively resolved simulations [133]. Furthermore, backmapping procedures contain clear opportunities for exploiting parallelism, which have yet to be explored.

1.5.3. QM/MM/CG

The availability of QM/MM and MM/CG methods begs the question of whether it is possible and informative to study QM/MM/CG models. It is easy to conjure such a triple-scale workflow that may be of interest. Consider the chemical system shown in Fig. 21, which describes the backmapping loop of a collection of liquid crystals containing azobenzene compounds, which may have potential applications in photo-switching [134]. Upon illumination, the azobenzene molecule can isomerize from a *trans* to a *cis* state:



then possibly decays back to the *trans* state, depending on the immediate neighborhood of other atoms. The first step shown in Fig. 21 involves constructing a CG simulation that represents a collection of azobenzene molecules. Then, similar to the fast equilibration work by Ozog et al. [22] mentioned in Section 1.5.2., we simulate the CG model long enough to reach equilibration. After equilibrating, a small region is chosen to be modeled with QM, and the atomic coordinates are

backmapped using techniques described in Section 1.5.2..2. A QM simulation determines whether the isomerization takes place, then is equilibrated using an MM model, then reinserted into the larger CG domain. This entire process is then repeated if necessary.

The workflow described in the previous paragraph has no implementation, the paper only presents the scenario as a multiscale problem of interest [108]. However, other recent publications suggests that an implementation of a triple-scale model is possible and accurate [16, 17] using the ChemShell interactive environment [116]. In fact, this is the same work used in the visualization from Fig. 1. The capabilities of ChemShell are quite interesting, because they exploit the simplicity of most hybrid methods (such as described in Section 1.5.1.) to support a sizeable collection of QM, MM, and CG supporting simulation tools. These currently include NWChem, GAMESS-UK, DALTON, Molpro, Gaussian, Q-Chem, DL_POLY, CHARMM, GROMOS, and several more. Fig. 22 shows ChemShell's software architecture, which enables modularity of the components at each scale. One disadvantage (to many) is that ChemShell relies on a custom TCL interactive shell. While this does enable the construction of intricate QM/MM/CG workflows, a more modern programming language such as Python may be a better choice because of its available scientific programming libraries, better extendibility, superior error and exception handling, and support for HPC computing.

1.6 Relevant Applications

This section itemizes several example applications that utilize the concepts discussed in this dissertation. In this author's opinion, the most encouraging publication comes from the BASF, which is the largest chemical producer in the

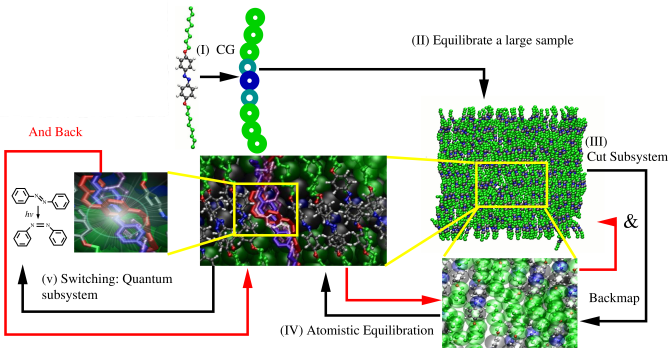


FIGURE 21. An example of a triple-scale workflow. The diagram is from [108].

world [38]. This paper presents applications of quantum chemistry in industry, and highlights the importance of QM/MM methods in designing new materials (CG methods are mentioned as being important as well, but beyond the scope of the paper). In addition, the following publications are relevant to multiscale and multiresolution computational chemistry simulations:

- HF and DFT simulations of Lithium/air batteries using a new 3D FFT implementation [10]
- Divide-and-conquer quantum MD with DFT for hydrogen on-demand [37]
- Multiscale crack modeling (w/ LAMMPS and other tools) using the PERMIX computational library [82]

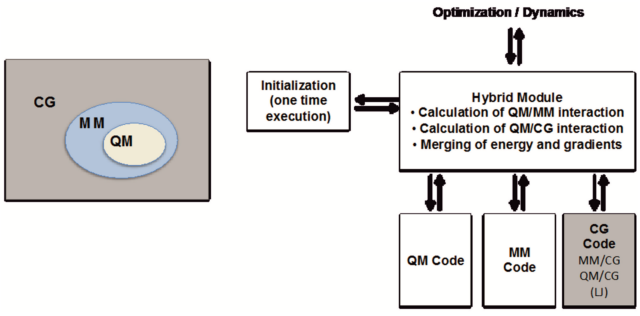


FIGURE 22. (A) Schematic of the QM/MM/CG partitioning. (B) Example of the components of a QM/MM/CG workflow (from the ChemShell code). The schematic is from [16].

- Light-induced phase transitions in liquid crystal containing azobenzene photoswitch (QM/MM/CG) [130]
- Cloud environment for materials simulations [135]
- Screening quantum chemistry databases for organic photovoltaics [72]
- Data mining to aid material design [73].

CHAPTER II

QUANTUM CHEMISTRY: SOFTWARE OPTIMIZATIONS

This chapter contains previously published material with co-authorship. Section 2.1 includes content on Inspector/Executor load balancing from ICPP 2013 [18]. This work was a collaboration with scientists at Argonne National Laboratory. Jeff Hammond, Jim Dinan, Pavan Balaji served as primary advisors and gave me close guidance during all research activities. Dr. Hammond wrote the introduction and background sections of the paper and supervised the software development. I wrote the rest of the paper and gathered all experimental data.

Section 2.2 describes the WorkQ system, first presented at ICPADS 2014 [19]. This work resulted in a Directed Research Project with Prof. Malony and collaborators Dr. Hammond and Dr. Balaji. We had weekly meetings discussing progress and next steps. I wrote the paper and Prof. Malony suggested edits and revisions. I wrote all the relevant code and gathered all experimental data.

Section 2.3 includes content about UPC++ and Hartree-Fock from IPDPS 2016 [20]. This work was a collaboration with scientists at Lawrence Berkeley National Laboratory: Kathy Yelick, Wibe de Jong, Yili Zheng, and Amir Kamil; as well as Prof. Malony and Dr. Hammond. Throughout this summer project, Dr. Yelick, Dr. de Jong, Dr. Zheng, and Dr. Kamil were close advisors and supplied much guidance. I wrote all code associated with the Hartree-Fock port to UPC++ and the progress thread feature to UPC++ itself. Dr. Hargrove contributed the fetch and add network atomic feature to GASNet, and a paragraph about the implementation. Dr. Kamil wrote the background section on UPC++ and the

multi-dimensional array library, and I wrote the rest. Dr. Zheng made several relevant bug fixes and relatively minor contributions to the paper prose.

2.1 Load Balancing Algorithms

2.1.1. Inspector/Executor Introduction

Load balancing of irregular computations is a serious challenge for petascale and beyond because the growing number of processing elements (PEs) – which now exceeds 1 million on systems such as Blue Gene/Q – makes it increasingly more difficult to find a work distribution that keeps all the PEs busy for the same period of time. Additionally, any form of centralized dynamic load balancing, such as master-worker or a shared counter (e.g., Global Arrays' `NXTVAL` [136]), becomes a bottleneck. The competition between the need to extract million-way parallelism from applications and the need to avoid load-balancing strategies that have components which scale with the number of PEs motivates us to develop new methods for scheduling collections of tasks with widely varying cost; the motivating example in this case is the NWChem computational chemistry package. One of the major uses of NWChem is to perform quantum many-body theory methods such as coupled cluster (CC) to either single and double (CCSD) or triple (CCSDT) order accuracy. Popular among chemists are perturbative methods such as CCSD(T) and CCSDT(Q) because of their high accuracy at relatively modest computational cost.[1] In these methods, (T) and (Q) refer to perturbative a posteriori corrections to the energy that are highly scalable (roughly speaking, they resemble MapReduce), while the iterative CCSD and CCSDT steps have

[1] The absolute cost of these methods is substantial when compared with density-functional theory (DFT), for example, but this does not discourage their use when high accuracy is required.

much more communication and load imbalance. Thus, this section focuses on the challenge of load balancing these iterative procedures. However, the algorithms we describe can be applied to noniterative procedures as well.

In this section of the dissertation, we demonstrate that the inspector-executor model (IE) is effective in reducing load imbalance as well as eliminating the overhead from the `NXTVAL` dynamic load balancer. Additionally, we find that IE algorithms are effective when used in conjunction with static partitioning, which is done both with task performance modeling and empirical measurements. We present three different IE load-balancing techniques which each display unique properties when applied to different chemical problems. By examining symmetric (highly sparse) versus nonsymmetric (less sparse) molecular systems in the context of these three methods, we better understand how to open doors to new families of highly adaptable load-balancing algorithms on modern multicore architectures.

2.1.2. NWChem and Coupled-Cluster Background

Chapter I introduced at a high-level the NWChem software framework and the PGAS paradigm on which it depends. The following subsections give a more thorough background of PGAS, NWChem, coupled-cluster methods, the Global Arrays programming model, and the Tensor Contraction Engine. A reader who is familiar with these concepts can safely jump ahead to Section 2.1.3..

2.1.2..1 PGAS

The availability and low cost of commodity hardware components have shaped the evolution of supercomputer design toward distributed-memory architectures. While distributed commodity-based systems have been a boon for

effectively and inexpensively scaling computational applications, they have also made it more difficult for programmers to write efficient parallel programs. This difficulty takes many forms: handling the diversity of architectures, managing load balance, writing scalable parallel algorithms, exploiting data locality of reference, and utilizing asynchronous control, to name a few. A popular parallel programming model that eases the burden on distributed-memory programmers is found in PGAS languages and interfaces.

In the PGAS paradigm, programs are written single program, multiple data (SPMD) style to compute on an abstract global address space. Abstractions are presented such that global data can be manipulated as though it were located in shared memory, when in fact data is logically partitioned across distributed compute nodes with an arbitrary network topology. This arrangement enables productive development of distributed-memory programs that are inherently conducive to exploiting data affinity across threads or processes. Furthermore, when presented with an application programming interface (API) that exposes scalable methods for working with global address space data, computational scientists are empowered to program vast cluster resources without having to worry about optimization, bookkeeping, and portability of relatively simple distributed operations.

Popular PGAS languages/interfaces include UPC, Titanium, Coarray Fortran, Fortress, X10, Chapel, and Global Arrays. The implementation in this work was built on top of Global Arrays/ARMCI, which is the subject of the next section.

2.1.2..2 NWChem

NWChem [137] is the DOE flagship computational chemistry package, which supports most of the widely used methods across a range of accuracy scales (classical molecular dynamics, ab initio molecular dynamics, density-functional theory (DFT), perturbation theory, coupled-cluster theory, etc.) and many of the most popular supercomputing architectures (InfiniBand clusters, Cray XT and XE, and IBM Blue Gene). Among the most popular methods in NWChem are the DFT and CC methods, for which NWChem is one of the few codes (if not the only code) that support these features for massively parallel systems. Given the steep computational cost of CC methods, the scalability of NWChem in this context is extremely important for real science. Many chemical problems related to combustion, energy conversion and storage, catalysis, and molecular spectroscopy are untenable without CC methods on supercomputers. Even when such applications are feasible, the time to solution is substantial; and even small performance improvements have a significant impact when multiplied across hundreds or thousands of nodes.

2.1.2..3 Coupled-Cluster Theory

Coupled-cluster theory [138] is a quantum many-body method that solves an approximate Schrödinger equation resulting from the CC ansatz,

$$|\Psi_{CC}\rangle = \exp(T)|\Psi_0\rangle,$$

where $|\Psi_0\rangle$ is the reference wavefunction (usually a Hartree-Fock Slater determinant) and $\exp(T)$ is the cluster operator that generates excitations out of the reference. Please see Refs. [139, 140] for more information.

A well-known hierarchy of CC methods exists that provides increasing accuracy at increased computational cost [141]:

$$\cdots < CCSD < CCSD(T) < CCSDT$$
$$< CCSDT(Q) < CCSDTQ < \cdots .$$

The simplest CC method that is generally useful is CCSD [142], has a computational cost of $O(N^6)$ and storage cost of $O(N^4)$, where N is a measure of the molecular system size. The "gold standard" CCSD(T) method [143, 144, 145] provides much higher accuracy using $O(N^7)$ computation but without requiring (much) additional storage. CCSD(T) is a very good approximation to the full CCSDT [146, 147] method, which requires $O(N^8)$ computation and $O(N^6)$ storage. The addition of quadruples provides chemical accuracy, albeit at great computational cost. CCSDTQ [148, 149, 150] requires $O(N^{10})$ computation and $O(N^8)$ storage, while the perturbative approximation to quadruples, CCSDT(Q) [151, 152, 153, 154], reduces the computation to $O(N^9)$ and the storage to $O(N^6)$. Such methods have recently been called the "platinum standard" because of their unique role as a benchmarking method that is significantly more accurate than CCSD(T) [155].

An essential aspect of an efficient implementation of any variant of CC is the exploiting of symmetries, which has the potential to reduce the computational cost and storage required by orders of magnitude. Two types of symmetry exist

in molecular CC: spin symmetry [156] and point-group symmetry [157]. Spin symmetry arises from quantum mechanics. When the spin state of a molecule is a singlet, some of the amplitudes are identical; and thus we need store and compute only the unique set of them. The impact is roughly that N is reduced to $N/2$ in the cost model, which implies a reduction of one to two orders of magnitude in CCSD, CCSDT, and CCSDTQ. Point-group symmetry arise from the spatial orientation of the atoms. For example, a molecule such as benzene has the symmetry of a hexagon, which includes multiple reflection and rotation symmetries. These issues are discussed in detail in Refs. [158, 159]. The implementation of degenerate group symmetry in CC is difficult; and NWChem, like most codes, does not support it. Hence, CC calculations cannot exploit more than the 8-fold symmetry of the D_{2h} group, but this is still a substantial reduction in computational cost.

While the exploitation of symmetries can substantially reduce the computational cost and storage requirements of CC, these methods also introduce complexity in the implementation. Instead of performing dense tensor contractions on rectangular multidimensional arrays, point-group symmetries lead to block diagonal structure, while spin symmetries lead to symmetric blocks where only the upper or lower triangle is unique. This is one reason that one cannot, in general, directly map CC to dense linear algebra libraries. Instead, block-sparse tensor contractions are mapped to BLAS at the PE level, leading to load imbalance and irregular communication between PEs. *Ameliorating the irregularity arising from symmetries in tensor contractions is one of the major goals of this work.*

2.1.2..4 Global Arrays

Global Arrays (GA) [160, 161] is a PGAS-like global-view programming model that provides the user with a clean abstraction for distributed multidimensional arrays with one-sided access (Put, Get, and Accumulate). GA provides numerous additional functionalities for matrices and vectors, but these are not used for tensor contractions because of the nonrectangular nature of these objects in the context of CC. The centralized dynamic load balancer `NXTVAL` was inherited from TCGMSG [136], a pre-MPI communication library. Initially, the global shared counter was implemented by a polling process spawned by the last PE, but now it uses ARMCI remote fetch-and-add, which goes through the ARMCI communication helper thread [162]. Together, the communication primitives of GA and `NXTVAL` can be used in a template for-loop code that is general and can handle load imbalance, at least until such operations overwhelm the computation because of work starvation or communication bottlenecks that emerge at scale. A simple variant of the GA "get-compute-update" template is shown in Alg. 4. For computations that are naturally load balanced, one can use the GA primitives and skip the calls to `NXTVAL`, a key feature when locality optimizations are important, since `NXTVAL` has no ability to schedule tasks with affinity to their input or output data. This is one of the major downsides of many types of dynamic load-balancing methods—they lack the ability to exploit locality in the same way that static schemes do.

While there exist several strategies for dynamically assigning collections of tasks to processor cores, most present a trade-off between the quality of the load balance and scaling to a large number of processors. Centralized load balancers can be effective at producing evenly distributed tasks, but they can have substantial

Algorithm 4 The canonical Global Arrays programming template for dynamic load balancing. `NXTVAL()` assigns each loop iteration number to a process that acquires the underlying lock and atomically increments a global counter. This counter is located in memory on a single node, potentially leading to considerable network communication. One can easily generalize this template to multidimensional arrays, multiple loops, and blocks of data, rather than single elements. As long as the time spent in `FOO` is greater than that spent in `NXTVAL`, `Get`, and `Update`, this is a scalable algorithm.

```
Global Arrays: A, B
Local Buffers: a, b
count = 1
next = NXTVAL()
for i = 1 : N do
  if ( next == count ) then
    Get A(i) into a
    b = FOO(a)
    Update B(i) with b
    next = NXTVAL()
  end if
  count = count + 1
end for
```

overhead. Decentralized alternatives such as work stealing [163, 164] may not achieve the same degree of load balance, but their distributed nature can reduce the overhead substantially.

2.1.2..5 Tensor Contraction Engine

The Tensor Contraction Engine (TCE) [165, 166] is a project to automate the derivation and parallelization of quantum many-body methods such as CC. As a result of this project, the first parallel implementations of numerous methods were created and applied to larger scientific problems than previously possible. The original implementation in NWChem by Hirata was general (i.e., lacked optimizations for known special cases) and required significant tuning to scale CC to more than 1,000 processes in NWChem [167]. The tuning applied to the

TCE addressed essentially all aspects of the code, including more compact data representations (spin-free integrals are antisymmetrized on the fly in the standard case), reduction in communication by applying additional fusion that the TCE compiler was not capable of applying. In some cases, dynamic load balancing (DLB) was eliminated altogether when the cost of a diagram (a single term in the CC eqns.) was insignificant at scale and DLB was unnecessary overhead.

The TCE code generator uses a stencil that is a straightforward generalization of Alg. 5. For a term such as

$$Z(i,j,k,a,b,c) += \sum_{d,e} X(i,j,d,e) * Y(d,e,k,a,b,c), \tag{2.1}$$

which is a bottleneck in the solution of the CCSDT equations, the data is tiled over all the dimensions for each array and distributed across the machine in a one-dimensional global array. Multidimensional global arrays are not useful here because they do not support block sparsity or index permutation symmetries. Remote access is implemented by using a lookup table for each tile and a GA `Get` operation. The global data layout is not always appropriate for the local computation, however; therefore, immediately after the `Get` operation completes, the data is rearranged into the appropriate layout for the computation. Alg. 5 gives an overview of a distributed tensor contraction in TCE. For compactness of notation, the `Fetch` operation combines the remote `Get` and local rearrangement. The `Symm` function is a condensation of a number of logical tests in the code that determine whether a particular tile will be nonzero. These tests consider the indices of the tile and not any indices within the tile because each tile is grouped such that the symmetry properties of all its constitutive elements are identical. In Alg. 5,

Algorithm 5 Pseudocode for the default TCE implementation of Eq. 2.1. For clarity, aspects of the Alg. 4 DLB template are omitted.

```
Tiled Global Arrays: X, Y, Z
Local Buffers: x, y, z
for all i, j, k ∈ Otiles do
  for all a, b, c ∈ Vtiles do
    if NXTVAL() == count then
      if Symm(i, j, k, a, b, c) == True then
        Allocate z for Z(i, j, k, a, b, c) tile
        for all d, e ∈ Vtiles do
          if Symm(i, j, d, e) == True then
            if Symm(d, e, k, a, b, c) == True then
              Fetch X(i, j, d, e) into x
              Fetch Y(d, e, k, a, b, c) into y
              Contract z(i, j, k, a, b, c)+=
                       x(i, j, d, e)*y(d, e, k, a, b, c)
            end if
          end if
        end for
        Accumulate z into Z(i, j, k, a, b, c)
      end if
    end if
  end for
end for
```

the indices given for the local buffer contraction are the tile indices, but these are merely to provide the ordering explicitly. Each tile index represents a set of contiguous indices so the contraction is between multidimensional arrays, not single elements. However, one can think of the local operation as the dot product of two tiles (*Otile* and *Vtile* in Algs. 5, 6, and 7).

TCE reduces the contraction of two high-dimensional tensors into a summation of the product of several 2D arrays. Therefore, the performance of the underlying BLAS library strongly influences the overall performance of TCE. For the purposes of this work, each tensor contraction routine can be thought of as a global task pool of tile-level 2D DGEMM (double-precision general matrix-matrix

multiplication) operations. This pool of work items is processed according to the following execution model:

1. A unique work item ID is dynamically assigned via an atomic read-modify-write operation to a dynamic load balancing counter (see [18] for details).
2. The global addresses of two tiles (A and B) in the global array space is determined (TCE hash lookup).
3. The corresponding data is copied to the local process space (via one-sided RMA) with `GA_Get`() calls.
4. A contraction is formed between the local copies of tiles A and B and stored into C. When necessary, a permute-DGEMM-permute pattern is performed in order to arrange the indices of the tensor tiles to align with the format of matrix-matrix multiplication.
5. Steps 2, 3, and 4 repeat over the work-item tile bundle; then C is accumulated (`GA_acc()` call) into a separate global array at the appropriate location.

Although this algorithm is specific to CC, we note we that it falls under a more general `get/compute/put` model that is common to many computational applications. For example, the problem of numerically solving PDEs on domains distributed across memory spaces certainly falls under this category.

Section 2.1.3. will motivate a scheme that reduces the load on the centralized `NXTVAL` mechanism within the TCE. Later, Section 2.2.3. will provide motivation for the development of an alternative execution model that is able to perform the `get/compute/put` computation more efficiently by overlapping communication and computation.

2.1.3. Inspector/Executor Motivation and Design

In this section we discuss our motivation for developing inspector-executor (IE) algorithms, the implementation of the IE in the TCE-CC module of NWChem, and the design of our cost partitioning strategy. We begin by characterizing the function of a simple version of the inspector, then augment the IE model by incorporating performance models of the dominant computational kernels. The performance models provide estimations of task execution time to be fed into the static partitioner, then to the executor. While the IE design and implementation is described within the context of NWChem's TCE-CC module, the methods can be applied to any irregular application where reasonably accurate estimation of kernel execution times is possible, either analytically or empirically. In our NWChem implementation, the inspector initially assigns cost estimations by applying performance models of computational kernels for the first iteration, then by doing in situ execution time measurements of tasks for subsequent iterations.

2.1.3..1 Inspector/Executor

When using Alg. 4 in TCE-based CC simulations, the inherently large number of computational tasks (typically hundreds of thousands) each require a call to `NXTVAL` for dynamic load balancing. Very large systems can potentially require many billions of fine-grained tasks, but the granularity can be controlled by increasing the tile size. Although `NXTVAL` overhead can be limited by increasing the tile size, it is far more difficult to balance such a coarse-grained task load while preventing starvation, so typically a large number of small-sized tasks is desirable. However, the average time per call to `NXTVAL` increases with the number

Algorithm 6 Pseudocode for the inspector used to implement Eq. 2.1, simple version.

```
for all i, j, k ∈ Otiles do
  for all a, b, c ∈ Vtiles do
    if Symm(i, j, k, a, b, c) == True then
      Add Task (i, j, k, a, b, c) to TaskList
    end if
  end for
end for
```

of processes (as shown below), so too many tasks is detrimental for strong scaling. This is the primary motivation for implementing the IE.

This increase in time per call to `NXTVAL` is primarily caused by contention on the memory location of the counter, which performs atomic read-modify-write (RMW) operations (in this case the addition of 1) using a mutex lock. For a given number of total incrementations (i.e., a given number of tasks), when more processes do simultaneous RMWs, on average they must wait longer to access to the mutex. This effect is clearly displayed in a flood-test microbenchmark (Fig. 24) where a collection of processes calls `NXTVAL` several times (without doing any other computation). In this test, only off-node processes are allowed to increment the counter (via a call to `ARMCI_Rmw`); otherwise, the on-node processes would be able to exploit the far more efficient shared-memory incrementation, which occurs on the order of several nanoseconds.The average execution time per call to `NXTVAL` always increases as more processes are added.

The increasing overhead of scaling with `NXTVAL` is also directly seen in performance profiles of the tensor contraction routines in NWChem. For instance, Fig. 23 shows a profile of the mean inclusive time for the dominant methods in a CC simulation of a water cluster with 10 molecules. The time spent within `NXTVAL` accounts for about 37% of the entire simulation. We propose an alternate algorithm

Algorithm 7 Pseudocode for the inspector used to implement Eq. 2.1, with cost estimation and static partitioning.

```
for all i, j, k ∈ Otiles do
  for all a, b, c ∈ Vtiles do
    if Symm(i, j, k, a, b, c) == True then
      Add Task (i, j, k, a, b, c, d, e, w) to TaskList
      cost_w = SORT4_performance_model_estm(sizes)
      for all d, e ∈ Vtiles do
        if Symm(i, j, d, e) == True then
          if Symm(d, e, k, a, b, c) == True then
            cost_w = cost_w + ...
            SORT4_performance_model_estm(sizes)
            cost_w = cost_w + ...
            DGEMM_performance_model_estm(m, n, k)
            Compute various SORT4 costs
          end if
        end if
      end for
    end if
  end for
end for
myTaskList = Static_Partition(TaskList)
```

which is designed to reduce this overhead by first gathering task information, then evenly assigning tasks to PEs, and finally executing the computations.

We collect relevant tensor contraction task information by breaking the problem into two major components: inspection and execution (similar to Refs. [169, 170, 171]). In its simplest form, the inspector agent loops through relevant components of the parallelized section and collates tasks (Alg. 6). This phase is limited to computationally inexpensive arithmetic operations and conditionals that classify and characterize tasks. Specifically, the first conditional of any particular tensor contraction routine in NWChem evaluates spin and point-group symmetries to determine whether a tile of the tensor contraction has a nonvanishing element [165], as introduced in section 2.1.2..3. Further along, in

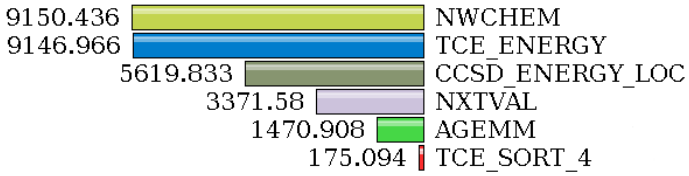


FIGURE 23. Average inclusive-time (in seconds) profile of a 14-water monomer CCSD simulation with the aug-cc-PVDZ basis for 861 MPI processes across 123 Fusion nodes connected by InfiniBand. The `NXTVAL` routine consumes 37% of the entire computation. This profile was made using TAU [168]; for clarity, some subroutines were removed.

a nested loop over common indices, another conditional tests for nonzero tiles of a contraction operand by spin and spatial symmetry. While the inspector's primary purpose is to a create an informative list of tasks to help accomplish load balance during the executor phase, it also has this advantage of revealing sparsity information by applying spin and spatial symmetry arguments before designating a particular task.

While the following section describes a more complicated version of the inspector, the executor is the same in both cases. The pseudocode for the executor is shown in Alg. 8, where tasks gathered in the inspection phase are simply looped over. The executor contains the inner loop of the default TCE implementation with another set of symmetry conditionals, which are found to always return true for the cases considered in this work (we cannot exclude the possibility that they may be false in some other cases). Because the inner loop is dense in the sense that no additional sparsity is achieved at that point, it is natural to aggregate these computations into a single task in order to reduce the number of calls to Accumulate, which is more expensive than either Put or Get because it requires a remote computation, not just data movement (which might be done in entirely in hardware with RDMA). Combining all the inner loop computations in a single

Algorithm 8 Pseudocode for the executor used to implement Eq. 2.1.

```
for all Task ∈ Tasklist do
  Extract (i, j, k, a, b, c) from Task
  Allocate local buffer for Z(i, j, k, a, b, c) tile
  if Symm(i, j, d, e) == True then
    if Symm(d, e, k, a, b, c) == True then
      Fetch X(i, j, d, e) tile into local buffer
      Fetch Y(d, e, k, a, b, c) tile into local buffer
      Z(i, j, k, a, b, c) += X(i, j, d, e)*Y(d, e, k, a, b, c)
      Accumulate Z(i, j, k, a, b, c) buffer into global Z
    end if
  end if
end for
```

task also has the effect of implying reuse of the output buffer, which is beneficial if it fits in cache or computation is performed on an attached coprocessor (while heterogeneous algorithms are not part of this work, they are a natural extension of it).

2.1.3..2 Task Cost Characterization

The average time per call to `NXTVAL` increases with the number of participating PEs, but the IE algorithm as presented so far will improve strong scaling only as much as the proportion of tasks eliminated by the spin and spatial symmetry argument. DLB with `NXTVAL` for large non-symmetric molecular systems (biomolecules usually lack symmetry) will still be plagued by high overhead due to contention on the global counter despite our simple inspection. In this section we further develop the IE model with the intent to eliminate *all* `NXTVAL` calls from the entire CC module (pseudocode shown in Alg. 7).

By counting the number of FLOPS for a particular tensor contraction (Fig. 25), we see that a great deal of load imbalance is inherent in the overall computation. The centralized dynamic global counter does an acceptable job of

handling this imbalance by atomically providing exclusive task IDs to processes that request work. To effectively eradicate the centralization, we first need to estimate the cost of all tasks, then schedule the tasks so that each processor is equally loaded.

In the tensor contraction routines, parallel tile-level matrix multiplications and ordering operations execute locally within the memory space of each processor. The kernels that consume the most time doing such computations are the `DGEMM` and `SORT4` subroutines. The key communication routines are the Global Arrays get (`ga_get`) and accumulate (`ga_acc`) methods, which consume relatively little time for sizeable and accurate simulations of interest. For this reason we use performance model-based cost estimations for `DGEMM` and `SORT4` to partition the task load of the first iteration of each tensor contraction (Alg. 7). For each call to `DGEMM` or `SORT4`, the model estimates the time to execute, and accrues it into $cost_w$. Details regarding the specific performance models used is beyond the scope of this work, but others have explored the development of models for such BLAS kernels [172]. During the first iteration of TCE-CC, we measure the time of each task's entire computation (in the executor phase) to capture the costs of communication along with the computation. This new measurement serves as the cost which is fed into the static partitioning phase for subsequent iterations.

2.1.3..3 Static Partitioning

In our IE implementation, the inspector applies the `DGEMM` and `SORT4` performance models to each tile encountered, thereby assigning a cost estimation to each task of the tensor contractions for the first iteration. Costs for subsequent iterations are based on online measurements of the each task's entire execution

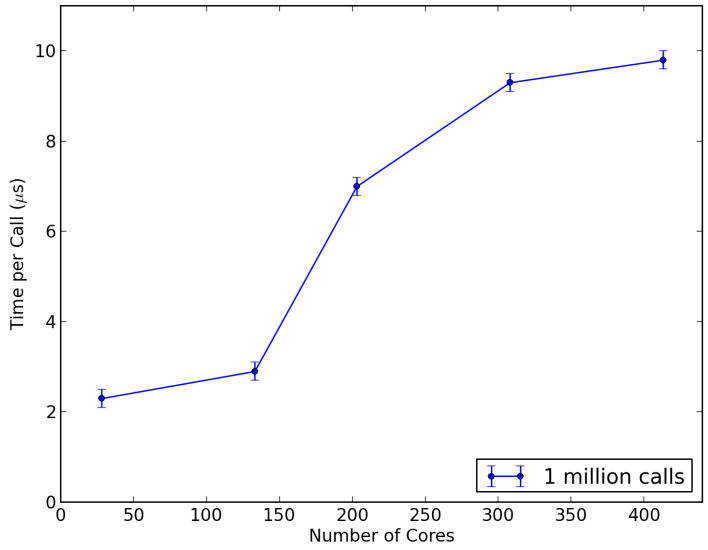


FIGURE 24. Flood benchmark showing the execution time per call to `NXTVAL` for 1 million simultaneous calls. The process hosting the counter is being flooded with messages, so when the arrival rate exceeds the processing rate, buffer space runs out and the process hosting the counter must utilize flow control. The performance gap from 150 to 200 cores is due to this effect occurring at the process hosting the `NXTVAL` counter.

time, which includes communication. In both cases, the collection of weighted tasks constitutes a static partitioning problem which must be solved. The goal is to collect bundles of tasks (partitions) and assign them to processors in such a way that computational load imbalance is minimized. In general, solving this problem optimally is NP-hard [173], so there is a trade-off between computing an ideal assignment of task partitions and the overhead required to do so. Therefore, our design defers such decisions to a partitioning library (in our case, Zoltan [174]), which gives us the freedom to experiment with load-balancing parameters (such as the balance tolerance threshold) and their effects on the performance of the CC tensor contraction routines.

Currently we employ static block partitioning, which intelligently assigns "blocks" (or consecutive lists) of tasks to processors based on their associated

weights (no geometry or connectivity information is incorporated, as in graph/hypergraph partitioning). However, incorporating task connectivity in terms of data locality has been shown to be a viable means of minimizing data access costs [175]. Our technique focuses on accurately balancing the computational costs of large task groups as opposed to exploiting their connectivity, which also matters a great deal at scale. Fortunately, our approach is easily extendible to include such data-locality optimizations by solving the partition problem in terms of making ideal cuts in a hypergraph representation of the task-data system (see Section 2.1.5.). An application making use of the IE static partitioning technique may partition tasks based on any partitioning algorithm.

2.1.3..4 Dynamic Buckets

When conducting static partitioning, variation in task execution times is undesirable because it leads to load imbalance and starvation of PEs. This effect is particularly noticeable when running short tasks where system noise can potentially counteract execution time estimations and lead to a poor static partitioning assignment. Furthermore, even when performance models are acceptably accurate, they generally require determination of architecture-specific parameters found via off-line measurement and analysis.

A reasonable remedy to these problems is a scheme we call dynamic buckets (Fig. 26), where instead of partitioning the task collection across all PEs, we partition across groups of PEs. Each group will contain an instance of a dynamic `NXTVAL` counter. When groups of PEs execute tasks, the imbalance due to dynamic variation is amortized since unbiased variation will lead to significant cancellation. Also, if the groups are chosen such that each counter is resident on a local compute

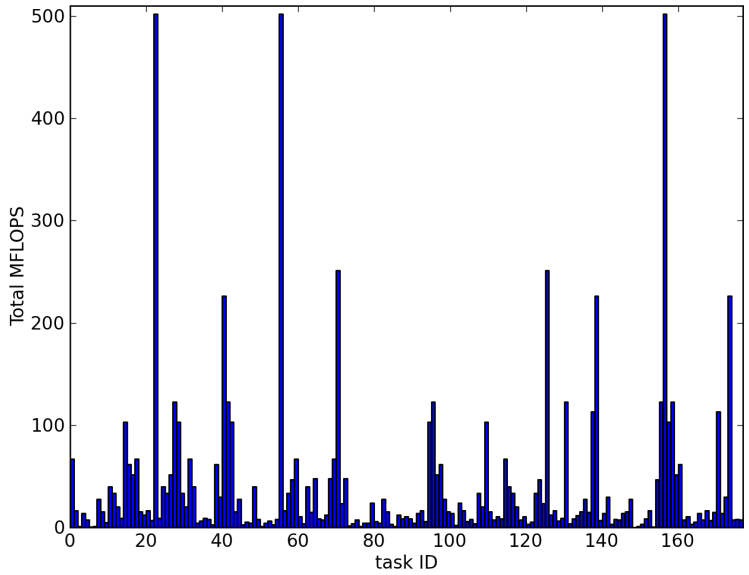


FIGURE 25. Total MFLOPS for each task in a single CCSD T_2 tensor contraction for a water monomer simulation. This is a good overall indicator of load imbalance for this particular tensor contraction. Note that each task is independent of the others in this application.

node relative to the PE group, then `NXTVAL` can work within shared memory, for which performance is considerably better. The other motivation for choosing the execution group to be the node is that contention for the NIC and memory bandwidth in multicore systems is very difficulty to model (i.e. predict) in a complicated application like NWChem, hence we hope to observe a reasonable amount of cancellation of this noise if we group the processes that share the same resources. The idea is that the node-level resources are mostly fixed and that noise will average out since the slowdown in one process due to another's utilization of the NIC will cancel more than the noise between processes on different nodes, since there is no correlation between NIC contention in the latter case. Finally, with a more coarse granularity of task groups, it is feasible that load balance would be acceptable even with a round-robin assignment of tasks to groups (i.e. without

performance model based task estimation) because of the adaptability inherent to having several dynamic counters.

When using the dynamic buckets approach, tasks are partitioned by applying the Longest Processing Time algorithm [176], which unlike block partitioning, is provably a 4/3 approximation algorithm (meaning it is guaranteed to produce a solution within ratio 4/3 of a true optimum assignment). First, tasks are sorted by execution time estimation in descending order using a parallel quicksort. Then, each task with the longest execution time estimation is assigned to the least loaded PE until all tasks are assigned. To increase the efficiency of the assignment step, the task groups are arranged in a binary minimum heap data structure where nodes correspond to groups. Tasks can be added to this minimum heap in $O(\log n)$ time (on average) where n is the number of task groups.

The dynamic buckets design in Fig. 26 also captures elements of topology awareness, iterative refinement, and work stealing. The results in Section 2.1.4. are based on an implementation with node-level topology awareness and a single iteration of refinement based on empirically measured execution times. We refer the reader to other works [163, 164] for information on work stealing implementations.

2.1.4. NWChem with Inspector/Executor Experimental Results

This section provides experimental performance results of several experiments on Fusion, an InfiniBand cluster at Argonne National Laboratory. Each node has 36 GB of RAM and two quad-core Intel Xeon Nehalem processors running at 2.53 GHz. Both the processor and network architecture are appropriate for this study because NWChem performs very efficiently on multicore x86 processors and InfiniBand networks. The system is running Linux kernel 2.6.18 (x86_64).

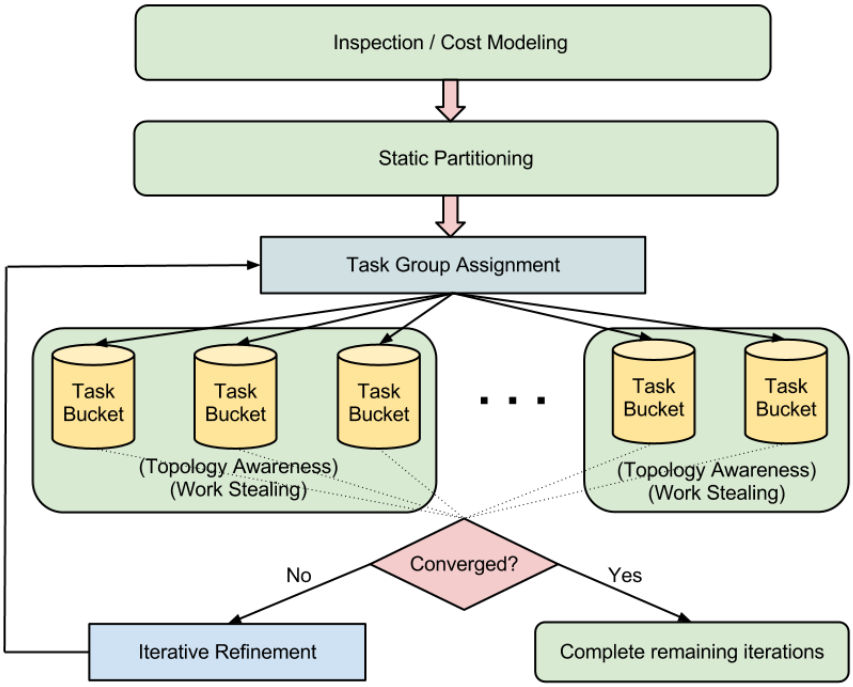


FIGURE 26. Inspector/Executor with Dynamic Buckets.

NWChem was compiled with GCC 4.4.6, which was previously found to be just as fast as Intel 11.1 because of the heavy reliance on BLAS for floating-point-intensive kernels, for which we employ GotoBLAS2 1.13. The high-performance interconnect is InfiniBand QDR with a theoretical throughput of 4 GB/s per link and 2 μs latency. The communication libraries used were ARMCI from Global Arrays 5.1, which is heavily optimized for InfiniBand, and MVAPICH2 1.7 (NWChem uses MPI sparingly in the TCE). Fusion is an 8 core-per-node system, but ARMCI requires a dedicated core for optimal performance [177]. We therefore launch all NWChem experiments with 7 MPI processes per node, but reserve all 8 cores using Fusion's job scheduler and resource manager. Because the application is utilizing 8 cores per node, results are reported in multiples of 8 in Figs. 27, 29, and 30.

First we present an analysis of the strong scaling effects of using `NXTVAL`. Then we describe experiments comparing the original NWChem code with two versions of inspector/executor: one, called I/E NXTVAL, that merely eliminates the extraneous calls to `NXTVAL`, and one that eliminates all calls to `NXTVAL` in certain methods by using the performance model to estimate costs and Zoltan to assign tasks statically. Because the second technique incorporates both dynamic load balancing and static partitioning, we call it I/E Hybrid. Finally, we show the improvement of the I/E Dynamic Buckets approach for a simulation where I/E Hybrid cannot overcome the effects from variation in task execution time due to system noise.

2.1.4..1 Scalability of centralized load-balancing

The scalability of centralized DLB with `NXTVAL` in the context of CC tensor contractions in NWChem was evaluated by measuring the percentage of time

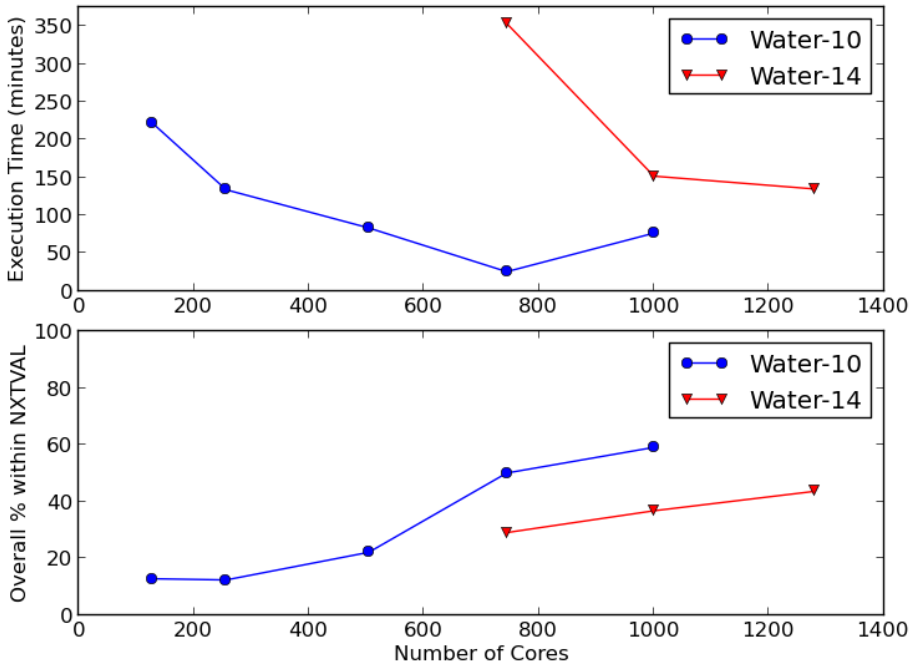


FIGURE 27. Total percentage of execution time spent in `NXTVAL` for a 10-H_2O CCSD simulation (15 iterations) with the aug-cc-pVDZ basis running on the Fusion cluster (without IE). The 14-H_2O test will not fit in global memory on 63 nodes (8 cores per node = 504 cores) or fewer. These data points were extracted from mean inclusive-time profiles as in Fig. 23.

spent incrementing the counter (averaged over all processes) in two water cluster simulations. The first simulation (blue curve in Fig. 27) is a simulation of 10-water molecules using the aug-cc-pVDZ basis, and the second simulation (red curve) is the same but with 14-water molecules. The percentages are extracted from TAU profiles of the entire simulation run, with the inclusive time spent in `NXTVAL` divided by the inclusive time spent in the application.

Fig. 27 shows that the percentage of time spent in `NXTVAL` always increases as more processors are added to the simulation. This increase is partly because of a decrease in computation per processor, but also because of contention for the shared counter, as displayed in Fig. 24. For 10-water molecules, `NXTVAL` eventually consumes about 60% of the overall application time as we approach 1,000 processes.

TABLE 1. Summary of the performance experiments

	N_2	Benzene	10-H_2O	14-H_2O
Simulation type	CCSDT	CCSD	CCSD	CCSD
# of tasks*	261,120	14,280	2,100	4,060
Ave. data size**	7,418	94,674	2.2 mil.	2.7 mil.
Scale limit (cores)	200	320	750	1,200

*from the largest tensor contraction
**in terms of `DGEMM` input, $mk + kn$

In the larger 14-water simulation, `NXTVAL` consumes only about 30% of the time with 1,000 processes, because of the increase in computation per process relative to the 10-water simulation. The 14-water simulation failed on 504 cores (as seen in Fig. 27) because of insufficient global memory.

2.1.4..2 Inspector/Executor DLB

Table 1 summarizes the NWChem experiments we performed in terms of their task load in the largest tensor contraction of the simulation. Figure 28 compares four different variants to clarify how performance improvements arise with the I/E algorithm. CC simulations fall into two broad categories, symmetrically sparse and dense (i.e., a benzene molecule versus an asymmetric water cluster). We found that problems falling in the sparse category are suitable for the I/E NXTVAL method because they have a large number of extraneous tasks to be eliminated. While the water cluster systems can potentially eliminate a similar percentage of tasks, their relatively larger average task size results in DGEMM dominating the computation. The differences in task loads between these problems necessitate different I/E methods for optimal performance, as shown below in Figs. 29 and 30.

Applying the I/E NXTVAL model to a benzene monomer with the aug-cc-pVTZ basis in the CCSD module results in as much as 33% faster execution of

TABLE 2. Inspector/Executor 300-node performance

Processes	2400
Nodes	300
I/E NXTVAL	498.3 s
I/E Hybrid	483.6 s
Original	-

code compared with the original (Fig. 29). The I/E NXTVAL version consistently performs about 25-30% faster for benzene CCSD. Table 2 shows a single measurement of much larger 300 compute node run. With such a large number of processes, the original code consistently fails on the Fusion InfiniBand cluster with an error in `armci_send_data_to_client()`, whereas the I/E NXTVAL version continues to scale to well beyond 400 processes. This suggests that the error is triggered by an extremely busy `NXTVAL` server.

2.1.4..3 Static Partition

The I/E Hybrid version applies complete static partitioning using the performance model cost estimation technique to long-running tensor contractions which are experimentally observed to outperform the I/E NXTVAL version. Fig. 29 shows that this method always executes in less time than both the original code and the simpler I/E NXTVAL version. Though it is not explicitly proven by any of the figures, this version of the code also appears to be capable of executing at any number of processes on the Fusion cluster, whereas the I/E NXTVAL and original code eventually trigger the ARMCI error mentioned in the previous section.

Unfortunately, it is a difficult feat to transform the machine-generated tensor contraction methods from within the TCE generator, so we have taken a top-down approach where the generated source is changed manually. Because there are over

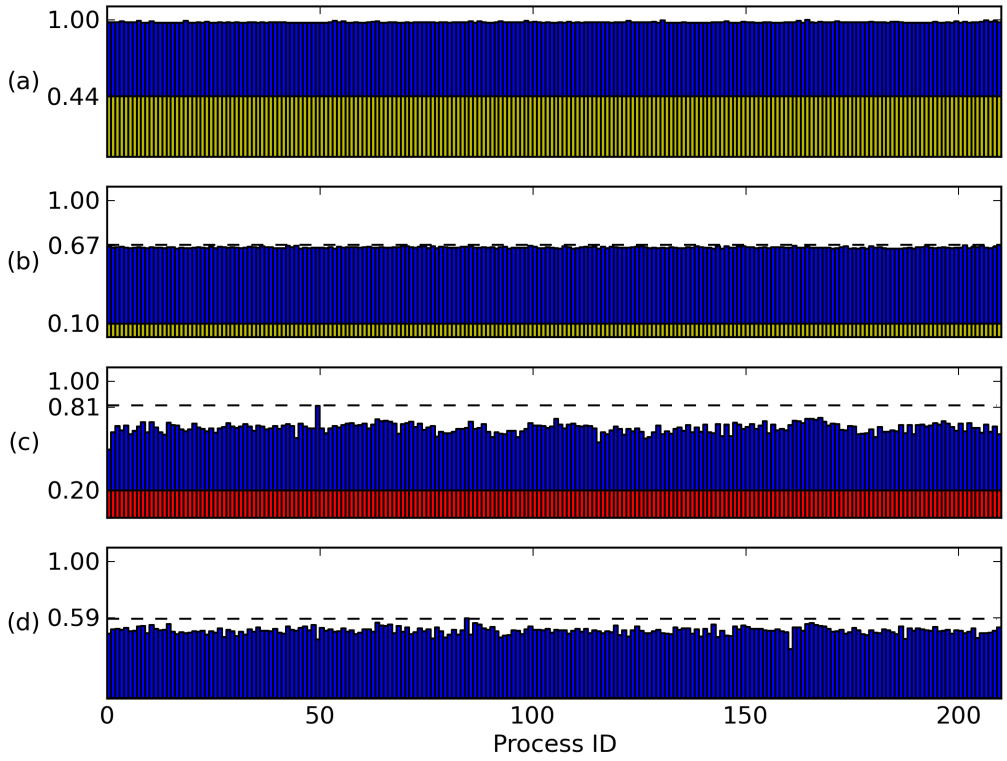


FIGURE 28. Comparative load balance of a tensor contraction for benzene CCSD on 210 processes: (a) Original code with total time in `NXTVAL` overlapped in yellow (all values are normalized to this maximum execution time). (b) I/E with superfluous calls to `NXTVAL` eliminated. (c) First iteration of I/E with performance modeling and static partitioning (overhead time shown in red). (d) Subsequent iterations of I/E static (with zero overhead and iterative refinement). Despite the increase in load variation in (d), the overall time is reduced by 8% relative to (b).

70 individual tensor contraction routines in the CCSDT module and only 30 in the CCSD module, we currently have I/E Hybrid code implemented only for CCSD.

2.1.4..4 Dynamic Buckets

I/E Dynamic Buckets (I/E-DB) is usually the method with the best performance, as seen in Fig. 30. This plot shows the two most time consuming tensor contractions in a 10-H_2O system. In this problem, I/E NXTVAL performs no better than the original code because of relatively less sparsity and larger

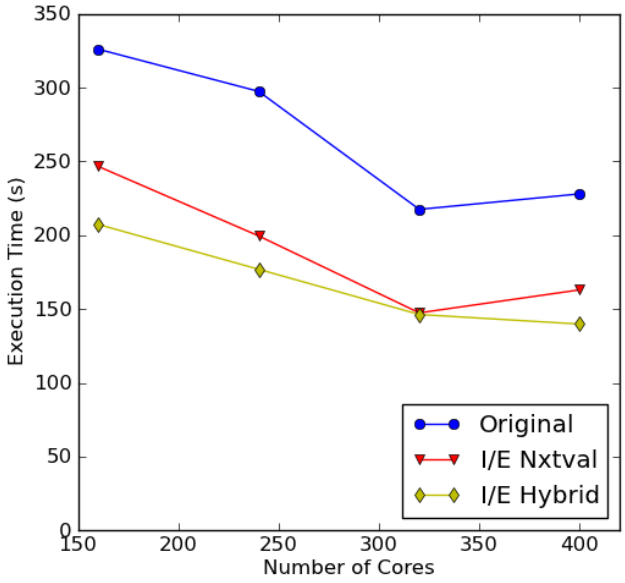


FIGURE 29. Benzene aug-cc-pVQZ I/E comparison for a CCSD simulation.

task sizes in the overall computation. I/E Hybrid (not shown) performs slightly worse than the original code. As explained in section 2.1.3..4, this is due to error in the task execution time estimations. The I/E-DB technique shows up to 16% improvement over IE-NXTVAL due to better load balance when dynamic counters manage groups of tasks.

2.1.5. Related Work in QM Load Balancing

Alexeev and coworkers have applied novel static load balancing techniques to the fragment molecular orbital (FMO) method [178]. FMO differs in computational structure from iterative CC, but the challenge of load balancing is similar, and their techniques parallel the IE cost estimation model. The FMO system is first split into fragments that are assigned to groups of CPU cores. The size of those groups is chosen based on the solution of an optimization problem, with three major terms representing time that is linearly scalable, nonlinearly scalable, and nonparallel.

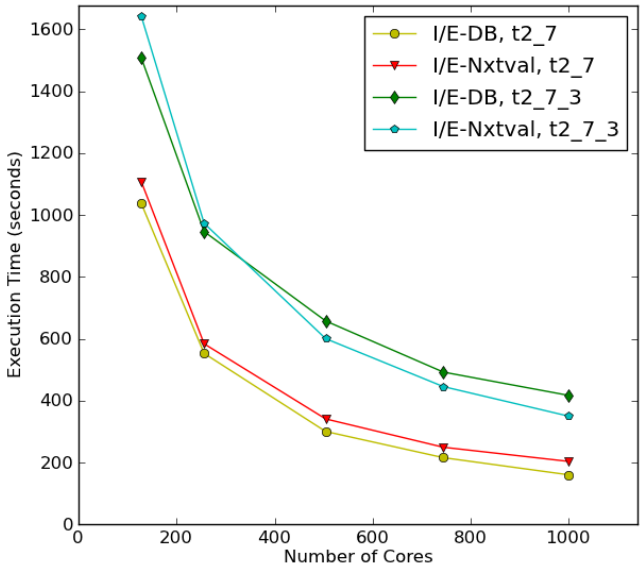


FIGURE 30. Comparison of I/E NXTVAL with I/E Dynamic Buckets for the two most time consuming tensor contractions during a 10-H_2O simulation, `t2_7` and `t2_7_3`. The execution time of the original code is not shown because it overlaps the performance of I/E NXTVAL.

Hypergraph partitioning was used by Krishnamoorthy and coworkers to schedule tasks originating from tensor contractions [175]. Their techniques optimize static partitioning based on common data elements between tasks. Such relationships are represented as a hypergraph, where nodes correspond to tasks, and hyperedges (or sets of nodes) correspond to common data blocks the tasks share. The goal is to optimize a partitioning of the graph based on node and edge weights. Their hypergraph cut optimizes load balance based on data element size and total number of operations, but such research lacks a thorough model for representing task weights, which the IE cost estimation model accomplishes.

The Cyclops Tensor Framework [179, 55] implements CC using arbitrary-order tensor contractions which are implemented by using a different approach from NWChem. Tensor contractions are split into redistribution and contraction phases, where the former permutes the dimensions such that the latter can be done by

using a matrix-matrix multiplication algorithm such as SUMMA [180]. Because CTF uses a cyclic data decomposition, load imbalance is eliminated, at least for dense contractions. Point-group symmetry is not yet implemented in CTF and would create some of the same type of load imbalance as seen in this work, albeit at the level of large distributed contractions rather than tiles. We hypothesize that static partitioning would be effective at mitigating load-imbalance in CTF resulting from point-group symmetry.

2.1.6. Inspector/Executor Conclusions and Future Work

We have presented an alternate approach for conducting load balancing in the NWChem CC code generated by the TCE. In this application, good load balance was initially achieved by using a global counter to assign tasks dynamically, but application profiling reveals that this method has high overhead which increases as we scale to larger numbers of processes. Splitting each tensor contraction routine into an inspector and an executor component allows us to evaluate the system's sparsity and gather relevant cost information regarding tasks, which can then be used for static partitioning. We have shown that the inspector-executor algorithm obviates the need for a dynamic global counter when applying performance model prediction, and can improve the performance of the entire NWChem coupled cluster application. In some cases the overhead from a global counter is so high that the inspector-executor algorithm enables the application to scale to a number of processes that previously was impossible because of the instability of the `NXTVAL` server when bombarded with tasks.

The technique of generating performance models for `DGEMM` and `SORT4` to estimate costs associated with load balancing is general to all compute-kernels and

can be applied to applications that require large-scale parallel task assignment. While other noncentralized DLB methods (such as work stealing and resource sharing) could potentially outperform such static partitioning, such methods tend to be difficult to implement and may have centralized components. The approach of using a performance model and a partitioning library together to achieve load balance is easily parallelizable (though in NWChem tensor contractions, we have found a sequential version to be faster because of the inexpensive computations in the inspector) and easy to implement and requires few changes to the original application code.

Because the technique is readily extendible, we plan to improve our optimizations by adding functionality to the inspector. For example, we can exploit proven data locality techniques by representing the relationship of tasks and data elements with a hypergraph and decomposing the graph into optimal cuts [175].

2.2 Runtime Execution Models

The previous section presented the Inspector/Executor algorithm, which reduces the overhead of centralized dynamic load balancing in the TCE by statically partitioning tasks to individual processes or process groups. However, that work focused much more on developing the *Inspector* component rather than the *Executor* component. The following section presents an intelligent Executor system called *WorkQ* which provides a relatively effortless way to overlap communication and computation in the TCE.

2.2.1. WorkQ Introduction

Many distributed-memory computational applications undergo a basic flow of execution: individual processes determine a task to complete, receive data from remote locations, compute on that data, send the result, and repeat until all tasks are handled or convergence is reached. Accomplishing this in a performance-optimal manner is complicated by communication wait times and variability of the computational costs among difference tasks. Several execution models and programming models strive to account for this by supporting standard optimization techniques, yet certain conditions exist for doing so effectively. In message-passing applications, for example, nonblocking communication routines must be preferred over blocking routines in order to hide the latency cost of communication. However, processes must also be capable of making *asynchronous progress* while communication occurs. At the same time, the balance of workload across processor cores must be maintained so as to avoid starvation and synchronization costs. If the variability of task execution time is not considered when incorporating latency-hiding optimizations, then suboptimal performance occurs. For instance,

if computation of a task finishes before a previous nonblocking routine completes, then starvation occurs despite the asynchronous progress. In order to eliminate this problem in such applications, there must exist a dynamic queue of task data in shared memory that accommodates irregular workloads. However, if this queue becomes overloaded with data relative to other queues on other compute nodes, then load imbalance and starvation also occur despite asynchrony. This section introduces and analyzes an execution model that accomplishes zero wait-time for irregular workloads while maintaining systemwide load balance.

Irregularity within computational applications commonly arises from the inherent sparsity of real-world problems. Load imbalance is a result of data decomposition schemes that do not account for variation due to sparsity. Not only is there fundamental variation in task dimensions associated with work items from irregular sparse data structures, but the variety and nonuniformity of compute node architectures and network topologies in modern supercomputers complicate the wait patterns of processing work items in parallel. For example, a process that is assigned a work item may need to wait for data to be migrated from the memory space of another process before computation can take place. Not only does incoming data often cross the entire memory hierarchy of a compute node; it may also cross a series of network hops from a remote location. The contention on these shared resources in turn complicates the development of distributed computational algorithms that effectively overlap communication and computation while efficiently utilizing system resources.

While nonblocking communication routines enable asynchronous progress to occur within a process or thread of execution, care must be taken to minimize overheads associated with overlapping. Polling for state and migrating data too

often between process spaces can be expensive. Also, often some local computation must occur before communication can take place. The Tensor Contraction Engine (TCE) of NWChem, the target application of this work, exhibits this behavior because a relatively sizable amount of local computation takes place to determine the global location of sparse tensor blocks before communication can take place. For these reasons, performance may be best when the communication sections have a dedicated core, especially in modern many-core environments, where sacrificing some cores for communication may result in the most optimal mapping for latency hiding.

In this section we study a new execution model, called WorkQ, that prioritizes the overlap of communication and computation while simultaneously providing a set of runtime parameters for architecture-specific tuning. This model achieves effective load balance while eliminating core starvation due to communication latency. Using an implementation of the WorkQ model, we perform various experiments on a benchmark that mimics the bottleneck computation within an important quantum many-body simulation application (NWChem), and we show good performance improvement using WorkQ. In Section 2.1.2., we provided the necessary background regarding the partitioned global address space (PGAS) model (Section 2.1.2..1), NWChem application 2.1.2..2, coupled-cluster method 2.1.2..3, and Tensor Contraction Engine (TCE) 2.1.2..5, so we now assume familiarity with those topics. Below, Section 2.2.2. discusses the motivation for constructing our execution model. Section 2.2.3. outlines the design and implementation of WorkQ, Section 2.2.4. describes a set of experimental evaluations, Section 2.2.5. discusses related work, and Section 2.2.6. summarizes our conclusions and briefly discusses future work.

2.2.2. Overlapping Communication and Computation

In this section we briefly present performance measurements that support our motivation for developing a new runtime execution model for processing tasks in applications such as the TCE-CC within Global Arrays. We begin by considering measurements from a simple trace of a tensor contraction kernel. We then discuss the implications of a new execution model.

2.2.2..1 Communication/Computation Overlap

In order to better understand the performance behavior of the TCE task execution model described at the end of section 2.1.2..5, we develop a mini-application that executes the same processing engine without the namespace complications introduced by quantum many-body methods. The details of this mini-app will be discussed in Sections 2.2.3..2 and 2.2.4..1, but here we present a simple trace and profile of the execution to better motivate and influence the design of our runtime in Section 2.2.3..

The (A) and (B) portions of Fig. 31 show an excerpt of a trace collected with the TAU parallel performance system [168] for 12 MPI processes on 1 node within a 16 node application executed on the ACISS cluster (described in Section 2.2.4.). This trace is visualized in Jumpshot with time on the horizontal axis, each row corresponding to an MPI process and each color corresponding to a particular function call in the application. Specifically, the purple bars correspond to step 1 in the TCE execution model described in the previous section. The green bars correspond to the one-sided get operation on the two tiles A and B from step 3 (step 2 is implicit in the mini-app and is thus not contained in a function). The yellow bars are non-communication work cycles, and the pink bars are DGEMM

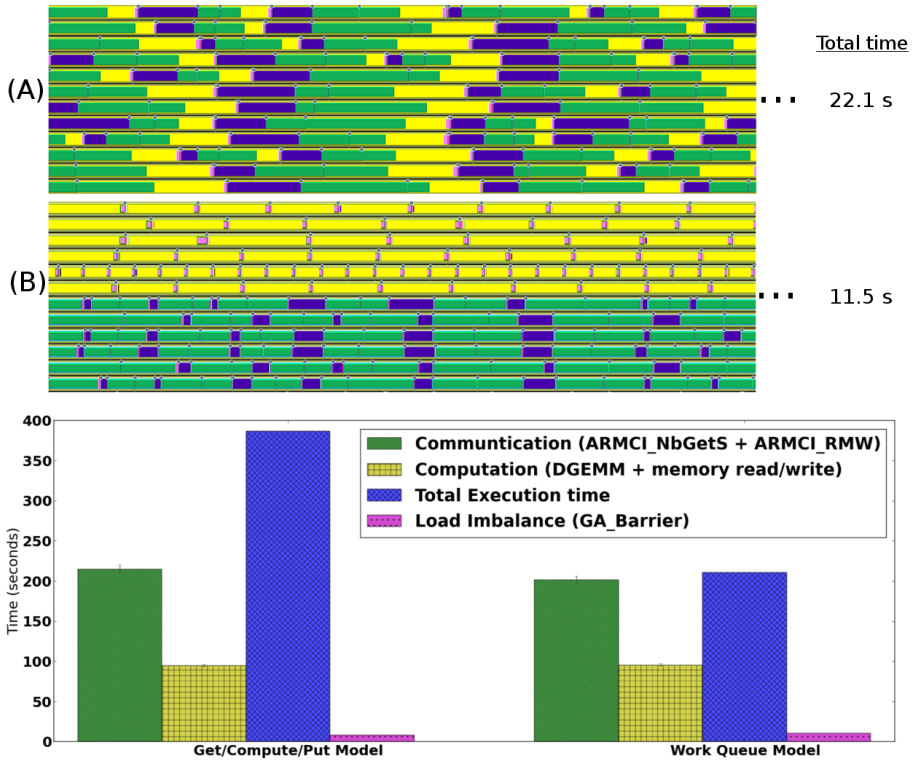


FIGURE 31. (Top:) TAU trace of original code (A) compared to WorkQ (B) for a 192 MPI process job. Yellow is computation, purple is `ARMCI_Rmw` (corresponding to `NXTVAL` call), green is `ARMCI_NbGetS` (corresponding to `GA_Get` call), and pink is `DGEMM`. (Bottom:) Profile information comparing the two execution models for a larger 960 MPI process job. The work queue implementation accomplishes full overlap without sacrificing load balance.

(these are small because of the relatively small tile size in this experiment). Yellow and pink together correspond to step 4. Step 5 is not shown in this timeline but occurs at a future point at the end of this task bundle. The 12 rows in portion (B) show a corresponding trace with our alternative parallel execution model in which 6 MPI processes dedicate their cycles to computation, and the other 6 processes dedicate their cycles to communication.

The bottom half of Fig. 31 contains timing information extracted from TAU profiles for a larger job with 960 MPI processes. The measurements clearly show that the work queue execution model accomplishes effective overlap of

communication with computation without sacrificing load imbalance. This results in a speedup of almost 2x over the original get/compute/put model for this experiment. The advantage can be inferred from the trace: in the original execution, there are moments when hardware resources are fully saturated with computation (i.e., all rows are yellow at a particular time) yet other moments where starvation is occurring (i.e., rows are green at a particular time). Besides dramatically reducing moments of work starvation, the alternative model enables tunability: for instance, the optimal number of computation versus communication processes can be determined empirically.

2.2.2..2 Variability in Work-Item Processing

The TCE engine uses blocking `GA_get()` and `GA_acc()` calls to gather and accumulate tiles, respectively, into the global space. While it is reasonable to use the corresponding nonblocking API for GA (`GA_nbget` and `GA_nbacc`) to accomplish overlap, doing so will not achieve optimal performance in the face of highly irregular workloads. For example, one can submit a nonblocking get before doing computation on local data; but if the computation finishes before the communication request is satisfied, then starvation occurs. Variation in execution time occurs often, because of either system noise or inherent differences in task sizes. This variability necessitates the calling of multiple nonblocking communication operations managed by a queue so that data is always available once an iteration of computation finishes. On the other hand, the number of work items in this queue must be throttled so that the queue does not become overloaded with respect to other queues on other nodes. If this were to happen,

then load imbalance would surely occur without the usage of techniques such as internode work stealing, which potentially incur high overheads.

2.2.3. WorkQ Design and Implementation

The desire to overlap communication and computation in a dynamic and responsive manner motivates the development of a library for managing compute node task queuing and processing within SPMD applications. We have implemented such a library, which we call WorkQ. This section presents the software architectural design for WorkQ, describes some implementation details and possible extensions, and presents a portion of the API and how it can be deployed for efficient task processing in distributed memory SPMD programs.

2.2.3..1 WorkQ Library

As described in Section 2.2.2., the TCE-CC task-processing engine has the potential to experience unnecessary wait times and relatively inefficient utilization of local hardware resources. Here we describe an alternative runtime execution model with the goals of (1) processing tasks with less wait time and core starvation, (2) exposing tunability to better utilize hardware resources, and (3) responding dynamically to real-time processing variation across the cluster.

Here we simplify the operations of TCE described in Section 2.1.2..5 into a pedagogical model that is akin to tiled matrix multiplication of two arrays, A and B . In this model, A and B contain GA data that is distributed across a cluster of compute nodes. The overall goal of the application is to multiply corresponding tiles of A and B, then to accumulate the results into the appropriate location of a separate global array, C. In order to accomplish this within the original execution

engine, individual processes each take on the execution loop from Section 2.1.2..5: `get(A); get(B); compute(A,B); acc(C)`. The behavior of a single compute node involved in this computation is characterized by the trace in the top half of Fig. 31: at any given moment in time, processes work within a particular stage of the execution loop.

In the WorkQ runtime, each compute node contains an FIFO message queue, $\mathbf{Q}_1$, in which some number of *courier processes* are responsible for placing A and B tile metadata onto $\mathbf{Q}_1$, then storing the incoming tiles into node-local shared memory. Meanwhile, the remaining *worker processes* dequeue task metadata bundles as they become available, then use this information to follow a pointer to the data in shared memory and perform the necessary processing. Once a worker process is finished computing its task result, it places the resultant data into a separate FIFO message queue, $\mathbf{Q}_2$, which contains data for some courier process to eventually accumulate into C.

We now describe the four primary components of the WorkQ implementation: the dynamic producer/consumer system, the on-node message queues, the on-node shared memory, and the WorkQ library API.

2.2.3..1.1 Dynamic Producer/Consumer This runtime system exhibits a form of the producer/consumer model in which courier processes are the producers and worker processes are the consumers. In the model described so far, couriers perform mostly remote communication, and workers constantly read/write data from/to local memory and perform a majority of the FLOPS in this phase of the application. However, we found via performance measurements that this system can still struggle with unacceptable wait times and starvation from the point of view of the workers. This situation occurs, for example, when $\mathbf{Q}_1$ is empty because

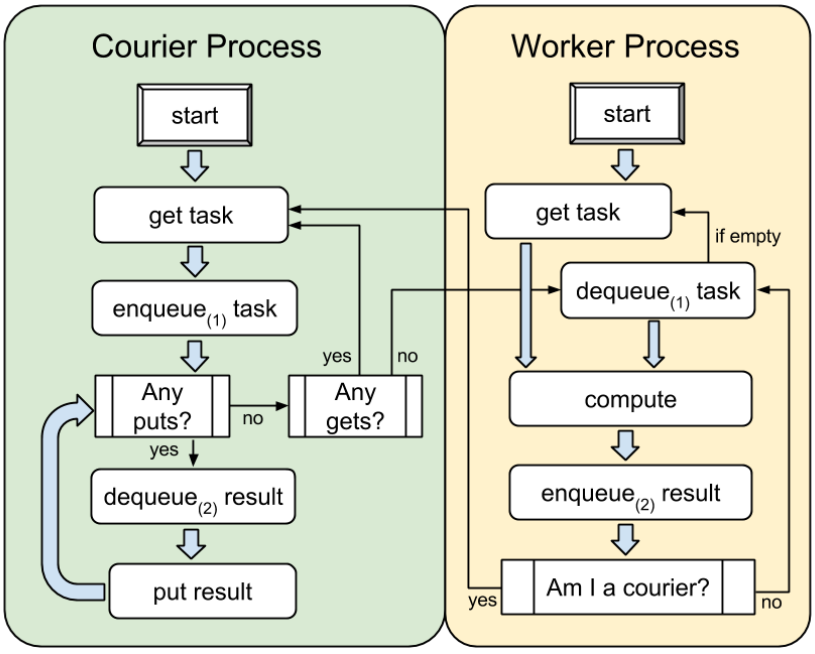


FIGURE 32. Flow diagram for the dynamic producer/consumer version of the WorkQ execution model. The left green column represents the activities of courier processes, and the right yellow column represents activities of worker processes. In this system, couriers can temporarily become workers and vice versa.

of either relatively long communication latencies or not enough couriers to keep the workers busy. For this reason, the WorkQ implementation allows for the dynamic switching of roles between courier and worker.

Figure 32 displays how role switching occurs in the WorkQ runtime. To bootstrap the system, *both* couriers and workers perform an initial `get()` operation. This "initialization" of $\mathbf{Q}_1$ is done to avoid unnecessary work starvation due to an empty initial queue. We determined this to be critical for performance, especially when the time to get/enqueue a task is greater than or equal to the compute time. If this is the case (and the number of workers approximately equals the number of couriers), the workers may experience several rounds of starvation before the couriers can "catch up."

After the first round (with workers computing on tiles they themselves collected), workers dequeue subsequent tasks to get data placed in $\mathbf{Q}_1$ by the couriers. If $\mathbf{Q}_1$ ever becomes empty when a worker is ready for a task, the worker will get its own data and carry on. On the other hand, if a courier finds either that $\mathbf{Q}_1$ is overloaded (as determined by a tunable runtime threshold parameter) or that $\mathbf{Q}_2$ is empty with no remaining global tasks, then the courier will become a worker, dequeue a task, and compute the result. In either case, the process will return to its original role as a courier until both $\mathbf{Q}_1$ and $\mathbf{Q}_2$ are empty.

2.2.3..1.2 Message Queues The nodewise metadata queues are implemented by using the System V (SysV) UNIX interface for message queues. This design decision was made because SysV message queues exhibit the best trade-off between latency/bandwidth measurements and portability compared with other Linux variants [181]. Besides providing atomic access to the queue for both readers and writers, SysV queues provide priority, so that messages can be directed to specific consumer processes. For example, this functionality is utilized to efficiently end a round of tasks from a pool: when a courier is aware it has enqueued the final task from the pool, it then enqueues a collection of finalization messages with a process-unique `mtype` value corresponding to the other on-node process IDs.

2.2.3..1.3 Shared Memory The message queues just described contain only metadata regarding tasks; the data itself is stored elsewhere in node-local shared memory. This approach is taken for three reasons: (1) it reduces the cost of contention on the queue among other node-resident processes, (2) Linux kernels typically place more rigid system limits on message sizes in queues (as seen with `ipcs -l` on the command line), and (3) the size and dimension of work items

vary significantly. The message queue protocol benefits in terms of simplicity and performance if each queued item has the same size and structure. Within each enqueued metadata structure are elements describing the size and location of the corresponding task data. The WorkQ library allows for either SysV or POSIX shared memory depending on user preference. There is also an option to utilize MPI_3 shared-memory windows (`MPI_Win_allocate_shared`) within a compute node. This provides a proof of concept for doing MPI+MPI [182] hybrid programming within the WorkQ model.

2.2.3..1.4 Library API The WorkQ API provides a productive and portable way for an SPMD application to initialize message queues on each compute node in a distributed-memory system, populate them with data, and dequeue work items. Here we list a typical series of calls to the API (because of space constraints, arguments are not included; they can be found in the source [183]):

- `workq_create_queue()`: a collective operation that includes on-node MPI multicasts of queue info
- `workq_alloc_task()`: pass task dimensions and initialize pointer to user-defined metadata structure
- `workq_append_task()`: push a microtask's data/metadata onto the two serialized bundles
- `workq_enqueue()`: place macrotask bundle into the queue then write real-data into shared memory
- `workq_dequeue()`: remove a macrotask bundle from the queue and read data from shared memory

- `workq_get_next()`: pop a microtask's metadata and real data in preparation for computation
- `workq_execute_task()`: (optional) a callback so data can be computed upon with zero copies
- `workq_free_shm()`: clean up the shared memory

WorkQ also includes a wrapper to SysV semaphores, which is needed only if the explicit synchronization control is needed (i.e., if certain operations should not occur while workers are computing). These functions are `workq_sem_init()`, `sem_post()`, `sem_release()`, `sem_getvalue()`, and `sem_wait()`.

2.2.3..2 TCE Mini-App

The performance of the WorkQ runtime system implementation is evaluated in two ways: directly, with the original NWChem application applied to relevant TCE-CC ground-state energy problems, and indirectly, with a simplified mini-app that captures the overall behavior of the TCE performance bottleneck (described in Section 2.1.2..5). The primary advantage of the mini-app is that it removes the need to filter through the plethora of auxiliary TCE functionalities, such as the TCE hash table lookups, or the many other helper functions within TCE. Although the mini-app will not compute any meaningful computational chemistry results, it captures the performance behavior of TCE in a way that is more straightforward to understand and simpler to tune. Furthermore, the tuned runtime configuration within the mini-app environment can be applied to NWChem on particular system architectures.

The TCE mini-app implements the pedagogical model described in Section 2.2.3..1: corresponding tiles from two global arrays (A and B) are

multiplied via a DGEMM operation and put into a third global array C. The mini-app is strictly a weak-scaling application that allows for a configurable local buffer length allocation on each MPI process. These buffers are filled with arbitrary data in the creation/initialization of A, B, and C. As in TCE, all global arrays are reduced to their one-dimensional representations [165]. The heap and stack sizes fed to the global arrays memory allocator [184] are set to as large as possible on a given architecture. Two versions of the code are implemented to calculate the entire pool of DGEMMs: one with the original `get/compute/put` model on every process and one with the WorkQ model on every compute node. The resulting calculation is verified in terms of the final vector norm calculated on C.

2.2.4. WorkQ Experimental Results

The performance of the WorkQ execution runtime compared with the standard `get/compute/put` model is evaluated on two different platforms. The first is the ACISS cluster located at the University of Oregon. Experiments are run on the 128 generic compute nodes, each an HP ProLiant SL390 G7 with 12 processor cores per node (2x Intel X5650 2.67 GHz 6-core CPUs) and 72 GB of memory per node. This is a NUMA architecture with one memory controller per processor. ACISS employs a 10 gigabit Ethernet interconnect based on a 1-1 nonblocking Voltaire 8500 10 GigE switch that connects all compute nodes and storage fabric. The operating system is RedHat Enterprise Linux 6.2, and MPICH 3.1 is used with the `-O3` optimization flag.

The second platform is the Laboratory Computing Resource Center "Blues" system at Argonne National Laboratory. The 310 available compute nodes each have 16 cores (2x Sandy Bridge 2.6 GHz Pentium Xeon with hyperthreading

disabled) and 64 GB of memory per node. All nodes are interconnected by InfiniBand Qlogic QDR. The operating system is Linux running kernel version 2.6.32. MVAPICH2 1.9 built with the Intel 13.1 compiler was used for our experiments.

Unless otherwise specified, performance experiments are executed with 1 MPI process per core, leaving 1 core open on each compute node for the ARMCI helper thread (for example, 11 processes per node on ACISS and 15 processes per node on Blues). Previous work has shown this mapping to be optimal for reducing execution time as suggested by detailed TAU measurements in NWChem TCE-CC [177].

The systems above provide a juxtaposition of the performance benefits gained with the WorkQ runtime between two very different network interconnects: Ethernet and InfiniBand (IB). The GA/ARMCI and MPI layers utilize socket-based connections on ACISS, meaning that the servicing of message requests involves an active role of each compute node's operating system. Blues, on the other hand, has full RDMA support, so data can be transferred between nodes without involving the sender and receiver CPUs.

2.2.4..1 TCE Mini-App

Our first experiment considers the weak scaling performance of the TCE mini-app on ACISS and Blues for two different tile sizes. The tile size in the mini-app corresponds to the common dimension of the blocks of data collected from the GAs described in Section 2.2.3..2. In this experiment, all DGEMM operations are performed on matrices with square dimensions, $N \times N$, where N is the so-called tile size. Figure 33 considers tile sizes 50 (2,500 total double-precision floating-point elements) and 500 (250,000 elements). The mini-app is a weak-scaling application

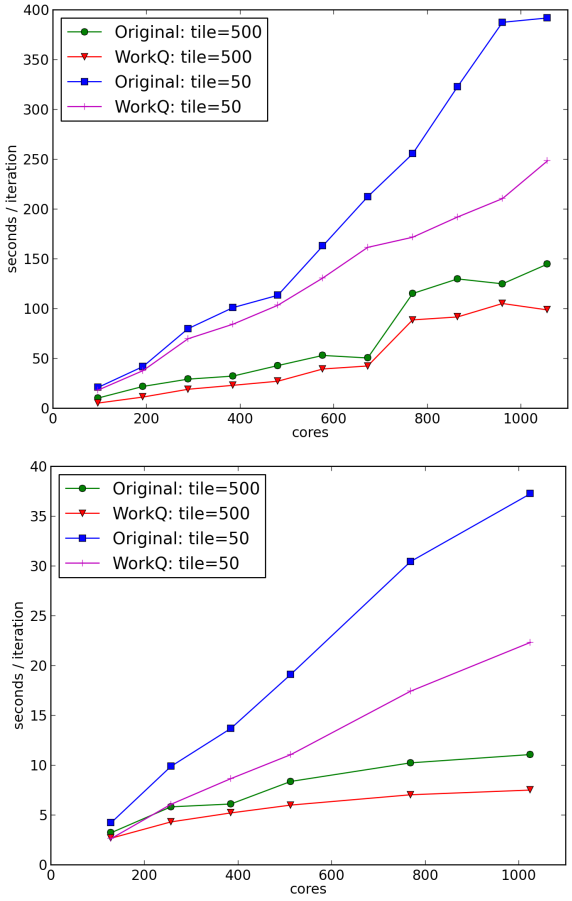


FIGURE 33. Weak-scaling performance of the TCE mini-app with ARMCI over sockets on ACISS (top) and ARMCI over InfiniBand on Blues (bottom) for different tile sizes. On ACISS the WorkQ implementation was run with 6 courier processes and 5 worker processes, and on Blues with 3 couriers and 4 workers. On both architectures, the WorkQ execution shows better relative speedup with small tile sizes but better absolute performance for relatively larger tile sizes.

in which a constant amount of memory is allocated to each process/core at any given scale. That is, if the scale is doubled, then the size of the overall computation is doubled (hence, execution time increases for higher numbers of processes). GA's internal memory allocator is initialized so that the total heap and stack space per node is about 20 GB.

Figure 33 clearly shows that using the relatively large tile size of 500 results in better overall absolute performance for both the WorkQ execution model

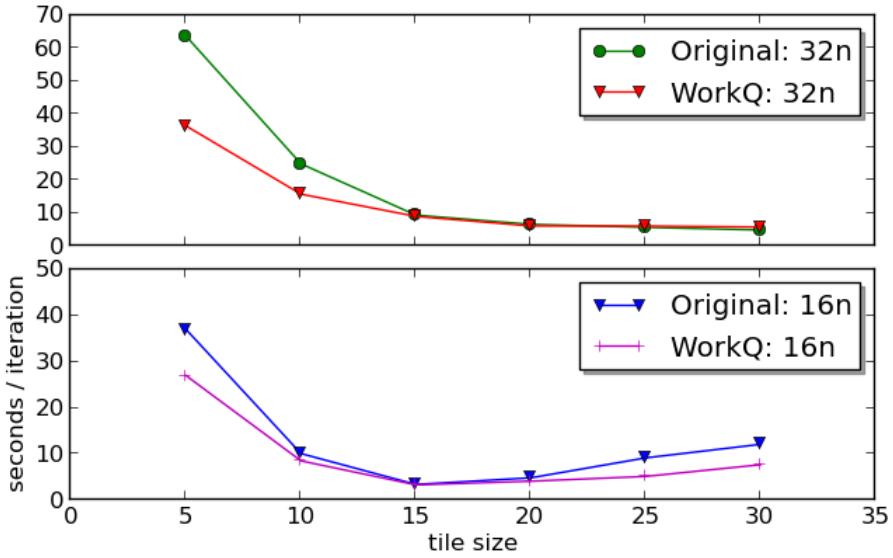


FIGURE 34. Time per CCSD iteration for w3 aug-cc-pVDZ on ACISS versus tile size. The top row contains execution measurements on 32 nodes (384 MPI processes), and the bottom row contains measurements on 16 nodes (192 MPI processes).

and the original execution model. This phenomenon is well understood [18] and is due mainly to the overhead associated with data management and dynamic load balancing when tile size is relatively small. In general, larger tile sizes are desirable in order to minimize this overhead, but at a certain point large tiles are detrimental to performance because it leads to work starvation. For instance, if more processes/cores are available to the application than there are number of tiles, then work starvation will surely occur.

On the other hand, the best speedups achieved with the WorkQ model on both systems are seen with the smaller tile size of 50, particularly at relatively large scales. Our TAU profiles show that at a tile size of 50, the total time spent in communication calls (`ARMCI_NbGetS` and `ARMCI_NbAccS`) is considerably larger than with a tile size of 500. These results suggest that at smaller tile sizes, there is more cumulative overhead from performing one-sided operations and therefore more

likelihood that processes will spend time waiting on communication. This scenario results in more opportunity for overlap but worse absolute performance because of the incurred overhead of dealing with more tasks than necessary.

2.2.4..2 NWChem

We now analyze the performance of the WorkQ model applied to TCE in NWChem by measuring the time of execution to calculate the total energy of water molecule clusters. These problems are important because of their prevalence in diverse chemical and biological environments [185]. We examine the performance of the tensor contraction that consistently consumes the most execution time in the TCE CCSD calculation, corresponding to the term

$$r_{h_1h_2}^{p_3p_4} \mathrel{+}= \frac{1}{2} t_{h_1h_2}^{p_5p_6} v_{p_5p_6}^{p_3p_4}$$

(see [165] for details regarding the notation). In TCE, this calculation is encapsulated within routine `ccsd_t2_8()` and occurs once per iteration of the Jacobi method.

Figure 34 shows the minimum measured time spent in an iteration of `ccsd_t2_8()` on a 3-water molecule cluster using the aug-cc-pVDZ basis set across a range of tile sizes. These measurements are on the ACISS cluster at two different scales: 32 compute nodes in the top plot and 16 compute nodes in the bottom plot, with 12 cores per node in each case. Here we use the minimum measured execution time for a series of runs because it is more reproducible than the average time [186]. On 16 nodes, we see overall performance improvement with WorkQ across most measured tile sizes. As in the TCE mini-app (Fig. 33), WorkQ shows

better performance *improvement* at small tile sizes but best *absolute* performance with a medium-sized tile. The performance with the small tile size is important because NWChem users do not know *a priori* which tile size is appropriate for a given problem. It is typically best to initially choose a relatively small tile size because load imbalance effects can be avoided with a finer granularity of task sizes.

2.2.5. Related Work to the WorkQ Execution Model

Other execution and programming models incorporate node-local message queues for hiding latency and supporting the migration of work units. For example, Charm++ provides internal scheduling queues that can be used for peeking ahead and prefetching work units (called *chares*) for overlapping disk I/O with computation [187]. The Adaptive MPI (AMPI) virtualization layer can represent MPI processes as user-level threads that may be migrated like chares, enabling MPI-like programming on top of the Charm++ runtime.

Another interesting new execution model targeted to exascale development is ParalleX, which is implemented by the HPX runtime [188]. The ParalleX model extends PGAS by permitting migration of objects across compute nodes without requiring transposition of corresponding virtual names. The HPX thread manager implements a work-queue-based execution model, where parcels containing active messages are shipped between "localities." Like ParalleX, WorkQ provides the benefit of implicit overlap and load balance with the added feature of dynamic process role switching, which keeps the queue populated if too few items are enqueued and throttled if too many are enqueued. Unlike HPX and Charm++, the WorkQ library API enables such implicit performance control on top of other portable parallel runtimes, such as MPI itself and Global Arrays/ARMCI.

In execution models based on node-local message queues, work stealing enables more adaptive control over load balance. The work-stealing technique is well studied, especially in computational chemistry applications [163, 189]. In a typical work-stealing implementation, local work items are designated as tasks that may be stolen by other processes or threads. In some sense, most tasks in WorkQ are stolen from on-node couriers by workers. Couriers work on their own tasks only if the queue is deemed overloaded, and workers take tasks from the global pool if the queue is running empty.

Novel developments in wait-free and lock-free queuing algorithms with multiple enqueuers and dequeuers [190] could potentially improve performance of this execution system by reducing contention in shared memory. SysV and POSIX queues provide atomicity and synchronization in a portable manner, but neither is wait-free or lock-free.

2.2.6. WorkQ Conclusion

The `get/compute/put` model is a common approach for processing a global pool of tasks, particularly in PGAS applications. This model suffers from unnecessary wait times on communication and data migration that could potentially be overlapped with computation and node-level activities. The WorkQ model introduces an SPMD-style programming technique in which nodewise message queues are initialized on each compute node. A configurable number of courier processes dedicate their efforts to communication and to populating the queue with data. The remaining worker processes dequeue and compute tasks. We show that a mini-application that emulates the performance bottleneck of the TCE achieves performance speedups up to 2x with a WorkQ library implementation. We

also show that WorkQ improves the performance of NWChem TCE-CCSD across many tile sizes on the ACISS cluster.

Future work will include the incorporation of internode work stealing between queues and performance analysis of the queuing system using event-based simulation techniques.

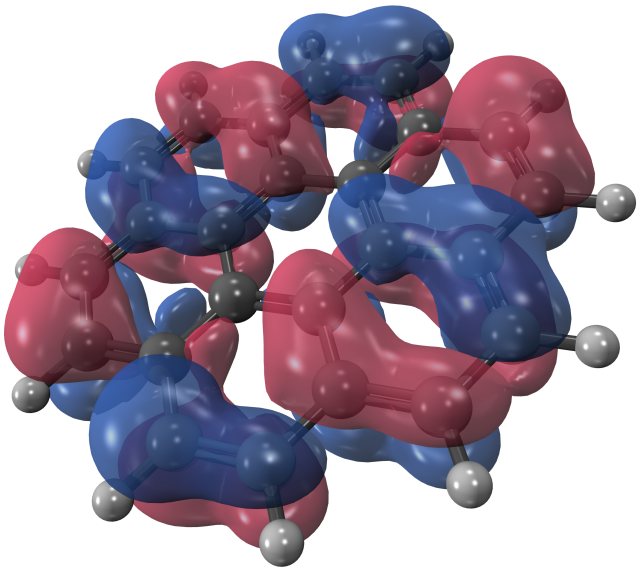


FIGURE 35. A graphene subunit with 36 atoms. The blue and red blobs show the lowest unoccupied molecular orbital based on a Hartree-Fock calculation.

2.3 Parallel Programming Models

2.3.1. UPC++ and DArrays Introduction

In order to develop the next generation of materials and chemical technologies, molecular simulations must incorporate quantum mechanics to effectively predict useful properties and phenomena. However, chemical simulations that include quantum effects are computationally expensive and frequently scale superlinearly in the number of atoms in the system simulated. Even for a relatively modest system like the graphene unit in Figure 35, substantial computing power is required for an accurate treatment of quantum effects. Despite the opportunities for exploiting parallelism, even the most sophisticated and mature chemistry software tools exhibit scalability problems due to the inherent load imbalance and difficulty in exploiting data locality in quantum chemistry methods.

The Hartree-Fock (HF) method is the quintessential starting point for doing such *ab initio* calculations, so most all quantum chemistry codes provide some sort of HF implementation. Additionally, the more accurate and very popular *post-Hartree-Fock* methods, such as coupled cluster and many-body perturbation theory, depend on the reference wavefunction provided by HF. Nearly all quantum chemistry codes base their post-Hartree-Fock software implementations on the programming model and data structures used in the corresponding HF component. It is therefore vital, when developing modern quantum chemistry codes, to start from the ground-up with very efficient and scalable programming constructs in the HF modules.

Unfortunately, the most well-used and scalable HF and post-HF codes often utilize programming models that do not embrace modern software capabilities, such as those provided by C++11 and C++14. For example, NWChem is generally considered to be the most scalable of all quantum chemistry codes [3],[4], yet it uses a toolkit called Global Arrays (explained fully in section 2.3.2..2), which only supports distributed arrays containing elements of type `int`, `float`, `double`, or `complex`. Even new codes such as GTFock, which introduce impressive algorithmic enhancements to HF, still use the same Global Arrays programming model without support for features like distributed structures of objects, active messages, and remote memory management. UPC++ is parallel programming library that supports these modern features. Others have recently utilized these capabilities of UPC++ for scientific calculations, while achieving comparable or better performance than similar implementations that use only MPI [191, 192].

This section explores the use of UPC++, a partitioned global address space (PGAS) extension for C++, for doing HF calculations. We base our

implementation on the GTFock code, but instead of using the Global Arrays (GA) library for operating on distributed data structures, we use a new DArray library written in UPC++. The DArray library incorporates the functionality of GA while supplanting it with new features, such as the ability to apply user-defined functions across distributed tiles of the global data structure. Using DArrays, our UPC++ HF application accomplishes the same algorithmic improvements as GTFock, while making efficient use of unique DArray capabilities. Our measurements show that the UPC++ version of Hartree-Fock achieves as much as 20% performance improvement over GTFock with up to 24,000 processor cores.

2.3.2. Hartree-Fock and UPC++ Background

2.3.2..1 The Hartree-Fock Method

The goal of the Hartree-Fock (HF) method is to approximately solve the many-body time-independent Schrödinger equation, $\mathbf{H}|\psi\rangle = E|\psi\rangle$, where $\mathbf{H}$ is the Hamiltonian operator, which extracts the sum of all kinetic and potential energies, E, from the wavefunction, $|\psi\rangle$. Here we make the standard set of assumptions: the Born-Oppenheimer approximation (in which nuclei are fixed but electrons move freely), Slater determinant wavefunctions (that easily satisfy the anti-symmetry principle), and non-relativistic conditions. After these approximations, our focus resides only on the electronic terms of the Hamiltonian and the wavefunction: $\mathbf{H}_{elec}|\psi_{elec}\rangle = E|\psi_{elec}\rangle$.

In both HF and post-HF methods, the molecular orbitals that express the electronic wavefunction consist of a sum of *primitive basis functions* from set $\{\phi_j\}$. We desire $\{\phi_j\}$ to be complete, but this is not practical since it generally requires an infinite number of functions. We therefore truncate to a finite n value large

enough to balance the trade-off between accuracy and computational cost:

$$|\psi_i\rangle = \sum_{j=1}^{n} c_{ij} \, |\phi_j\rangle$$

Typically, each primitive is a Gaussian function centered at the nucleus location. The HF method attempts to determine the c_{ij} values that best minimize the ground state energy in accordance with the variational principle. By utilizing a numerical technique for iteratively converging the energy, each subsequent iteration becomes more and more consistent with the field that is imposed by the input molecule and basis set. The method is accordingly called the self-consistent field (SCF) method, and Algorithm 1 shows its de facto procedure in pseudocode.

Many SCF iterations are required for energy convergence, so steps 1-3 of Algorithm 1 cost much less than steps 5-10. Step 5 normally comprises a majority of the execution time in Hartree-Fock codes, because each element of the Fock matrix requires computing several two-electron integrals:

$$F_{ij} = H_{ij}^{core} + \sum_{\lambda\sigma} (2(\mu\nu|\lambda\sigma) - (\mu\lambda|\nu\sigma))$$

The two-electron integrals on the right-hand side are plentiful and expensive to compute [34]. They take this form:

$$(\mu\nu|\lambda\sigma) = \iint \phi_\mu^*(\mathbf{r}_1)\phi_\nu(\mathbf{r}_1) r_{12}^{-1} \phi_\lambda^*(\mathbf{r}_2)\phi_\sigma(\mathbf{r}_2) d\mathbf{r}_1 d\mathbf{r}_2 \tag{2.2}$$

As mentioned, there are formally $\mathcal{O}(n^4)$ integrals to compute within the basis set. However, from the definition in 2.2, we see there are only $\sim n^4/8$ unique integrals

by permutational symmetry and the number of non-zero integrals is asymptotically $\mathcal{O}(n^2)$ when Schwartz screening is employed.

While the two-electron integrals comprise the most computation time in SCF, communication and synchronization overheads can very well dominate, particularly at large scale. As discussed below (Section 2.3.2..5), inherent communication costs arise from the parallel decomposition of HF codes, leading to load imbalance and synchronization delays. Parallel HF applications also exhibit highly diverse accesses across distributed process memory spaces. As such, HF is well-suited for a programming model that emphasizes lightweight and one-sided communication within a single global address space. This is the subject of the next section.

2.3.2..2 PGAS in Quantum Chemistry

The algorithmic characteristics and resource requirements of HF (and post-HF) methods clearly motivate the use of distributed computation. HF tasks are independent and free to be computed by any available processor. Also, simulating a molecule of moderate size has considerable memory requirements that can easily exceed the memory space of a single compute node. However, at a high level of abstraction, *indexing into distributed HF data structures need not be different than indexing shared memory structures.* This programming abstraction is the basis of the partitioned global address space (PGAS) model for distributing and interacting with data. In computational chemistry, this model is advantageous for interacting with arrays and tensors productively and efficiently.

The PGAS model utilizes one-sided communication semantics, allowing a process to access remote data with a single communication routine. Remote memory access (RMA) is particularly advantageous in HF applications for three

primary reasons. First, HF exhibits highly irregular access patterns due to the intrinsic structure of molecules and the necessary removal of integrals from the Fock matrix in a procedure called "screening." Second, there is a need for efficient atomic accumulate operations to update tiles of the global matrices without the explicit participation of the target process. Finally, dynamic load balancing in HF is usually controlled by either a single atomic counter [3], or many counters [48, 18], both of which require a fast one-sided fetch-and-add implementation.

The NWChem software toolkit [49] paved the way towards the ubiquity of PGAS in computational chemistry using the Global Arrays (GA) library and the underlying ARMCI messaging infrastructure [3, 4]. GTFock followed suit, also using GA for tiled accesses across the irregularly distributed Fock and density matrices. The GA model is applicable to many applications, including ghost cells and distributed linear algebra, but GA's primary use today is for quantum chemistry. GA is limited to a C and Fortran API, but does have Python and C++ wrapper interfaces. Besides UPC++, which is the subject of the next section, there exist many other PGAS runtimes, including but not limited to: Titanium, X10, Chapel, Co-Array Fortran, and UPC.

2.3.2..3 UPC++ and Multidimensional Arrays

UPC++ is a C++ library for parallel programming with the PGAS model. It includes features such as global memory management, one-sided communication, remote function invocation and multidimensional arrays [191].

2.3.2..3.1 Multidimensional Arrays A general multidimensional array abstraction is very important for scientific applications, but unfortunately, the

support for multidimensional arrays in the C++ standard library is limited [193]. The UPC++ library implements a multidimensional array abstraction that incorporates many features that are useful in the PGAS setting, and we build our own distributed array representation on top of UPC++ multidimensional arrays. Here, we briefly introduce the concepts that are relevant to implementing the distributed arrays used in the HF algorithm:

- A *point* is a set of N integer coordinates, representing a location in N-dimensional space. The following is an example of a *point* in three-dimensional space:

```
point<3> p = {{ -1, 3, 2 }};
```

- A *rectangular domain* (or *rectdomain*) is a regular set of points between a given lower bound and upper bound, with an optional stride in each dimension. Rectangular domains represent index sets, and the following is an example of the set of points between $(1, 1)$, inclusive, and $(4, 4)$, exclusive:

```
rdomain<2> rd( PT(1,1), PT(4,4) );
```

- A UPC++ multidimensional array, represented by the `ndarray` class template, is a mapping of points in a rectdomain to elements. The following creates a two-dimensional `double` array over the rectdomain above:

```
ndarray<double,2> A( rd );
```

UPC++ supports a very powerful set of operations over domains, including union, intersection, translation, and permutation. Since UPC++ multidimensional arrays can be created over any rectangular domain, these domain operations

simplify the expression of many common array operations. New views of an array's data can also be created with transformations on its domain. For example, the following code transposes an array `A` into a new array `B` by creating `B` over a transpose of `A`'s domain, creating a transposed view of `A`, and copying that view into `B`:

```
rectdomain<ndim> permuted_rd =
    A.domain().transpose();
ndarray<T, ndim> B( permuted_rd );
B.copy(A.transpose());
```

An `ndarray` represents an array that is stored in a single memory space. However, the memory space containing an `ndarray` may be owned by a remote process; the `global` template parameter is used to indicate that a given `ndarray` may reside in a remote memory space:

```
ndarray<T, ndim, global> remote_array;
```

In keeping with the PGAS model, UPC++ supports one-sided remote access to an `ndarray`, both at the granularity of individual elements and, using the `copy` operation, a set of multiple elements.

The one-sided `copy` operation on `ndarray`s is an especially powerful feature. In the call B.copy(A), the library automatically computes the intersection of the domains of `A` and `B`, packs the elements of `A` that are in that intersection if necessary, transfers the data from the process that owns `A` to the process that owns `B`, and unpacks the elements into the appropriate locations in `B`. Active messages are used to make this process appear seamless to the user, and the library also supports non-blocking transfers using the async_copy() method on `ndarray`s.

A final domain and array feature used in the HF code is the upcxx_foreach construct, which iterates over the points in a domain. This allows the expression of dimensionality-independent loops, such as the following code that sets all elements of an array `A` to 0.0:[2]

```
upcxx_foreach( pt, A ) { A[pt] = 0.0; };
```

2.3.2..3.2 Other UPC++ Features UPC++ shared arrays, a feature inherited from the Unified Parallel C (UPC) language, are simple 1D arrays block-cyclically distributed across all processes. The following declares a shared array:

```
shared_array<T> A;
```

where `T` is the type of the array elements and should be *trivially copyable*. Before its first use, a shared array must be explicitly initialized by calling `init()` with the array size and block size as arguments. The creation of a shared array is collective, which means that the set of processes storing the array must agree on the same array size and block size. Shared arrays are limited in that they are only one-dimensional and have fixed block sizes.

To help move computation and save communication, UPC++ extends C++11's `async` feature for distributed-memory systems, enabling functions to execute on any node in a cluster. UPC++ uses C++11 variadic templates to package function arguments together with the function object and ships the closure to a remote node via GASNet [194].

[2]This example is just for illustration, since the call A.set(0.0) accomplishes this much more concisely.

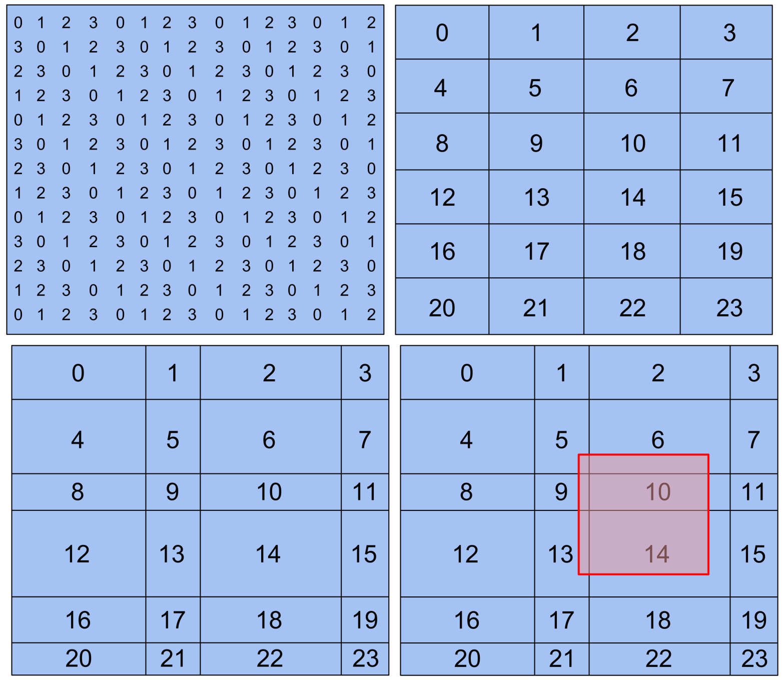


FIGURE 36. By default, UPC++ uses a regular round-robin distribution (upper left), while GA uses a regular 2D block-cyclic distribution (upper right). After screening and repartitioning, HF arrays are *irregularly* distributed (bottom left). A tiled access (in red) may span multiple owning processes (bottom right), in this case ranks 5, 6, 9, 10, 13, and 14.

2.3.2..4 Distributed Arrays with Irregular Ownership

HF and post-HF methods require storing large, irregularly distributed arrays. Figure 36 shows examples of *regularly* distributed arrays with round-robin and 2D block-cyclic assignments, and *irregularly* distributed arrays with 2D blocks. In HF, the irregular distributions arise after screening out negligible integral quantities. For example, the top right of Figure 36 may represent the initial assignment of the Fock matrix to processes. However, after determining the number of non-zeros in the collection of all shell pairs, the matrix is *repartitioned* so that each process is assigned approximately the same number of non-zero tasks. This new distribution might look more like the bottom left of Figure 36.

Like GA, UPC++ shared arrays can be either regularly or irregularly distributed, as shown in Figure 36. Unlike GA, element assignment in UPC++ is round robin by default; however, irregular arrays with `N` dimensions containing elements of type `T` can easily be created in this manner:

```
shared_array< ndarray<T, N> > A;
A.init( my_local_array );
```

where each rank defines `my_local_array` across its relevant portion of the global rectdomain. GA creates a similar structure using the `NGA_Create_irreg` API call. GA supports arbitrary tiled accesses from the global array; however, prior to this work, UPC++ did not support such accesses unless done for each owning process *explicitly*. More specifically, if `tile` is the object represented in red in Figure 36, then a `get` call would need to occur for *all* ranks 5, 6, 9, 10, 13, and 14. This motivates a higher level of abstraction when doing tiled access operations. As a result, we implemented a *distributed* version of the multidimensional array library, the implementation of which is the subject of Section 2.3.3..2.

2.3.2..5 Load Balancing in Quantum Chemistry

The primary hindrance to scalability in quantum chemistry codes is often due to load imbalance [50, 51, 18]. Although individual two-electron integrals do not possess drastic differences in execution time, bundles of *shell quartets* can vary greatly in cost. It is necessary to designate shell quartets as task units in HF codes because it enables the reuse of intermediate quantities shared by basis functions within a quartet [52, 48]. The goal is to assign these task units to processors with minimal overhead and a schedule that reduces the makespan.

NWChem balances load with a centralized dynamic scheduler, which is controlled by a single global counter control referred to as `nxtval` (for "next value"). Each task has a unique ID, and a call to `nxtval` *fetches* the current ID of an uncomputed task, then atomically *adds* 1 to the counter so the next task gets executed. Once the counter reaches the total number of tasks, all work is done. For this reason, the performance of RMA fetch-and-add operations is very important for the scalability of computational chemistry codes like NWChem and GTFock. This has motivated the implementation and analysis of hardware-supported fetch-and-ops on modern interconnects, such as Cray Aries using the GNI/DMAPP interfaces [45].

The `nxtval` scheme exhibits measurable performance degradation caused by network contention, but it can be alleviated by an informed static partitioning of tasks and the addition of atomic counters to every process or compute node [18]. GTFock takes this notion one step further by following the static partitioning phase with *work stealing*. During the local phase, each process only accesses the local memory counter; however, during the work stealing phase, the process accesses other counters remotely. As illustrated in Algorithm 2, each process in GTFock begins an SCF iteration by prefetching the necessary tiles from the global density matrix and storing them into a local buffer. After all local tasks are computed, the global Fock matrix is updated. Then, each process probes the `nxtval` counters of remote processes, looking for work to steal. As we will see in Section 2.3.4..2, this algorithm results in many more *local* fetch-and-adds than *remote*, which has important performance implications for how the operations take place.

2.3.2..6 Making Progress in PGAS Runtimes

Modern network hardware often supports RDMA (remote direct memory access), which can transfer contiguous data without involving the remote CPU at the destination. But for more complex remote operations such as accumulates, non-contiguous data copies, and matrix multiplications, the remote CPU needs to participate in processing the tasks. In addition, the GASNet communication library used by UPC++ also requires polling the network regularly to guarantee progress. For example, in Algorithm 2, if process A is busy doing computation in line 5 while process B is trying to steal a task in line 9, then GASNet on process A may be unable to make progress until the main application thread can reach line 4 to probe the runtime. This scenario unfortunately leads to work starvation on process B.

The key design issue here is how to make CPUs *attentive* to remote requests without wasting cycles for unnecessary polling. For instance, Global Arrays usually enables asynchronous progress with a software agent that polls continuously (interrupt-driven progress is less common, but was used on Blue Gene/P, for example). This explains why NWChem usually shows the best performance when each compute node dedicates a spare core to the constantly polling helper thread [177]. However, the optimal polling rate is generally application-dependent, so UPC++ provides two polling options: 1) explicit polling by the user application; and 2) polling at regular intervals where the user controls when to start and stop. UPC++ also allows processes on the same compute node to cross-map each other's physical memory frames into its own virtual memory address space (e.g., via POSIX or System V shared memory API), which can be used to implement a process-based polling mechanism like Casper [195].

2.3.3. DArray Design and Implementation

This section describes the design and implementation of our Hartree-Fock in UPC++. We focus on the new DArray library, which incorporates the functionality of Global Arrays into UPC++.

2.3.3..1 UPC++ Hartree-Fock

We base our UPC++ implementation of HF on the GTFock code because of its exceptional load balancing capabilities and support for OpenMP multi-threading. GTFock itself contains about 5,000 lines of C code, and it uses the optimized electron repulsion integral library, OptErd [196], which contains about 70,000 lines of C and Fortran.

Because the main GTFock application is written in C, porting it to use UPC++ was mostly straightforward for *computational* components of the code. In particular, many GTFock routines solely do calculations (such as two-electron integrals) with little to no communication. An example is the local computation of the Fock matrix elements, which is the computational bottleneck of the application and makes optimized use of OpenMP clauses. In some cases, our UPC++ implementation uses C++ objects in place of GTFock's C structures, but the port primarily retains the C-style memory management and data structures.

For *communication* regions however, porting the GTFock code required substantial development and even new UPC++ features. In particular, the functionality of Global Arrays needed to be created using the UPC++ model. To aid with this, we created the DArray library, which is the subject of the next section.

2.3.3..2 The DArray Library

The DArray library uses UPC++ functionality to implement tiled operations on irregularly distributed arrays, like those in GTFock. This section begins by presenting how the library is used and ends with some discussion regarding the implementation. The creation of a DArray object is very similar to the creation of UPC++ multidimensional arrays:

```
/* Create a local block with rectdomain (p_local, q_local)
in the SPMD style, then call the DArray constructor.
The rectdomain (p_global, q_global) defines the global
array space of dimension M × N. */

/* Local array */
point<2> p_local = PT( p_row, p_col );
point<2> q_local = PT( q_row, q_col );
rectdomain<2> rd = RD( p_local, q_local );
ndarray<T, 2> local_arr( rd );
...

/* Global DArray */
point<2> p_global = PT( 0, 0 );
point<2> q_global = PT( M, N );

rectdomain<2> rd_global = RD( p_global, q_global );
DArray<T, 2> A( prows, pcols, rd_global, local_arr );
```

Now the DArray can be used for arbitrary gets and puts, even if the desired `tile` is owned by multiple ranks:

```
/* Get data residing in global rectdomain (p,q),
   then place it into a local ndarray. */
ndarray<T, 2> tile;
tile = A( p, q );

/* Put local data into rectdomain rd. */
ndarray<T, 2> tile( rd );
upcxx_foreach( pt, rd ) { tile[pt] = rand(); }
A.put( tile );
```

UPC++ `async` functions also allow for custom user-defined functions to be applied to arbitrary tiles. For example, here is how we implement accumulate:

```
/* User-defined accumulate function. */
void my_accum(ndarray<T,2,global> local_block,
              ndarray<T,2,global> remote_block) {
  ndarray<T,2,global> block(remote_block.domain());
  block.copy(remote_block);
  upcxx_foreach( pt, block.domain() ) {
    local_block[pt] += block[pt];
  }; }

/* Accumulate local data into rectdomain rd. */
ndarray<T, 2> tile( rd );
upcxx_foreach( pt, rd ) { tile[pt] = rand(); }
A.user_async( tile, my_accum );
```

Finally, DArrays support simple linear algebra operations such as matrix addition, multiplication, and transpose. Currently, only two-dimensional operations are supported, because they are relevant to the HF application. However, we plan to incorporate functionality for arbitrary array dimensions in the future.

Within the DArray library itself, UPC++ multidimensional arrays are used to keep track of data ownership, make copies to/from restricted rectdomains when necessary, and manipulate local data blocks. For instance, the `DArray::transpose()` method first performs a local `ndarray::transpose()` on each block, then makes a restricted-set copy to the ranks that own the data in the transposed view.

2.3.3..3 Load Balancing / Work Stealing

Our implementation of Hartree-Fock in UPC++ uses the same strategy as GTFock for load balancing as described in Section 2.3.2..5 and outlined in Algorithm 2. However, there are many ways to implement the dynamic task counters, and we will see in Section 2.3.4..2 that these alternatives have very different performance characteristics. This section outlines the different alternatives we have explored for carrying out the counter fetch-and-adds.

2.3.3..3.1 UPC++ Shared Arrays This naive implementation simply uses a UPC++ shared array of counters and does fetch-and-adds to the elements directly. However, this is incorrect because shared arrays do not support atomic writes without locks. If multiple processes simultaneously add to a shared counter, it is possible for one of the processes to overwrite the counter with a stale value. We describe this version here only to clarify that the operation is not trivial.

2.3.3..3.2 UPC++ asyncs This approach uses UPC++ `async` functions to increment the values of a shared array or some element of a global DArray. Because `async`'s are enqueued at the target process, they are executed atomically.

2.3.3..3.3 GASNet active messages GASNet active messages (AM's) allow user-defined handlers to execute at a target process with the message contents passed as arguments. UPC++ `async` functions are built on top of GASNet AM's, but they carry more infrastructure to enable greater flexibility (for instance, an `async` can launch GPU kernels or contain OpenMP pragmas). On the other hand, GASNet AM handlers are more lightweight than `asyncs`, which makes them a good candidate for a simple operation like fetch-and-add. UPC++ has an experimental feature for launching such an AM with the `upcxx::fetch_add()` function.

2.3.3..3.4 GASNet GNI atomics The Cray Gemini and Network Interface (GNI) and Distributed Shared Memory Application (DMAPP) interface provide access to one-sided communication features available to Cray Aries and Gemini network interface controllers (NIC's). For instance, 8-byte aligned atomic memory operations, such as fetch-and-add, have native hardware support on the NIC itself. They also have an associated cache for fast accesses to such data, which is particularly efficient for remote work stealing. However, accessing counters resident in a *local* NIC cache has relatively high overhead compared to accessing DRAM. GASNet does not yet expose such network atomics, but for this work we introduce a prototype implementation of fetch-and-add on 64 bit integers, anticipated to be included in GASNet-EX, the next-generation GASNet API.

2.3.3..4 Attentiveness and Progress Threads

As discussed in Section 2.3.2..6, making good progress in the PGAS runtime is very important. This is particularly true while a process attempts to steal work from a victim that is busy doing computation. In order to improve the

attentiveness while the application executes Algorithm 2, we added a progress thread *start/stop* feature, which the application may use to initiate runtime progress polling. In its initial implementation, the *progress_thread_start()* function spawned a pthread that calls *gasnet_AMpoll()* intermittently. We experimented across several orders of magnitude for the polling frequency and chose every 10 μs for the Hartree-Fock application. The user should call the start function just before the process will do computation and will *not* do any communication. If communication occurs before calling *progress_thread_stop()*, it requires a thread-safe version of GASNet.

Section 2.3.4..3 presents measurements that suggest that calling *pthread_join()* at the end of each task incurs too much overhead for the HF application because there are potentially very many small tasks per process. We therefore introduce the *progress_thread_pause()* function. This function makes use of *pthread_cond_wait()* to sleep the progress thread and block it on a condition variable, whereas *pthread_join()* typically busy waits on a mutex (as in the GNU C Library implementation), consuming valuable CPU resources. In this new version, *progress_thread_start()* spawns the progress thread upon the first call, then signals the thread to unblock and continue polling in subsequent calls. Algorithm 9 shows the correct placement of these progress thread controller functions.

2.3.4. Hartree-Fock with DArray Measurements and Results

This section highlights the performance measurements of our UPC++ HF application and compares it with GTFock, which uses Global Arrays and ARMCI. All experiments were run on the *Edison* supercomputer at the National Energy Research Scientific Computing Center (NERSC). Edison is a Cray XC30 petaflop

Algorithm 9 Load balance and work stealing

```
Statically partition tasks and prefetch data (steps 1-3 of Alg. 2).
while a task remains in local queue
   upcxx::progress_thread_start()
   compute task
   upcxx::progress_thread_pause()
end while
update Fock matrix DArray via accumulate
for Every process p
   while a task remains in p's queue
      get remote blocks
      upcxx::progress_thread_start()
      compute task
      upcxx::progress_thread_pause()
   end while
   update Fock matrix DArray via accumulate
end for
upcxx::progress_thread_stop()
```

system featuring the Aries interconnect with a Dragonfly topology. Edison compute nodes contain two Intel® Xeon® E5-2695 processors with two-way Hyper-Threaded cores for a total of 48 "logical cores" per compute node. Each node has 64 GB memory.

For software, our experiments were run with the default Edison module for the Intel programming environment (5.2.56 and `-O3`), Cray LibSci for BLAS, LAPACK, and ScaLAPACK, and a custom build of the development version of GASNet preceding the 1.26.0 release. Global Arrays is the 5.4b version linked with LibSci (and peigs disabled).

2.3.4..1 Single Node Performance

We begin with a single-node performance exploration of UPC++ and OpenMP across sensible combinations of processes and threads. The data are shown in Figure 37, where each measurement is the total time spent in an SCF iteration (averaged across 5 runs). Black boxes are not possible to run on Edison

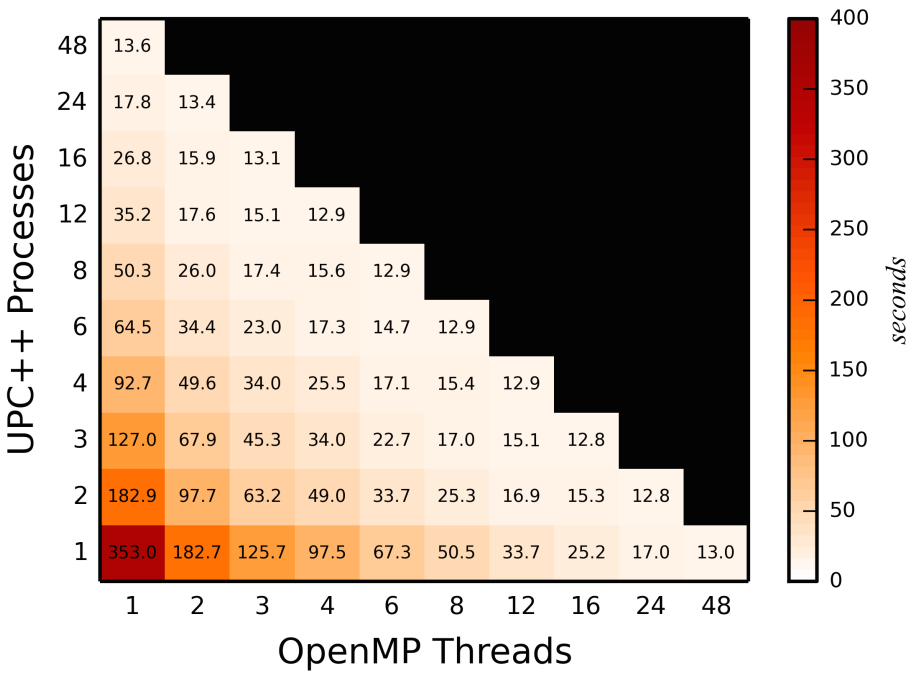


FIGURE 37. Single-node performance of the UPC++ HF application with UPC++ processes vs. OpenMP threads.

compute nodes because the job scheduler does not allow oversubscribing. The best performance is usually seen when using more threads relative to processes (except in the pathological case of only 1 process on 1 CPU socket). This is a particularly exciting result because we know that memory consumption is best with fewer UPC++ processes. Also, this property is favorable for performance on many-core architectures [46]. Other codes, such as NWChem, only exhibit good performance with 1 single-threaded process per core [177], which is a troubling characteristic in light of the prominent adoption of many-core architectures.

The best performance is seen along the diagonal, for which all executions exploit Hyper-Threading. The absolute best performance is with 2 processes (1 per socket) and 24 OpenMP threads per process. Therefore, all of our scaling experiments below are run with this configuration.

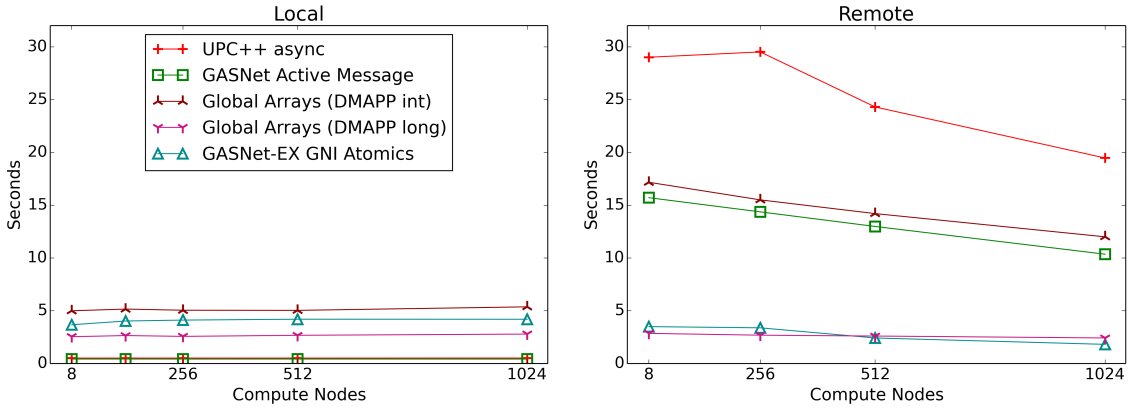


FIGURE 38. Flood microbenchmark with 1 million fetch/add calls per process - local (left) and remote (right) versions.

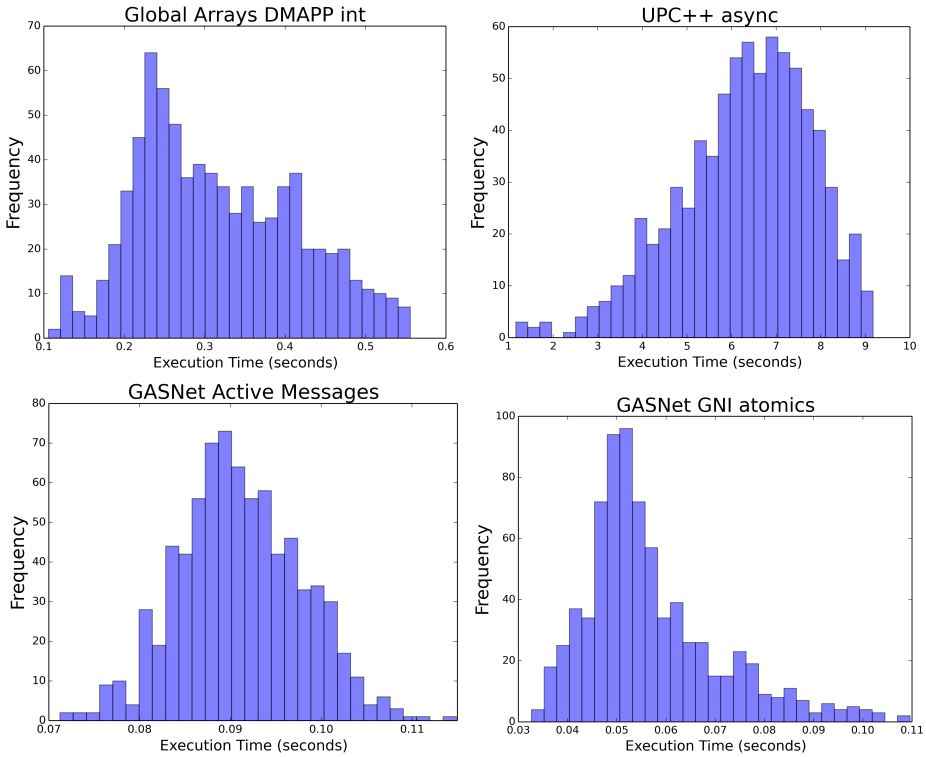


FIGURE 39. TAU profile data comparing the time in `nxtval` for Global Arrays (upper left), UPC++ `asyncs` (upper right), GASNet AM's (lower left), and GASNet GNI atomics (lower right). The experiment is $C_{40}H_{82}$ with 768 processes.

2.3.4..2 Load Balancing / Work Stealing

Our initial implementation of the task counters for load balancing/work stealing (described in Section 2.3.3..3) used UPC++ `asyncs`. However, TAU profile data, in the upper-right histogram of Figure 39, shows this method incurs too much overhead for such a lightweight operation. This motivated a microbenchmark analysis for the various implementation options described in Section 2.3.3..3. The microbenchmark is simple: each process owns a counter, and the benchmark ends when all counters reach 1 million. We create two different versions: one in which only *local* counters are incremented, and one in which random *remote* counters are incremented. The results are shown in Figure 38.

The most important feature of Figure 38 is that the GNI hardware-supported atomic operations on Aries show extremely good performance for *remote* fetch-and-adds, but relatively poor performance for *local* fetch-and-adds. This makes sense: GNI atomics must probe the NIC cache line for the counter, even if it is local. UPC++ `asyncs` and GASNet AM's access main memory and therefore exhibit less latency overhead. We note here that GTFock uses the `C_INT` type (4 byte integers) for task counters. However, in the ARMCI backend, this does not use the DMAPP implementation of hardware-supported atomics on Aries. It is a simple fix to use type `C_LONG`, which immediately shows much better remote fetch-and-add performance.

Due to the nature of Algorithm 2, the HF application does ∼90% local fetch-adds and ∼10% remote fetch-adds in high performing executions. Therefore, when comparing the in-application performance of GNI atomics to GASNet AM's in Figure 39, we do not see a drastic improvement. However, there is a slight benefit

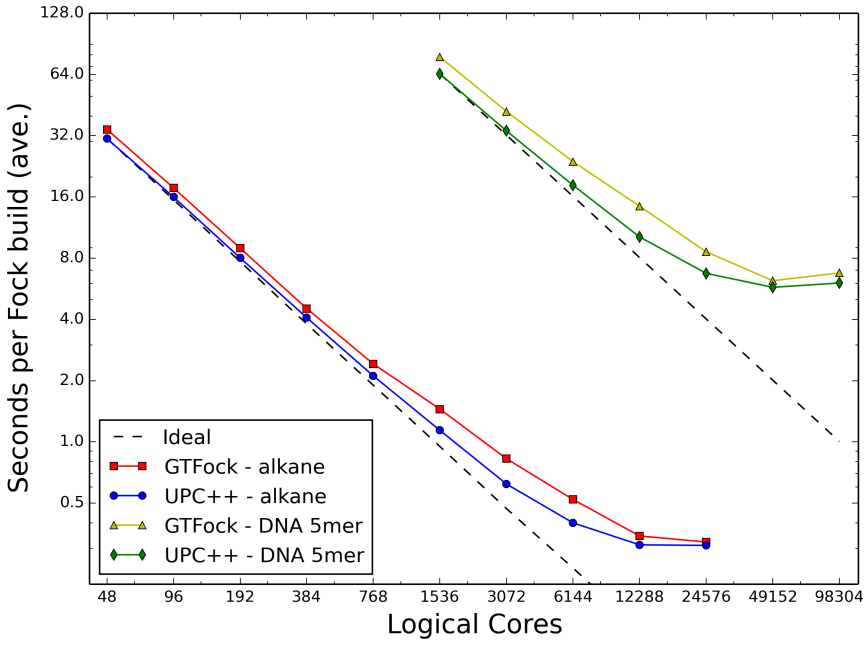


FIGURE 40. Strong scaling of UPC++ HF compared to GTFock and GA on Edison for two molecules: $C_{40}H_{82}$ and a DNA 5-mer. The "ideal" curves are projected with respect to the UPC++ execution at 1 node and 32 nodes, respectively.

to using the atomics overall. Therefore, in our scaling studies in section D below, we use the GNI implementation.

2.3.4..3 Progress and Attentiveness

We noticed in TAU profiles (not included due to space constraints) that *pthread_join()* consumes too much execution time when calling the stop function in every iteration of the task loop. This overhead is enough to degrade performance, particularly when the task granularity is small and there are a large number of tasks. The effect is particularly noticeable when running with a large number of processes, since the relative time of the overhead eventually overshadows the time spent doing the two-electron integrals. To alleviate this effect, we added the pause

feature described in Section 2.3.3..4. This optimization is included in our strong scaling measurements with a polling rate of 10 microseconds. We also use a PAR build of GASNet because we require its thread-safety and do not observe any effect on the performance of the HF application.

2.3.4..4 Strong Scaling

With the preceding optimizations made to the UPC++ HF application, we now compare performance to the original GTFock implementation using Global Arrays and ARMCI on the Edison cluster. We look at two molecules: an alkane polymer ($C_{40}H_{82}$) and a DNA 5-mer, both using the cc-pVDZ Dunning basis set. The alkane requires 970 basis functions and the DNA 5-mer requires 3,453. Both are run with the configuration that gives the best performance: 1 process bound to each socket and 24 OpenMP threads per process. The minimum of multiple runs (2-5) is reported to reduce noise due to system variation. In summary, UPC++ HF achieves 10-20% performance improvement, with the best gains occurring in the region just before strong scaling inevitably dwindles.

2.3.5. Related Work to DArrays

Related work explores relatively simple Hartree-Fock implementations in other PGAS languages like Fortress, Chapel, and X10 [12, 197]. The work from [12] presents interesting implementations in all three languages, but unfortunately deferred performance results to future work. The work from [197] reports performance measurements, but only for static partitioning, saying that the dynamic load balancing implementation does not scale, and using the X10 `async` for spawning tasks consistently runs out of memory.

Alternative communication infrastructures for Global Arrays have been explored in projects such as ARMCI-MPI [198]. Also, in [199] Gropp et al. present an edifying version of Global Arrays written directly with MPI-3, along with several versions of `nxtval`, including a threaded implementation.

Our UPC++ HF application has inherited the diagonalization-free purification technique for calculating the density matrix [54] from GTFock. The purification code is written in MPI, and the fact that our application uses it highlights the inter-operability of UPC++ and MPI. However, to keep things simple, our performance measurements have not included time spent in purification (except in Figure 37). The numerical effectiveness of this approach compared to classic diagonalization methods is left as future work. This is important because it affects the convergence behavior of the SCF algorithm, which will ultimately determine how well our UPC++ HF application will compare in performance to NWChem.

Related load balancing research includes resource sharing barriers in NWChem [164], inspector-executor load balancing in NWChem [18], exploiting DAG dependencies in the tensor contractions [51], and performance-model based partitioning of the fragment molecular orbital method [50]. Related work in optimizing runtime progress includes Casper, which was used to improve progress performance in NWChem [195]. It uses a process-based design that dedicates a few CPU cores to assist in communication progress of other processes, and it shows a performance benefit over traditional thread-based schemes with continuous polling. Future work might consider a comparison with our user-specified thread start/pause/stop approach.

2.3.6. DArrays Conclusion

Our results demonstrate that a highly tuned Hartree-Fock implementation can deliver substantial performance gains on top of prior algorithmic improvements. The performance analysis and optimization techniques presented in this work are also applicable to a broad range of use cases that would benefit from dynamic work stealing and a global view of distributed data storage. Looking forward, we believe that both novel algorithm design and sophisticated implementation optimization are crucial to scaling real applications on upcoming parallel systems with heterogeneous many-core processors, deep memory levels, and hierarchical networks.

To facilitate tiled operations on irregularly distributed arrays, we designed and developed the DArray library*, which is capable of applying tiled gets, puts, and user-defined functions across irregularly distributed arrays containing elements of any type. DArray will be an important and reusable building block for many other similar kinds of computational problems. In future work, we plan to further enhance it by: 1) adding conditional gets on DArray blocks, where only non-zero blocks are transferred after screening; 2) optimizing other initialization-only operations (e.g., DGEMM); and 3) including other features such as diagonalization and LU decomposition. Finally, our promising OpenMP performance measurements motivate a follow-up performance analysis in preparation for the deployment of the *Cori* system at NERSC, which will equip over 9,300 Intel Knights Landing processors with the Cray Aries high-speed Dragonfly topology interconnect.

*The UPC++, Multidimensional Array, and DArray code repositories are publicly available online at `https://bitbucket.org/upcxx`

CHAPTER III

MOLECULAR DYNAMICS: SCIENTIFIC WORKFLOWS

This chapter contains previously published material with co-authorship. Section 3.1 includes content from ICCS 2015 [21] and the Journal of Computational Science 2015 [22]. This work was a collaboration with the Guenza Lab. I wrote all the workflow code (at the time of the 2015 publications), which wraps several existing Fortran/C programs and the LAMMPS application to create an automated workflow pipeline. Prof. Guenza wrote most of the background section of the paper, and I wrote the rest of the prose and gathered experimental data. Prof. Malony provided guidance and several minor paper edits.

Section 3.2 includes content from PDP 2016 [23]. This is another collaborative project with the Guenza Lab. I wrote all the regular grid backmapping code and analysis scripts, and prepared all experimental data. I also wrote most of the paper with several minor edits from Prof. Malony. Prof. Guenza wrote much of the IECG background section and also provided several minor edits.

Section 3.3 includes unpublished content that is in submission at the time of this writing. This work is another collaborative effort with the Guenza Lab. It is a journal paper extension from the PDP conference paper above with approximately 30-40% new content. I gathered all the new experimental data and wrote most of the new prose. However, the math that verifies the CG intramolecular harmonic period is proportional to $1/\sqrt{n_b}$ has contributions from Josh Frye, and the model for predicting the number of interactions given the granularity was developed by Richard Gowers.

3.1 Coarse-Grained Workflows

3.1.1. Fast Equilibration Workflow Introduction

Despite considerable advancements in hardware and software technologies for supporting large-scale molecular simulations, computational chemists are confined to simulating relatively small systems compared to most laboratory experiments and real world bulk measurements. This constraint quickly becomes apparent in the simulation of polymer melts, which is important for materials science applications. Polymers are macromolecules that consist of repeating units of monomers (such as CH_2) that form long molecular chains. When simulating polymeric systems, many interesting properties depend on the chain length [200], such as the boiling/melting points, the viscosity, and the glass transition temperature. Even with a small number of molecules, however, it is computationally expensive to simulate chains with more than 10^5 monomers each, which is a reasonable chain length to study. Limited memory space also constrains simulations to 10^6-10^{10} total atoms, even on large allocations of modern supercomputers. Considering that a drop of butane contains approximately 10^{20} atoms suggests that these capabilities do not come close to bulk experiments in the laboratory.

Defining a new representation of the polymer as a chain of soft colloidal particles greatly reduces the amount of information to be collected and controlled, which speeds up the simulation. It is well known that modeling fewer colloidal particles with an appropriate potential decreases the degrees of freedom and computational requirements by an amount proportional to the granularity [92]. The representation of a polymer as a chain of soft blobs also allows the chains to

more easily cross each other, decreasing the required time for the simulation to find the equilibrium structure. However, the information on the molecular local scale needs to be restored at the end of the simulation to account for properties on the *united atom* (UA) scale. In a nutshell, it is important to alternate between the *coarse-grained* (CG) representation (which speeds up the simulation) and the UA representation (which conserves the local scale information). By quickly switching back and forth from UA to CG, we open doors to new studies of polymeric systems while maintaining simulation accuracy and efficiency. While optimizing the computational performance of CG codes is important, without a way to incorporate UA-level detail, CG efficiency has relatively less value. Therefore, this chapter focuses on integrating an approach for conducting simulations that exploit both CG efficiency and UA accuracy.

Unfortunately, it is not trivial to automate the transformation between the CG and UA representations for general sets of input parameters. Accomplishing the overall task involves multiple processing steps, with different applications, programming languages, naming schemes, and data representations. This issue is common to many other scientific software environments and is collectively referred to as the scientific workflow problem [201]. In short, the collection of all the required steps to conduct an overall research study comprises a *workflow* that may consist of several large simulations, real-world experiments, human intervention and analysis, and more. In this paper, we present a lightweight and customizable approach for creating an effective scientific workflow in the context of our CG/UA simulation experiments. We discuss techniques that are general to other computational problems and explore new ideas that are not prevalent in other more heavyweight workflow toolkits.

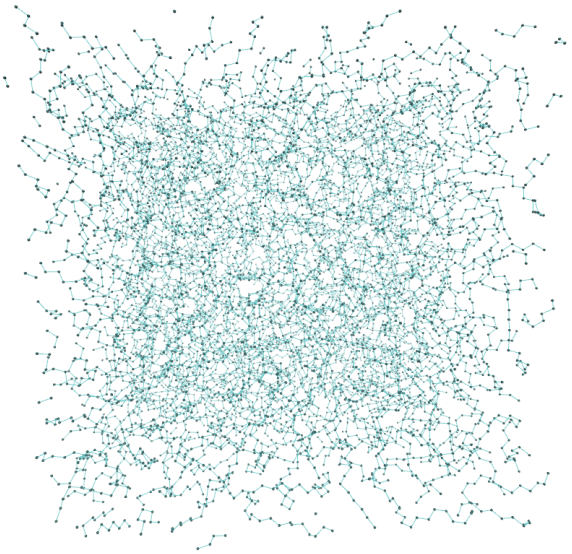


FIGURE 41. UA representation with 80 chains 120 monomers per chain

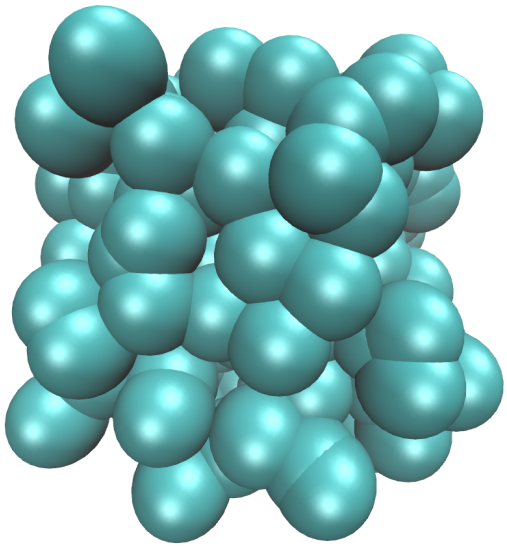


FIGURE 42. The corresponding CG representation with 3 sites per chain (and 40 internal monomers per site). CG spheres appear hard, but are soft with long range effects.

The paper is organized as follows: Section 3.1.2. contains background information regarding the CG approach and integral equation theory, Section 3.1.3. discusses the design and implementation of the fast equilibration workflow,

Section 3.1.4. presents our experiments in evaluating the workflow, Section 3.1.5. discusses related work, and Section 3.1.6. contains concluding remarks.

3.1.2. Integral Equation Coarse-Graining Background

This section briefly reviews the coarse-grained approach based on integral equation theory. We consider a homopolymer fluid (in which all monomers are the same type) with molecular number density, ρ_m, consisting of n polymer chains. Some number of monomer sites, N, makes up each polymer chain, where each site is usually taken to be either a CH, CH_2 or CH_3 group. This bead-spring description is the UA simulation model. Within the UA description, the Polymer Reference Inter Site Model (PRISM) site-averaged Ornstein-Zernike equation relates the relevant pair correlation functions in Fourier Space, [202]:

$$\hat{h}^{mm}(k) = \hat{\omega}^{mm}(k)\hat{c}^{mm}(k)[\hat{\omega}^{mm}(k) + \rho\hat{h}^{mm}(k)] \tag{3.1}$$

where the "mm" superscript denotes monomer-monomer interactions and $h^{mm}(k)$ is the Fourier transform of the total correlation function, $h^{mm}(r)$. In fact, $h^{mm}(r) = g^{mm}(r) - 1$, where $g(r)$ is the well known radial distribution function. Also from Eqn.(3.1), $c^{mm}(k)$ is the direct correlation function, $\omega^{mm}(k)$ is the intra-chain structure factor, and ρ is the monomer site density, given as $\rho = N\rho_m$. The CG representation can be fully represented as a function of the physical and molecular parameters that define the atomistic description. The key quantity to be solved in the CG representation is the potential, which must be included as an input to the molecular dynamics (MD) simulation of the polymer melt in the reduced CG representation. This potential has been derived analytically using an integral equation formalism. In the CG model, each chain contains an arbitrary number n_b

of chemically identical blocks, or "blobs". Each block contains a sufficiently large number of sites so that we can utilize Markovian properties of the internal chain's random walk. The integral equation coarse-graining approach (IECG) allows us to determine an analytical solution to the potential when $N/n_b = N_b \approx 30$. At that scale the structure of the chain follows a random walk, and the distribution of the CG units along the chain is Markovian, so that the IECG formalism can be solved analytically [97]. In a macromolecular liquid, the random distribution of the CG units is a general property of any molecule when sampled at large enough lengthscales [203].

The IECG model conserves thermodynamics while accurately reproducing the structure of polymer liquids across variable levels of resolution [97, 102, 103]. Insights from the IECG theory provide reasonable justifications for some of the advantages and shortcomings of coarse-graining methods in general [99, 204].

The potential is solved analytically in the mean spherical approximation, which is valid for low compressible polymer liquids [205]. In the limit of large separations in real space, where $r >> 1$ (in units of the polymer size), the potential is approximated as [104, 102]

$$\begin{aligned} V^{bb}(r) \;\approx\; & k_B T\Big[\left(\frac{45\sqrt{2}N_b\Gamma_b^{1/4}}{8\pi\sqrt{3}\sqrt[4]{5}\rho_m R_{gb}^3}\right)\frac{\sin(Q'r)}{Q'r}e^{-Q'r} \\ & -\left(\frac{\sqrt{5}N_b}{672\pi\rho_m\Gamma_b^{1/4}R_{gb}^3}\right)\big[(13Q^3(Q'r-4))cos(Q'r) \\ & +\left(\frac{945+13Q^4}{\Gamma_b^{1/4}}\right)r\sin(Q'r)+\frac{945r}{\Gamma_b^{1/4}}cos(Q'r)\big]\frac{e^{-Q'r}}{Q'r}\Big]\;, \end{aligned} \tag{3.2}$$

where $Q' = 5^{1/4}\sqrt{3/2}\Gamma_b^{-1/4}$ and $Q \equiv Q'\Gamma_b^{1/4}$. The key quantity of interest here is the universal parameter $\Gamma_b = N_b\rho|c_0|$, which is defined once we decide the level of coarse-graining, N_b, as well as the molecular and thermodynamic parameters of our system. This quantity also depends on the direct correlation function at $k = 0$,

c_0, which is in principle not known. However, this function relates to the potential between atomistic units and the isothermal compressibility of the liquid, so it can be determined numerically or from experiments.

When compared with full atomistic simulation (i.e., UA) the CG simulations that use the potential of Eq. 3.2 show quantitative consistency in the structure and thermodynamics. The CG potential is in this way fully transferable, and it can be conveniently applied in MD simulations of polymer melts, at the chosen thermodynamic and molecular parameters, with computational gain. The ability of CG models to maintain thermodynamic and structural properties while varying the coarse-graining resolution is important when one develops computational techniques with variable resolution. It allows the computational time of the MD simulations to be controlled by changing the resolution as needed. This notion motivates the development of a workflow that supports transformation between the UA and CG descriptions, which dynamically adjusts CG resolution at desired simulation times and spatial regions, while maintaining simulation accuracy and equilibrium through transitions to united-atom simulation.

3.1.3. The Fast Equilibration Workflow

This section describes the fast equilibration workflow, which consists of a series of computational programs and analyses that comprise an overall application for quickly stabilizing a polymeric liquid. Figure 43 shows the 7 high-level stages involved in the workflow, each of which may involve multiple processing steps. Each step is accomplished by one or more programs, applications, or simulations. Before this work, these steps each required manual intervention by a researcher, but now they are automated by our fast equilibration workflow.

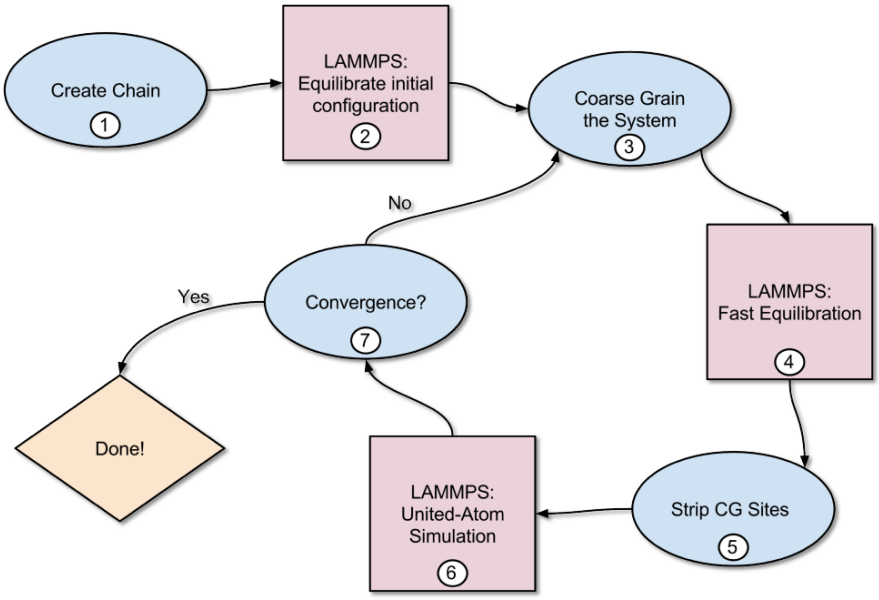


FIGURE 43. The UA↔CG Workflow. Blue circles (1,3,5,7) represent stages of custom programs that either generate coordinates and potentials, transform data for input into LAMMPS, or conduct convergence analysis. Red squares (2,4,6) represent parallel LAMMPS MD simulations executing via MPI. Our workflow system automates this process for a given set of input parameters.

3.1.3..1 Workflow Design

Stages 1 and 2 from Figure 2 initialize the workflow by generating a geometric configuration and equilibrating the system just enough to remove instabilities. Stage 1 randomly generates a polymer system of n chains, each with N monomers. Each chain begins at a random coordinate, and the subsequent N monomers bond randomly onto the enclosing spherical surface. Given a desired n, N and ρ, we generate a collection of random chains within a simulation box of volume V with periodic boundary conditions and length, L, such that $V = L^3$. We also include periodic boundary image flags to help reconstruct the polymer chains later. Because randomly generated configurations likely contain regions of high strain and numerical instabilities due to overlapping chains, we carefully adjust

the configuration of these areas. Stage 2 accomplishes this with a brief LAMMPS simulation in which chains slowly drift apart via a soft repulsive potential for 10 picoseconds of simulation time. Then, we minimize energy via a Lennard-Jones potential with a series of short executions, in which the system runs for 1,000 timesteps with an incrementally increasing amount of time per step (e.g., 0.02, 0.08, 0.25 femptoseconds, etc).

Stages 3 through 7 constitute an iterative strategy that alternates between the UA and CG representations towards equilibration. Stage 3 determines the center of mass for each soft-sphere that encompasses a group of N_b monomers. The center of mass coordinates are the "fictitious sites" of the soft colloids from which the multi-block potential is derived. During this stage we also store the internal monomer configurations within each block for rigid tracking within the upcoming CG simulation (see Section 3.1.3..2.2). The multi-block potential, which depends on parameters such as temperature, density, and number of blocks per chain, is generated at runtime during the first iteration [97]. The ensuing LAMMPS execution in Stage 4 simulates the CG system with the multi-block potential, treating the internal monomer chains within a block as coupled rigid bodies. In our experiments below, Stage 4 runs approximately 60,000 timesteps at 3 femtoseconds per step, although this is easily customizable.

After the CG simulation, Stage 5 restores the UA description by applying the saved internal chain coordinates to their new location within each updated block position. To mitigate any unphysical bond lengths within chains, Stage 6 runs a short simulation within the UA description using the same Lennard-Jones pair potential from Stage 2. Finally, in Stage 7, we determine whether or not

---

LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator) is a classical molecular dynamics code. `http://lammps.sandia.gov`

to perform another iteration. While there are several alternatives for evaluating convergence effectively, we choose to consider the total correlation function, $h^{mm}(r)$, because of its relevance in deriving system thermodynamic properties (see Section 3.1.4..2). Finally, if we determine that the system has satisfactorily equilibrated, the workflow is complete.

3.1.3..2 Workflow Implementation

This section describes in more detail the implementation of the workflow design discussed in the previous section. Our overall approach involves bundling each set of Fortran and C programs that encapsulate Stages 1, 3, 5, and 7 (in Figure 2) with Python scripts. Then, using a standard Python argument parsing system, `argparse`, to define all the parameters in Table 3 at launch time, we subsequently pass the parameter bundle (as a Python object) through the entire workflow program. During initial development of the workflow, the computational and analytical programs were hard coded to contain the parameters, filenames, and directory paths. In addition, no record of launch configurations were stored (such as number of MPI processes, the cluster name, or execution time). After easily restructuring the internal programs and procedures to extract relevant values of the parameters from the `argparse` interface, a fully automated system for doing UA/CG polymer equilibration is in place. For instance, one can now launch the entire workflow with a single command:

```
fast_equil -nchain 350 -nmonomer 192 -sitesperchain 6 -temperature 450

        -bondlength 1.55 -mono-mass 14.0 -cnot -9.7 -timestep 1.0 ...
```

Certain flags, such as `-viz`, can provide outlets for user analysis by stopping the workflow and presenting a visualization of a desired quantity, such as $h^{mm}(r)$, or

The source code is available at `https://github.com/mguenza/Fast-Equilibration-Workflow`

Input Parameter	Create Chain	Coarse Grain		Strip CG	Converge	LAMMPS
	`create_chain`	`cg_chain`	`multiblock`	`strip_cg`	`hmmr`	`lmp_run`
ρ (density)	yes	no	yes	no	yes	no
T (temperature)	no	no	yes	no	no	yes
n (# of chains)	yes	yes	no	yes	yes	yes*
N (monomers)	yes	yes	yes	yes	yes	yes*
n_b (# of blocks)	no	yes	yes	yes	no	yes*
c_0 (dc constant)	no	no	yes	no	no	no
L (box length)	no	yes	yes	no	yes	yes
p (MPI procs.)	no	no	no	no	no	yes
Experiment ID	no	no	no	no	no	no
...						

TABLE 3. Input parameter dependencies for the stages of UA/CG equilibration. Our workflow system obviates tracking dependencies by implicitly passing all parameters throughout each stage, forming a unique parameter space for each overall experiment. (Items marked with * are not directly passed to LAMMPS, yet they determine the total number of input atoms.)

the simulation box itself. If flags are not specified, they are set to default values as disclosed by the `--help` reference flag.

Upon each launch of a workflow instance, the parameter set, execution timestamp, and path to output datafiles are saved to a database management system (DBMS) for historical reference and provenance. Our current implementation interfaces the Python workflow runtime code with a local SQL-based DBMS. Future work will consider how to extend this feature towards collaborative efforts by exploring remote database interactions and knowledge awareness/extraction.

The previous discussion is related to defining the parameter space within the various components of the workflow in an automated fashion. However, there are some peculiarities in the LAMMPS side of the workflow in Stages 2, 4, and 6 that deserve special attention. This is the subject of the following subsections.

3.1.3..2.1 Generating LAMMPS Input and the Multi-Block Potential at Runtime

Our initial implementation of the multi-block potential utilizes the `table` feature of LAMMPS for doing bond, angle, and intermolecular calculations. Although a direct evaluation of the potential may feasibly achieve better performance, the `table` approach establishes the overall workflow and, if necessary, is substitutable in future versions. Because the multi-block potential depends on the parameters ρ, T, N, n_b, c_0, and L, we generate the potential at runtime. Fortunately, this takes less than 5 seconds on a standard processor, which is negligible compared to a typical cycle of the workflow, which can take several hours even on hundreds of processors.

Another interesting feature of our workflow environment is that we generate LAMMPS input files via a template processor system. Originally, template processors were designed to be used to create dynamic websites in which content is generated on-the-fly by filling an HTML template file with runtime data [206]. This avoids the error-prone, tedious, and inflexible approach of generating content with code in this manner:

```
print("<html>\n <body>\n <p>" + str(mycontent) + \
                "</p>\n </body>\n </html>\n")
```

Template processing, on the other hand, renders the following `index.html` input template:

```
<html> <body> <p> @user.name : @user.age </p> </body> </html>
```

with standard programming objects like so:

```
template.render("index.html", {"user": Jane})
```

An intriguing analogy existed when we initially generated LAMMPS input files,

```
print("minimize "+ str(etol) +" "+ str(ftol) +" "+ str(iter) + ... )
```

So, with a LAMMPS input template (`in.lammps`),

```
minimize @etol @ftol @iter @maxeval ...
```

we can render the simulation input files in the same fashion:

```
parameters = {"etol": 1e-4, "ftol": 1e-6, "iter":100, "maxeval": 1000}

template.render("in.lammps", parameters)
```

Template processing is central to the popular *model-view-controller* (MVC) architectural pattern. It is clear that many scientific workflows fit the same design pattern, where the data is the model, the workflow is the controller, and the simulation input/output is the view. Just as the MVC paradigm emphasizes a separation of concerns between database engineers, backend programmers, and designers [207]; we see MVC applying equally as well to the data scientists, software engineers, and domain specialists that represent the stakeholders in any effective scientific workflow. It is prudent to note the ubiquity of managing scientific simulations through input files (LAMMPS, GROMACS, NWChem, Gaussian, GAMESS, OpenMC, Nek5000, and many more), and that controlling the parameters dynamically with a clear separation of implementation concerns exposes new research possibilities and opportunities for productivity.

3.1.3..2.2 Transforming from the CG to the UA Description with the POEMS Library Our approach for transforming between the UA and CG descriptions involves storing the coordinates internal to blocks before the CG simulation and treating them as rigid bodies. This is done using the *Parallelizable Open Source Efficient Multibody Software* (POEMS) library included with LAMMPS. In POEMS, when computing the total force and torque on rigid bodies, the coordinates and velocities of the atoms update such that the collection of bodies move as a coupled set [208]. When transiting from UA to CG, the definition

of the rigid blocks and their internal atomic identities occurs in Stage 3 of the workflow simultaneously with the generation of the CG description. When mapping from CG back to UA in Stage 5, the final state of the simulation from Stage 4 is used to restore the rigid coordinates. After stripping the fictitious sites from the simulation box, Stage 6 commences with the appropriate timestamp update to keep the overall simulation time consistent.

In our simulations, we observe that calculating the force and torque on rigid bodies comes at a performance cost. However, more efficient methods for carrying the rigid coordinates exist, for instance, by either 1) ignoring rigid body calculations or 2) regenerating random internal configurations between stages of the workflow. Because our approach only requires that chains' random walks be Markovian, approach 2 may bear fruit. Our preliminary studies show that simulations applying methods 1 and 2 exhibit speedups over UA on the order of the granularity factor. Future work will consider in detail the performance of alternative methods for managing the internal configurations.

3.1.4. Fast Equilibration Workflow Experiments

3.1.4..1 Experimental Setup and Evaluation

We conducted our scientific workflow evaluation on the ACISS cluster located at the University of Oregon. Experiments are run on the 128 generic compute nodes, each with 12 processor cores per node (2x Intel X5650 2.67 GHz 6-core CPUs) and 72 GB of memory per node. This is a NUMA architecture with one memory controller per processor. ACISS employs a 10 gigabit Ethernet interconnect that connects all compute nodes and storage fabric. The operating

system is RedHat Enterprise Linux 6.2, and MPICH 3.1 is used with the `-O3` optimization flag.

Overall, the overhead introduced by stages 1, 3, and 5 is negligible (far less than 1% of the total workflow execution time). Due to space limitations, we defer computational performance measurements for another publication. We instead focus on the correctness and validity of our approach as verified by the radial distribution function.

3.1.4..2 Radial Distribution Function Analysis

Analysis of the radial distribution function is critically important in the evaluation of the validity of a simulation result. Given this function and assuming pairwise additivity, all the thermodynamic properties of the liquid may be calculated [209]. This function, often called $g(r)$, is defined as

$$g(r) = \frac{1}{\rho}\left\langle \frac{1}{N}\sum_{i}^{n}\sum_{j\neq i}^{n}\delta(\vec{r}-\vec{r}_{ij})\right\rangle$$

and $h(r) = g(r) - 1$ (we introduced $h(r)$ in Section 3.1.2.). We calculate $h(r)$ at the convergence step of each workflow iteration, and compare it to the previous determination of $h(r)$. If the mean percentage error is less than a user-defined threshold, then the workflow completes. If available, we can alternatively compare to a long-running "full UA" simulation (with no CG). To clearly specify that we calculate this quantity in the UA (monomer) representation (*not* including the block sites), we henceforth use the mm superscript, denoting this function as $h^{mm}(r)$.

Figure 44 shows $h^{mm}(r)$ for several phases of the workflow. In this experiment, the full UA "gold standard" ran for 1.25 nanoseconds of simulation

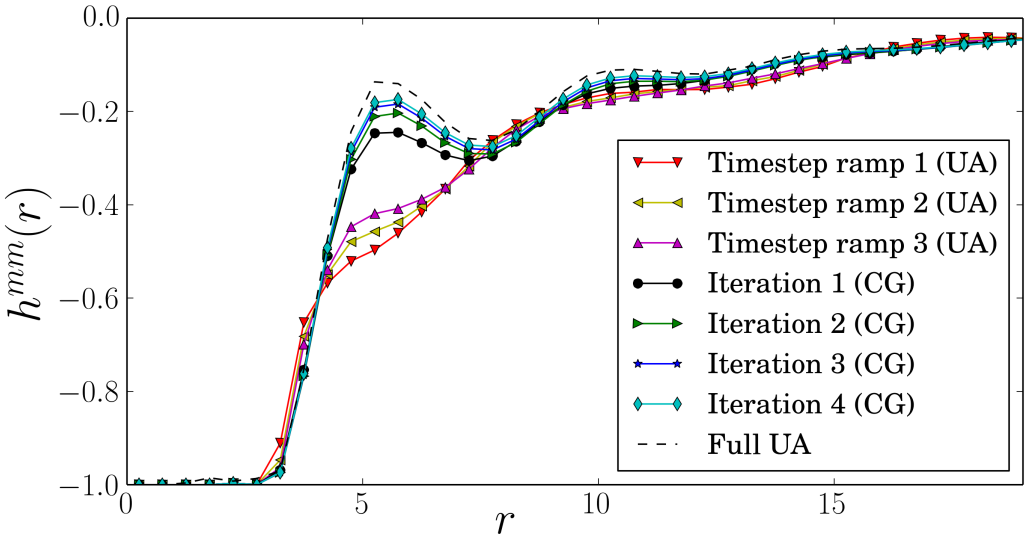


FIGURE 44. The total correlation function, $h^{mm}(r)$, for a workflow experiment with ρ=0.03355 sites/Å^3, T=450K, n=350, N=192, n_b=32, c_0=−9.67344, L=126.05Å, and p=96 MPI processes on the ACISS cluster. The bottom curves show 3 ramping steps from Stage 2 of the workflow (as in Figure 2) and the top curves show 4 iterations through Stages 3-7. Satisfactory convergence occurs (both in terms of $\Delta h_i^{mm}(r)$ and w.r.t. full UA) after 4 workflow iterations.

time, and is shown as the black dotted line. The 4 CG iterations were run for a total of 0.72 nanoseconds, and each UA transition stage (Stage 6 from Figure 2) ran for 0.0125 nanoseconds. The "Timestamp Ramp" steps correspond to Stage 2 with total simulation times of 20, 80, and 250 femtoseconds, respectively. The figure clearly shows that the system converges quickly towards the correct liquid character.

Without the automated workflow, creating Figure 44 would have required too many human hours and too much tedious intervention between simulations to have been practical. Furthermore, experiments of this nature are easy to launch with different input parameters through a job scheduler, such as PBS. It is no more difficult to conduct an experiment with a few simulations than with hundreds of simulations, except for the time it takes to execute.

3.1.5. Related Work to Scientific Workflows

Related research supports the notion that the CG representation equilibrates more quickly than fully atomistic replicas, and that re-expressing a melt in the UA description can more completely equilibrate the system [92]. While our verification of $h^{mm}(r)$ is encouraging for verifying correctness, this function is known to be relatively insensitive to certain geometric flaws, particularly in bonding. For instance, it may be possible to have a well-matched $h^{mm}(r)$, but still have occasional bond lengths that are unphysical. Instead, Auhl [210] evaluates $\langle R^2(n)\rangle/n$ and shows that this phenomenon may occur and can be fixed. Future work will examine $\langle R^2(n)\rangle/n$ in Stage 7 of the workflow (which may be less expensive to compute than $h^{mm}(r)$).

Research and development on scientific workflows is pervasive. Some of the more popular frameworks include Kepler (with recent notable applications in drug design [211] and support of distributed data-parallel patterns, such as MapReduce [212]) Pegasus [213], and Taverna [214]. Other related work focuses on the intricacies of data modeling and dataflow in scientific workflows [42]. Instead of committing to a full-blown workflow framework up-front, this work has focused on the advantages of using a considerably lightweight system for managing input parameters while benefiting from simple data provenance and template processing. As far as we are aware, template processing capabilities within other scientific workflows is limited, and is not to be confused with reusable "workflow templates", which is a powerful plug-and-play concept found in any good workflow suite. Our future work may consider porting our implementation to a more powerful workflow system, but we are so far content with the portability, ease of customization, lack of a GUI, and template processing features within this work.

3.1.6. Fast-Equilibration Workflow Conclusion

Coarse-graining methods benefit from reduced computational requirements, which expands our capabilities towards simulating systems with a realistic number of atoms relative to laboratory bulk experiments. However, without integrating local scale UA information, configurations cannot equilibrate as completely. This section focused on a workflow based solution for exploiting CG efficiency and UA accuracy by iterating between UA and CG representations towards convergence. By utilizing a lightweight scheme for propagating parameters, controlling simulation input files via template processing, and generating multi-block potentials in the CG description, we have constructed an automated system for conducting large scale experiments of polymer liquid systems. We have verified the correctness of the approach of mapping between the atomic and coarse-grained descriptions by comparing the radial distribution function of a long-running atomic simulation with a series of iterations through the workflow.

The fast equilibration workflow system enables the running of experiments that were previously not possible, such as parameter sweeps across different densities, CG granularities, temperatures, and more. Our simple workflow system is based on Python, and incorporates support for parallel (MPI) simulations, data provenance, and template processing. Future work will examine and optimize the computational efficiency, introduce more powerful workflow features, and explore more thorough verifications of correctness.

3.2 Backmapping between Representations

3.2.1. Introduction to Backmapping within IECG

On the largest modern supercomputers, molecular dynamics (MD) simulations of polymer systems contain billions of atoms and span roughly a few nanoseconds of simulation time per week of execution time. Unfortunately, most macromolecular processes of interest contain many orders of magnitude more particles and often bridge microsecond or even millisecond timescales or longer. These include phenomena like phase separation in polymer blends and composite materials [90], polymer crystallization, and glass formation and aging [91] to mention just a few. Despite our pervasive access to massively parallel computers, full *united-atom (UA)* simulations do not come close to representing real-world polymer systems (see Figure 45), because they are too computationally expensive and slow. This makes direct comparison between experiments and simulations impossible as the two systems are dynamically different. For these reasons, scalability of MD simulations is paramount.

Simply put, we require new approximation methods that capture the relevant physics and chemistry while requiring fewer computational resources. The most promising approach is the *coarse-graining (CG)* method, in which groups of atoms are represented as one collective unit. CG has proven to be valuable for eliminating unnecessary degrees of freedom and tackling the scaling complexity of larger problems [92]. The key issue is how to simultaneously maintain solution accuracy and high performance. With the alternation of CG and atomistic simulations enabled by the workflow presented in this section, it is possible to quickly equilibrate the system during the CG simulation, then reintroduce local

details into a UA simulation, taking advantage of the performance of the CG simulation and the realism of the UA representation.

The *Integral Equation Coarse-Grained (IECG)* model by Guenza and colleagues [96, 97, 98, 99, 101] adopts an analytically-derived potential and dramatically improves spatial and temporal scaling of polymer simulations, while accurately preserving thermodynamic quantities and bulk properties [102, 103, 104]. Several numerical techniques and force fields exist for performing coarse-grained simulations [105, 106, 107]. However, these methods generally preserve either structure or fully preserve thermodynamics, but not both. As a result, only a small level of coarse-graining is typically adopted to limit the errors in the simulated structure and thermodynamics. In contrast, our work adopts the analytical approach offered by IECG theory, because it recovers crucial structural and thermodynamic quantities such as the equation of state, excess free energy, and pressure, while enabling a much higher level of coarse-graining and the corresponding gains in computational performance.

Although CG polymer physics is a mature field, little has been done to analyze the performance benefits of CG versus UA representations. While it is clear that CG will exhibit computational gains, does it strong scale to as many

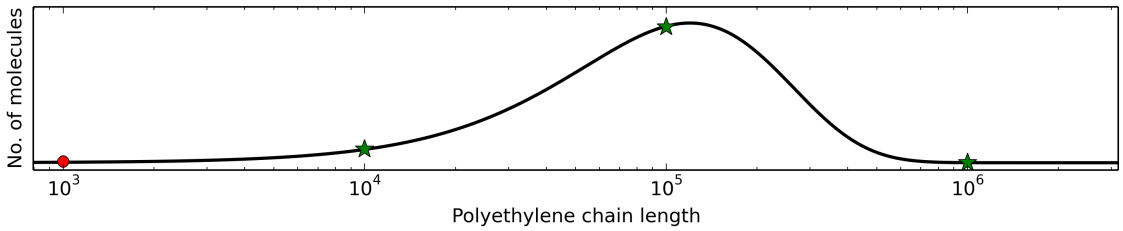


FIGURE 45. A representation of the average polyethylene chain length determined by chromatography experiments [95]. Most studies are limited to very short chain lengths ($\leq$ 1000) due to the prohibitive cost of UA simulations, but this project freely explores the realistic systems with 10^4 to 10^6 monomers per chain.

processors as the corresponding UA simulation? Likely not, because CG tracks far fewer overall particles, sometimes by orders of magnitude. Accordingly, the scalability of CG simulations likely depends on the granularity factor, e.g., the number of UA coordinates a CG unit represents.

One reason for the lack of performance analysis in CG research is likely due to the inherent complexity and variability in executing useful CG simulations. For instance, CG representations are generally based on a corresponding (usually unequilibrated) UA geometry. A helper program, which is usually independent of the MD simulation framework, maps the UA representation into the CG representation. Furthermore, after the CG simulation equilibrates, we usually desire a "backmapped" geometric description of the equilibrated system in a UA representation to restore properties at the local molecular scale. The amalgamation of these processing steps encompass a scientific workflow for conducting CG simulations, shown pictorially in Figure 46. In order to benefit most from CG computational gains, the coupled processing stages of this workflow must be high performance, low overhead, and asynchronous whenever possible.

In this section of the dissertation, I present such a scientific workflow that integrates the analytical IECG approach for calculating CG forces with new high-performance techniques for mapping back and forth between the UA and CG descriptions in LAMMPS. This workflow benefits from the performance of CG, while maintaining the accuracy of the full-atom representation. Our workflow optimizations legitimize our comparisons between UA and CG execution times. Scaling results show speedups of up to 12x at 3,072 cores on the Hopper system at NERSC. Furthermore, our workflow opens possibilities for the validation of polymeric systems that have never before been simulated at realistic chain lengths.

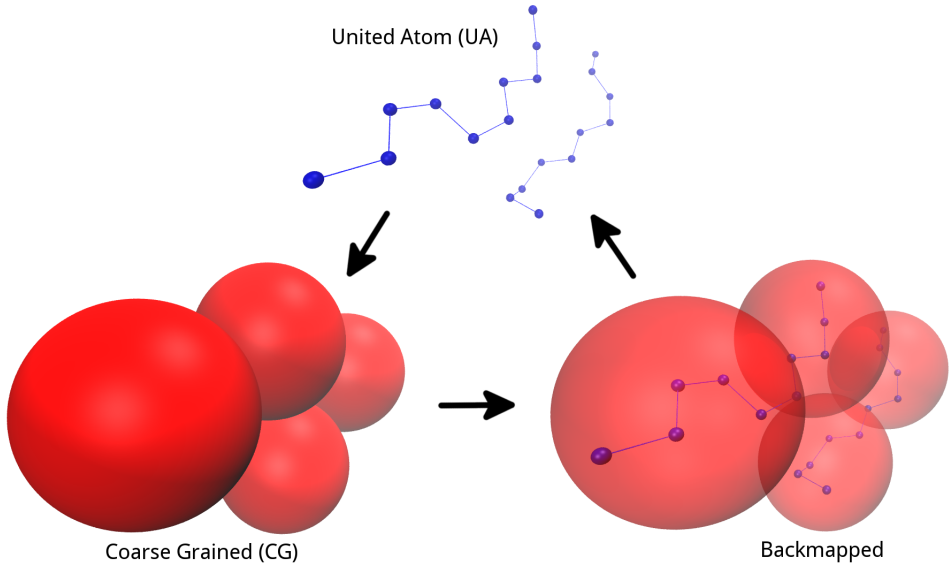


FIGURE 46. The high-level progression of a UA↔CG workflow. The CG representation is calculated from UA coordinates, and the UA representation is recovered by solving a backmapping problem (described in Sections 3.2.2..3 and 3.2.3.). CG spheres appear hard, but are soft with long range effects.

3.2.2. Background: Backmapping and the UA↔CG Workflow

This section provides background information and context for understanding the motivation, design, and implementation of our scientific workflow that manages CG simulations of polymer melts. Section 3.2.2..1 describes our analytically based CG methods. Sections 3.2.2..2 and 3.2.2..3 examine the UA↔CG workflow and two crucial optimizations for ensuring efficient execution.

3.2.2..1 Background: Integral Equation Coarse-Graining

This section briefly reviews the Integral Equation Coarse-grained approach [96, 97, 98, 99, 101]. We simulate a homopolymer fluid (in which all monomers are the same chemical species) with monomer number density ρ_m, consisting of n polymer chains at temperature T. Each polymer contains N

monomers, each located at a three-dimensional Cartesian coordinate, which we call a monomer *site*. At the atomistic resolution, MD simulations were performed with LAMMPS using the UA model, where each site is either a CH, CH_2 or CH_3 group. In the coarse-grained representation, each polymer is described as a chain of soft particles, or spheres, and each sphere represents a number of monomers N_b. The number of spheres per chain is given by $n_b = N/N_b$. Models that have fewer spheres per chain track fewer degrees of freedom than more fine-grained models, which promotes computational efficiency.

The IECG is an integral equation formalism that builds on the Ornstein-Zernike [215] equation and the PRISM atomistic theory [216]. The IECG model gives a complete description of the coarse-grained system as consisting of the effective intermolecular potential between coarse-grained units on different chains, effective bond potentials, and angle potentials designed to preserve Gaussian statistical distributions [217] and by postulating that the effective intermolecular potential must act between monomers farther apart on the same chain [102, 103, 104]. The intermolecular pair potential acting between CG units is fully represented as a function of the physical and molecular parameters that define the system, which are N_b, ρ_m, T, the liquid compressibility, and the direct correlation function c_0.

In the specific regime of $N_b \geq 30$ it is possible to derive an analytical form of the potential. At that scale the structure of the chain follows a random walk, and the distribution of the CG units along the chain is Markovian. This is a general property of the macromolecules [217] when sampled at large enough lengthscales. It should be stressed that in the IECG papers, the analytical potential serves as an approximation, under reasonable assumptions, for the numerical potential that is

used in simulations. Having an analytical potential allows one to understand the scaling behavior of the potential with structural parameters, as well as to estimate thermodynamic quantities of interest. The relevant equations are quite sizeable and beyond the scope of this work, but the complete analytical forms can be found in previous publications [103].

Using this potential we perform simulations of the CG systems, and then compare thermodynamic quantities and structural quantities of interest from these simulations with UA simulation data. The agreement between CG and atomistic descriptions is quantitative, where the direct correlation contribution at large distances, $c(k \rightarrow 0) = c_0$, is the only non-trivial parameter. It is evaluated either from experiments or from theory. Consistency for structural and thermodynamic properties is observed in all comparisons between numerical solutions of the IECG, analytical solutions, UA simulations, and mesoscale simulations [103].

3.2.2..2 Background: The UA↔CG Workflow

The CG representation enables simulations to explore larger chemical systems because it exposes far fewer degrees of freedom than the UA representation. CG can also explore longer timescales because it does not suffer from the geometric constraints within UA systems, such as those caused by entanglements that prohibit efficient dynamics. Unlike bonded monomer chains, CG soft spheres may overlap, which expedites the equilibration of the melt that would have otherwise been entangled. Furthermore, previous work has shown that fundamental thermodynamic properties are fully captured by the CG representation when using our analytically-derived potential [97, 102, 103].

However, after accomplishing equilibration in the CG representation, we still require molecular information on the local scale to account for all properties of interest. By transforming the CG system to a UA representation, we can potentially deliver an equilibrated system having atomistic detail to material scientists at a fraction of the full-atomistic execution time. Furthermore, if we can alternate between the CG representation and the UA representation in an automated manner, then we can simultaneously benefit from the performance of CG and the accuracy of UA. Novel approaches for adaptive resolution in molecular dynamics, in which more interesting regions are coarse-grained at a finer resolution than less interesting regions [119], also require innovative methods for on-the-fly mapping back and forth between UA and CG. Section 3.2.3..1 describes our UA↔CG scientific workflow approach, which accomplishes this feat.

After a CG simulation has equilibrated to a minimal energy configuration, a crucial question is then: which UA system(s) are accurately represented by this arrangement? This raises the notion of the backmapping problem which is the subject of Sections 3.2.2..3 and 3.2.3..

3.2.2..3 Background: Backmapping

In homopolymer systems, transforming from the UA representation to the CG representation is straightforward: for each subchain having N_b monomers, the new soft sphere coordinate is simply the center of mass of the subchain. On the other hand, the *reverse* procedure of mapping from the CG representation to the UA representation is not generally well-defined. This transformation of a CG model into a UA model is a popular research topic, commonly referred to as the *backmapping* problem. For our homopolymer system, the backmapping problem is

simply stated as follows: given a collection of CG soft sphere chains coordinates, insert monomer chains in such a way that we would recover the original CG configuration if we were to coarse-grain the system again.

It is easy to see that solutions to backmapping problems are not unique, because there are many different UA configurations that could map back to a given CG configuration. Much backmapping work focuses on biomolecules [132], but relatively little work has explored homopolymers [133]. However, efficient backmapping procedures in polymer simulations are imperative for developing full-fledged adaptively resolved simulations.

In previous work [22], we used the Parallelizable Open source Efficient Multibody Software (POEMS) library to avoid backmapping. POEMS treats the internal subchains of each CG soft sphere as a set of *coupled rigid bodies*. This greatly reduces degrees of freedom in the simulation, and has the additional benefit of eliminating the need for solving the backmapping problem. Unfortunately, this approach suffers from poor computational performance, and our performance profiles unequivocally suggest it is due to time spent in POEMS' *Solve*, *initial_integrate*, and *final_integrate* functions. Also, an analysis of internal monomer distances versus endpoint distances (e.g., the monomers that connect adjacent CG spheres) shows that the endpoint bonds stretch to unphysical distances. This issue motivates the need for a backmapping procedure that leads to more physical bond distances throughout the system. Section 3.2.3. presents the design of our new backmapping procedure.

3.2.3. Workflow and Backmapping Design

This section discusses the design of our scientific workflow environment for obtaining high-performance equilibration of polymer melts with atomistic accuracy. We henceforth refer to the overall process as the *UA↔CG workflow*. Section 3.2.3..1 briefly describes the UA↔CG workflow implementation with the goal of providing context and motivation for the backmapping procedure. Further details regarding the workflow implementation can be found elsewhere [22]. Section 3.2.3..2 answers why, when, and how backmapping occurs within the workflow. We then describe our backmapping algorithm, and discuss future directions for possible improvements.

3.2.3..1 Design: The UA↔CG Workflow

The UA↔CG workflow consists of a series of computational programs and analyses that comprise an overall application for quickly stabilizing a randomly generated polymer melt of n chains, each with N monomers per chain. Subsequently, the workflow may be used to do production simulations of the equilibrated polymeric liquid. Figure 43 shows the 7 high-level stages involved in the equilibration workflow, each of which may involve multiple processing steps. Each step is accomplished by one or more programs, applications, or simulations. Before this work, these steps each required manual intervention by a researcher, but now they are automated by our workflow.

The workflow consists of a set of standardized Python wrappers of each Fortran, C, or C++ program that encapsulates Stages 1, 3, 5, and 7 from Figure 43. Some of these programs are actually our own custom versions of LAMMPS tools (such as `chain.f`, which generates random polymer chain systems)

that are optimized to run large-scale polymer systems. We use the standard Python argument parsing system, `argparse`, to define all simulation parameters at launch time. We subsequently pass the parameter bundle as a Python object throughout the entire workflow application.

In order for the workflow to be useful, Steps 3, 5, and 7 must have low overhead when compared to the full UA simulation itself. Furthermore, Step 4, which comprises the CG component of the equilibration, must exhibit better performance than UA and must converge towards a physically correct configuration. In our original implementation, the performance of Step 4 was unsatisfactory, especially when considering the vastly fewer degrees of freedom in the CG representation. The next section on backmapping discusses the source of this performance obstacle and our solution. In short, Step 4 executes most efficiently when completely discarding the internal UA coordinates tracked in steps 1-3. Unfortunately, this raises the additional concern of needing to recover those coordinates. This issue is considered in detail in Section 3.2.3..2.

It is worthwhile to note that workflow performance can be improved by eliminating step 2, which may be a relatively expensive UA simulation. To do this, we can transform the randomly generated configuration in step 1 directly to the CG representation. Despite starting from a non-equilibrated UA system, the CG equilibration is so fast (as we will see in Section 3.3.2..2) that we can save a large amount of execution time by skipping step 2. Step 1 builds the UA chains with the desired monomer density, step 2 is skipped, step 3 transforms the chains to CG, and step 4 rapidly equilibrates them.

Much of the data transfer between the workflow steps occurs by processing LAMMPS dump files to create new LAMMPS input files. In most cases, the time

spent reading files is negligible compared to the MD simulations, particularly when we configure LAMMPS to write to disk at relatively large timestep intervals. However, the larger the interval, the less information can be included in convergence analysis. In our studies, convergence is detected by examining the percent error change of the radial distribution function [22] within the UA representation. If the average percent error is below a user-defined threshold, then the workflow completes.

3.2.3..2 Design: Backmapping Software

One approach for backmapping is to store the subchain fragments in a database for later reinsertion into the soft spheres. However, this makes little sense for polymeric systems, because the necessary size of the database quickly becomes prohibitive. Firstly, we need a separate database of fragments for every value of N_b. Secondly, we need a large number of databases for different monomer types (e.g., different bond distances and masses). Finally, in order to obtain good statistics, many different suitable configurations are required for each possible N_b, and monomer type.

The approximate reconstruction approach for backmapping is far more suitable for homopolymer systems. If the configurational statistics of the reconstructed system is close enough to the equilibrium configuration, a perfect solution in not required because a quick UA simulation can remove any geometric strain and produce an equilibrated system (Step 6 of the workflow from Figure 43). In our first implementation, we take advantage of this technique by generating a very simple configuration of chains on a regular Cartesian grid. The steps for constructing a polymer melt on such a grid are as follows:

1. Store the center of mass coordinates for each soft sphere along the polymers.
2. Calculate the midpoints between the center of mass coordinates of each pair of adjacent spheres.
3. Initialize a regular grid across the simulation box with grid point distances equal to the desired bond length.
4. Redefine the above midpoints as the desired *endpoints* of each subchain and place onto the nearest grid point.
5. Generate paths connecting each pair of endpoints with the "Manhattan distance" length between endpoints.
6. Randomly extend each the path to the desired number of edges, N_b, by inserting *extensions*.

Figure 47 illustrates a simple example of the random extension in step 6. The leftmost graph is the result of steps 1-5 for a single subchain. By removing edge $\mathtt{e}_1$ and inserting the $\mathtt{ext}_1$ extender in its place, we extend the length of the chain's path by 2. Next, edge $\mathtt{e}_2$ is randomly selected and the path is extended by another 2 bond lengths. If we desire 10 monomers per sphere, then the algorithm is complete. Figure 48 shows a full example with roughly 30 monomers per soft sphere. For brevity, we omit certain implementation details in this algorithm description. The fully commented source code can be found in the UA↔CG Workflow repository, freely available online [218].

This backmapping approach has the advantage of being straightforward to implement, lightweight, relatively general, and potentially parallel. For polyethylene, however, this approach has the disadvantage of producing chains with unphysical bond angles along the carbon backbone, which means that a longer

simulation is required to equilibrate the newly generated UA system. An extension to the regular grid approach is to instead construct a *tetrahedral* grid, in which angles between grid points are forced to be 109.5° instead of 90°. We leave the tetrahedral implementation and comparing its equilibration requirements as future work.

3.2.4. Backmapping Experiments

Stages 1, 3, and 5 of the workflow only transform a *single* simulation snapshot. The algorithms are $O(m)$ (m is the total number of monomers), involve no communication, and typically comprise far less than 1% of the total workflow execution time. Stages 2, 4, and 6 potentially involve millions or billions of snapshots with much communication as particles advance. Stage 4 alone may consume over 90% of the workflow execution time (depending on how many time steps are assigned to each stage), making it the clear performance bottleneck. Therefore, Section 3.2.4..2 focuses on strong and weak scaling experiments of Stage 4 in the workflow, emphasizing the benefit of using our analytically-derived multiblock potential in the CG representation.

In addition, Section 3.3.2..2 discusses the overall performance of UA versus CG by considering the rate of convergence for each model.

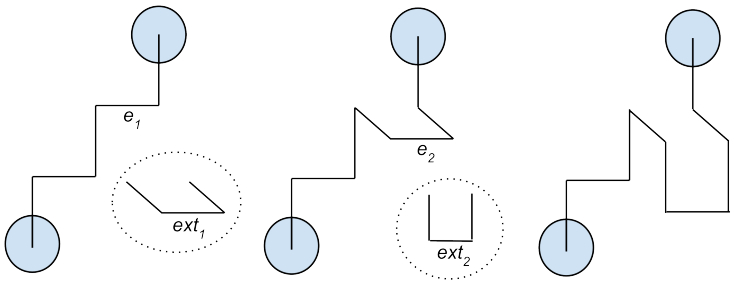


FIGURE 47. Simple example of regular grid backmapping.

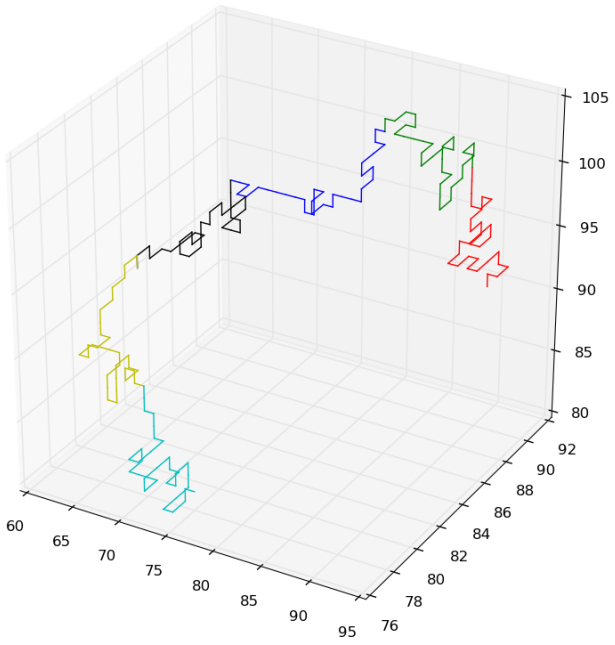


FIGURE 48. A randomly generated path with fixed endpoints on a regular grid. This is the UA configuration backmapped from a CG representation with ∼30 monomers per sphere. Each color corresponds to a different subchain, and units are in angstroms.

3.2.4..1 Experimental Setup and Evaluation

Experiments were conducted on the ACISS cluster located at the University of Oregon. We use the 128 generic compute nodes, each an HP ProLiant SL390 G7 with 12 processor cores per node (2x Intel X5650 2.67 GHz 6-core CPUs) and 72 GB of memory per node. ACISS employs a 10 gigabit HP Voltaire 8500 Ethernet switch that connects all compute nodes and storage fabric. The operating system is RedHat Enterprise Linux 6.2, and we used Intel version 14.0 compilers with OpenMPI 1.7. The latest version of LAMMPS (version 10-Feb-2015) was used with a slight modification to the chain generation tool to enable massive particle scaling.

We also include scaling experiments conducted on the Hopper system at NERSC. Hopper is a Cray XE6 cluster, where each compute node has 2 AMD 12-

core MagnyCours (24 cores total) running at 2.1 GHz. There are 32 GB of DDR3 RAM per node. We use the default LAMMPS 20140628 module, and the default PGI Cray compiler, version 14.2-0.

3.2.4..2 UA versus CG Performance per Timestep

Figures 49 and 50 show the strong and weak scaling of the UA versus CG components of the workflow on ACISS. These timings measure 500 femtoseconds of simulation time, then extrapolate to hours of execution time per nanosecond of simulation time, since we know that several nanoseconds are typically required to reach equilibration. The auxiliary workflow programs (CGgen, UAInit, and BMinit) never take more than a few seconds, which constitutes far less than 0.1% of the overall runtime for full simulations, so we do not include those times in these measurements.

It is important to note that all CG simulations in this section were performed with a 1 femtosecond timestep *independent of the granularity of the GC model.* However, we will see (in Figure 53) that the choice of the CG model affects the minimum timestep that needs to be considered in the MD simulation: the higher the level of coarse-graining the larger the timestep, which leads to a computational speed up even on a single processor.

Figure 49 shows a strong scaling experiment on ACISS, where each data point corresponds to a simulation that *represents* 200,000 UA monomer sites. In the UA case we run 20 chains, each with 10,000 monomers. The number of sites in the CG representation, however, depends on the granularity of the decomposition. For instance, with 10 CG sites per chain, each soft sphere contains 1,000 monomers, and only 200 total soft spheres are simulated. Therefore, it comes at no great

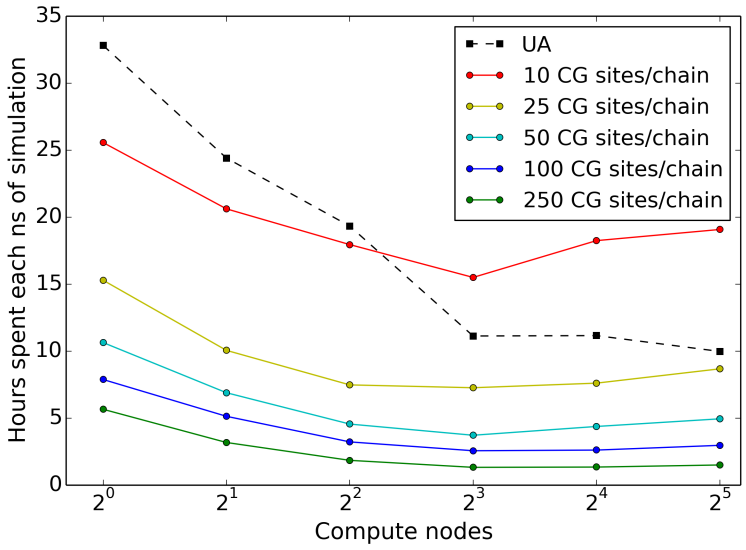


FIGURE 49. Strong scaling on ACISS for 20 chains each with 10,000 monomers. As expected, CG does not strong scale to as many processes as UA because it simulates fewer particles. However, absolute performance is usually far better for CG than UA, even at scale. Including more sites per chain is faster *per timestep*, but this ignores the rate of convergence of thermodynamic quantities and the use of a simulation timestep that gets larger the higher the level of coarse-graining (see text for further explanation).

surprise that the CG component of the workflow stops strong scaling after 8 nodes on ACISS (or 96 processes). At 8 nodes, only about 2 spheres reside in each process, and at 16 nodes, there are more processes than spheres, which is highly undesirable for an application like LAMMPS where communication between neighboring atoms plays a large role. At 250 sites per chain there are a total of 5,000 soft spheres, which is still far too small to exhibit good strong scaling at large numbers of processes.

It may be unexpected that the absolute performance is better for larger numbers of sites per chain, and is best at 250. There are two main reasons for this. The first is that we use an identical timestep in every CG simulation, whereas realistic MD should always adopt the maximum timestep allowed for a given level

of granularity. This issue will be explored in more detail in Section 3.3.2..2. The second effect is that having fewer sites per chain requires longer-ranged interactions (which we quantify in Section 3.3.2..1). The larger cutoff results in more neighbor particles per process, and the application becomes communication bound. Also, these measurements do not yet take into consideration the rate of convergence of structural and thermodynamic quantities, which we expect to be faster for coarser models.

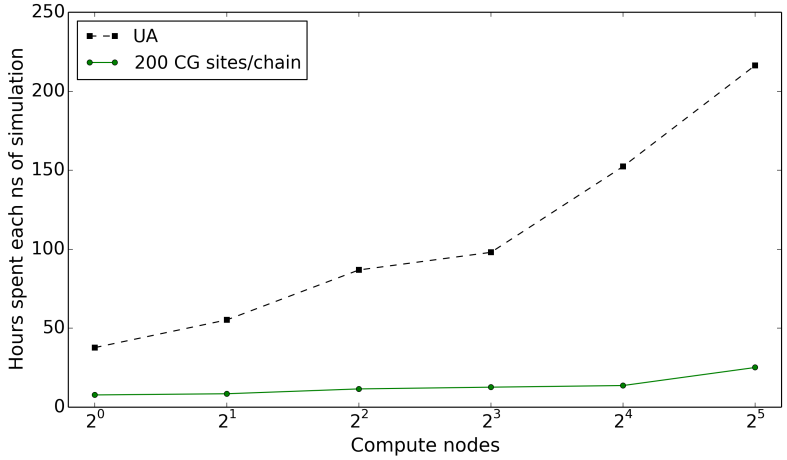


FIGURE 50. Weak scaling on ACISS for 20 chains per compute node, each chain containing 10,000 monomers.

Figure 50 shows the corresponding weak scaling experiment on ACISS, in which the number of chains per compute node is held constant at 20 chains for each execution. For instance, at 32 nodes, UA simulates 640 chains for a total of 6.4 million particles. CG also simulates 640 chains at 32 nodes, but at 200 sites per chain 32,000 spheres are tracked. The plot clearly shows that tracking 32,000 soft spheres exhibits far better weak scaling than simulating 6.4 million atoms. This is certainly expected, but it is also important to note that using our potential in the CG simulation also accurately produces an equilibrated system with the same

thermodynamic properties as the UA simulation. Previous work has demonstrated excellent recovery of pressure, free energy, and structural correlations at several different granularities of up to 1,000 soft spheres per chain [103]. Until this work, modeling 10,000 to 1 million monomer sites per chain was impractical.

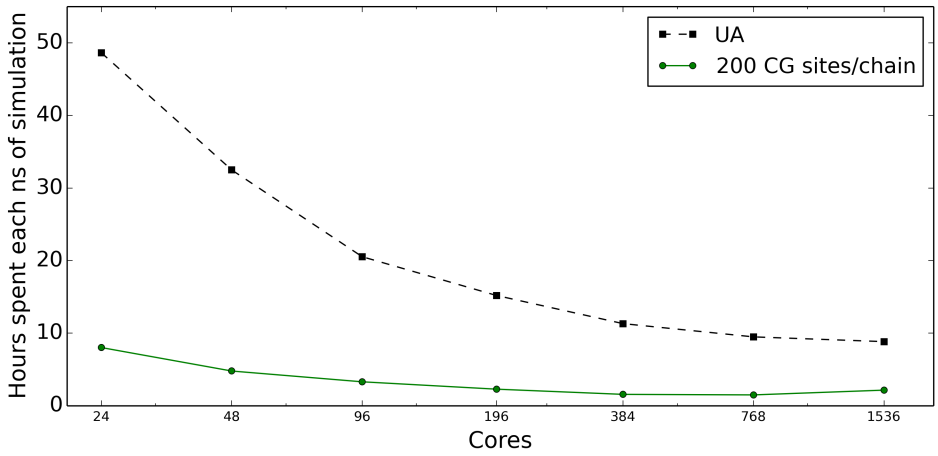


FIGURE 51. Strong scaling on Hopper for 20 chains each with 10,000 monomers.

To provide evidence that the above results are independent of the cluster architecture, we ran a comparable experiment on the Hopper system at NERSC. Figure 51 shows a strong scaling experiment with 20 chains and 10,000 monomers per chain for UA versus CG with 200 sites per chain. As before, the CG simulation stops strong scaling before UA, likely due to the much smaller number of particles tracked (1,000 versus 200,000). On the other hand, Figure 52 confirms that the CG simulation weak scales far better than UA, up to 3072 processes.

We also ran equivalent weak scaling experiments with 2 chains per node and 100,000 monomers per chain with very similar performance results. We omit those results here for brevity, but the fact that this is possible is exciting for future validation work and our ability to simulate systems that reflect real-world polyethylene melts.

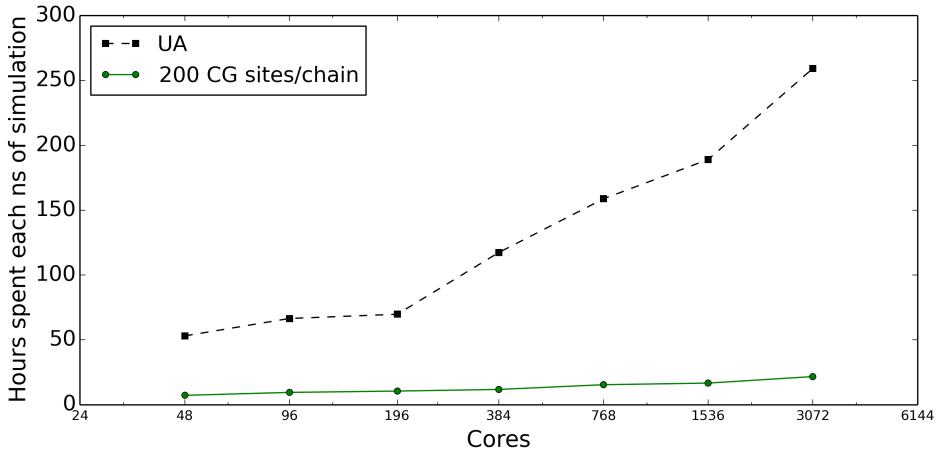


FIGURE 52. Weak scaling on Hopper. Each UA simulation runs 20 chains per node with 10,000 monomers per chain. The CG simulations have a granularity of 200 soft sphere sites per chain, so each sphere contains 50 monomers. Our CG simulations weak scale nicely, and offer a speedup of about 12x at 3,072 cores.

While it is possible to achieve speedups on the order of 10x for CG simulations based on IECG theory, it is important to converge to a useful configuration in UA space. Measurements of our backmapping procedure are on the order of a few seconds, even for systems containing millions of particles. Compared to the several hours required to complete a full equilibration, this time is negligible, and gives more significance to our CG versus UA comparisons.

3.3 Modeling for the Simulation Timestep

3.3.1. Introduction to MD/CG Simulation Timesteps

The coarse-grained potential developed by Guenza and co-workers describes a collection, each of which we call a *block*, of monomers. As described in Section 3.2.2..1, each block models several individual UA monomer units. Therefore, the CG polymer chain forms a *multi-block* model, in which different intramolecular and intermolecular terms describe the relevant interactions.

A primary goal of this chapter is to consider how simulation performance varies with the number of blocks used to model polymers. This is particularly important because previous work shows that the CG potential correctly models the free energy, structure, and thermodynamics of polymer melts [103], so we want to choose the best *granularity* to optimize computational performance. Since reducing the blocks per polymer reduces the degrees of freedom, we might expect this to improve simulation performance. However, we also need to consider how the long-range effects of the coarser potential increase the size of the Verlet list data structure.

In addition, the integration timestep, which is the amount of time between evaluations of the potential, directly affects computational efficiency. If the timestep is chosen to be too short, then wasted calculations increase the number of issued CPU instructions, but do not lead to improvements in accuracy. If the timestep is too long, then aliasing errors may lead to incorrect trajectories that degrade the structural and thermodynamic correctness of the simulation. Typically, UA simulations choose a timestep of about 1 femtosecond to capture the relevant atomistic dynamics. In general, the timestep is selected to be two

orders of magnitude smaller than the fastest dynamics in the simulation, which for molecular liquids are the bond fluctuations. CG simulations, however, allow for a larger timestep because the frequency of intramolecular vibrations is considerably slower.

To see this, consider that the potential between bonded sites in the multi-block model is derived from the direct Boltzmann inversion of the probability distribution of the effective bond length [103]:

$$v_{bond}(r) = -k_B T \ln[P(r)/r^2] \tag{3.3}$$

where

$$P(r) = 4\pi \left[\frac{3}{\pi 8 R_{gb}^2}\right]^{3/2} r^2 \exp\left[-\frac{3}{8R_{gb}^2}\right]. \tag{3.4}$$

and $R_{gb} = R_g/\sqrt{n_b}$ with R_g being the polymer radius of gyration and n_b the number of blocks per polymer chain. Substituting Eqn. 3.4 into Eqn. 3.3, we have

$$v_{bond}(r) = \frac{3k_B T}{8R_{gb}^2} r^2 + const. \tag{3.5}$$

Comparing this to the well-known potential for a harmonic oscillator, $U(r) = kr^2/2$, we see that the bond's spring constant is

$$k_{bond} = \frac{3k_B T n_b}{4R_g^2} \tag{3.6}$$

and the period of this harmonic oscillator is $\tau_{bond} = 2\pi\sqrt{m/k_{bond}}$, or

$$\tau_{bond} = 2\pi\sqrt{\frac{4mR_g^2}{3k_B T n_b}} \propto \frac{1}{\sqrt{n_b}} \tag{3.7}$$

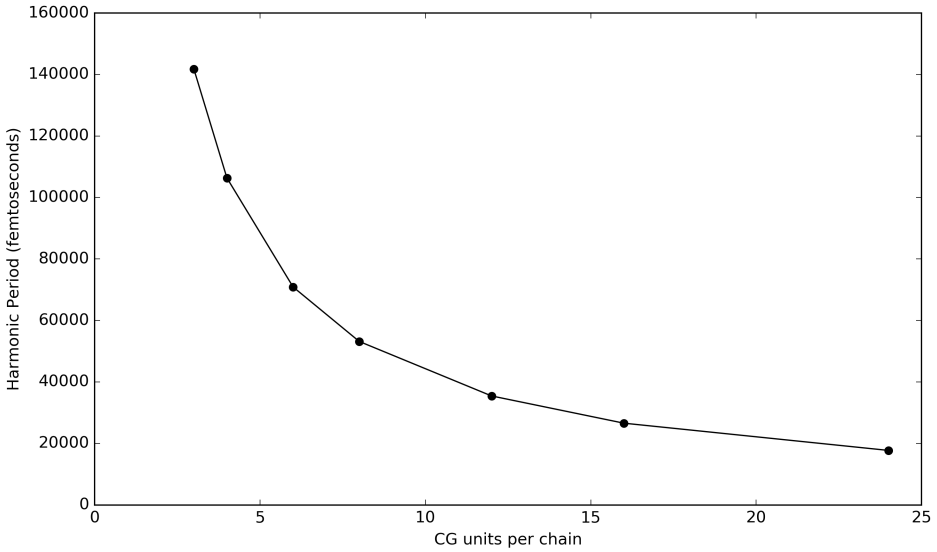


FIGURE 53. slope=1/2 analysis From Equation 3.7, this plot shows the intra-molecular harmonic period, τ_{bond}, across several CG granularities, where $n_b = \{3, 4, 6, 8, 12, 16, 24\}$. Each polyethylene chain contains 192 monomers, temperature is 509 K, and $R_g^2 = 541.0334\text{Å}^2$.

This rather interesting result suggests that coarsening (that is, decreasing the blocks per chain) the multi-block model allows us to increase the CG simulation timestep proportionally to $1/\sqrt{n_b}$.

As a more concrete example, Figure 53 shows the values of T_{bond} for a polyethylene system in which T=509K and $R_g^2 = 541.0334\text{Å}$. For the tri-block model ($n_b = 3$), intra-molecular vibrations have a period of ∼10,000 femtoseconds. To be able to catch the bond vibrational motion with accuracy in the MD simulation, the timestep has to be considerably shorter. Standard practice is for sample frequency to be at least a couple of orders of magnitude faster than the period, which is about 1 picosecond. This is a considerable improvement over the typical UA timestep of 1 femtosecond. For a six-block model the timestep is around 600 femtoseconds, and it further reduces as the granularity coarsens.

3.3.2. Timestep Experiments

Next, we performed an empirical study to verify that the expected period values shown in Figure 53 accurately estimate an appropriate timestep parameter for CG simulations. This experiment involves running LAMMPS at several different timesteps while keeping the granularity n_b fixed. The intent is to determine how large the timestep can be set in order to converge to the correct pressure. We set n_b=6 in this experiment because our analysis in Section 3.3.2..1 showed that the 192 monomer system shows the best performance with n_b=6, which is the highest value that still maintains the requirement $N_b \geq 30$. Then, we run several CG simulations while varying the timestep to find the maximum value that converges to the correct pressure of 356 $\pm$ 1 atm (as determined by a gold-standard UA simulation). The results are shown in Table 4, where we see that larger timsteps require fewer simulation steps to reach pressure convergence. Specifically, we can increase the timestep to 600 femtoseconds until LAMMPS fails with a numerical error resulting from a CG unit that is launched too rapidly out of the simulation box. Typically, the timestep in a simulation is selected to be two orders of magnitude lower that the fastest dynamics simulated. The expected period values shown in Figure 53 suggest a harmonic period of ∼60,000 femtoseconds, which corresponds to a timestep of ∼600 femtoseconds. This prediction is in very good agreement with the value found empirically.

3.3.2..1 Analysis of CG Computational Performance

The results shown in Figure 49 may initially seem counter-intuitive. The plot suggests that models with finer granularity perform better than models with coarser granularity, despite the higher number of particles and more degrees of

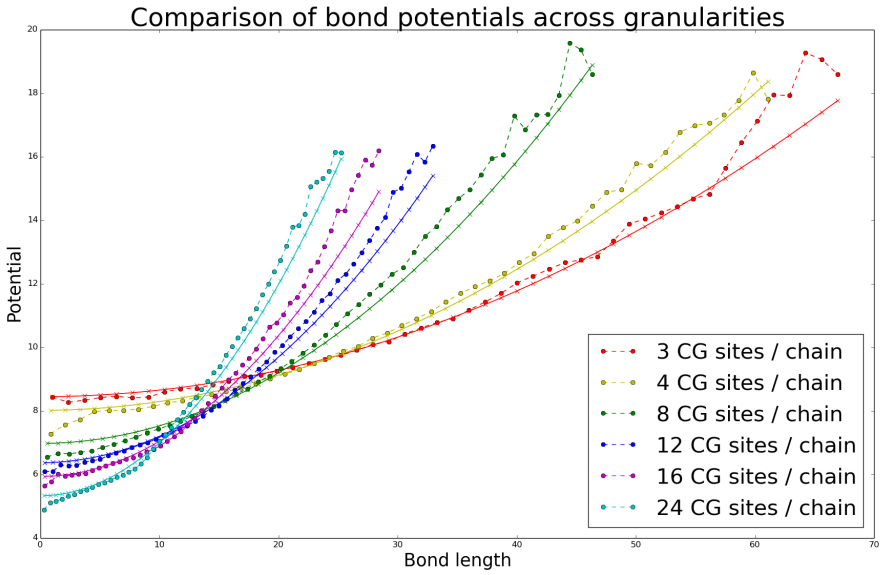


FIGURE 54. Theory versus simulation for the harmonic intramolecular potential (v_{bond} from Equation 3.3) across several n_b granularities. The model parameters are the same as in Tables 4 and 5.

freedom. In fact, several effects compete against each other to determine the overall computational performance of CG simulations. We mentioned in the previous section that the speedup of simulations for coarser models is due in part to having the opportunity to choose a larger minimum timestep. A second effect, which instead slows the performance, is that coarser models have a longer ranged potential, which leads to a larger cutoff distance. The theoretical scaling of the timestep with CG granularity was introduced in Section 3.3.1., but this section describes the effect of changing this cutoff in more detail, and presents an explanatory performance model. The performance of the simulation with increasing timestep is further discussed in Section 3.3.2..2.

The speed of performing an MD simulation is proportional to the number of nonbonded pairs that must be evaluated, N_{ints}. This can be approximated as the product of the total number of particles and the average number of particles each

timestep (femtoseconds)	Pressure (atm)	steps to convergence
1	356.3 ± 1.0	95,000
10	356.2 ± 0.7	6,100
100	355.5 ± 0.5	500
200	355.5 ± 0.5	300
300	355.6 ± 0.7	200
400	355.9 ± 1.9	90
500	356.2 ± 2.1	50
600	numerical error	N/A

TABLE 4. Empirical timestep experiment in LAMMPS. The parameters are 350 chains, N=192, n_b=6, T=509K, c_0=−10.019, R_g=541.034Å^2. The "steps to convergence" column is calculated by determining the first 10 timesteps whose average pressure is within 1 atm of 356. The overall pressure is then calculated for the following 10,000 steps. For timesteps greater than 600, the dynamics become too fast and inter-molecular atoms tend to launch apart after a few steps, causing a LAMMPS error.

particle interacts with:

$$N_{ints} = (\rho_b V_{system}) \times (\rho_b V_{search}) \tag{3.8}$$

where V_{system} and V_{search} are the simulation volume and search volume, respectively, and the CG site number density, ρ_b, is proportional to the number of blocks per polymer chain, n_b:

$$\rho_b = \frac{\rho_m}{N} n_b. \tag{3.9}$$

As before, ρ_m is the monomer density and N is the number of monomers per polymer chain, which are constant for a given system.

The system volume is lower-bounded by the cutoff of the coarse-grained potential (r_{cut}). At a minimum, the box length in each dimension must be twice

the potential cutoff, giving this total volume as

$$V_{system} = (2r_{cut})^3. \tag{3.10}$$

Finally, the search volume around each particle is a sphere with radius r_{cut}:

$$V_{search} = \frac{4}{3}\pi r_{cut}^3. \tag{3.11}$$

Combining Equations 3.8, 3.9, 3.10, and 3.11 gives

$$N_{ints} = (n_b)^2 (r_{cut})^6 \left(\frac{\rho_m}{N}\right)^2 \left(\frac{32\pi}{3}\right). \tag{3.12}$$

This equation provides a useful model for predicting how granularity affects the number of interactions tracked by a simulation, which is directly related to computational performance.

Our approach for determining r_{cut} is to choose a root (or zero) of the derivative of the potential energy function. This eliminates the possibility of particles experiencing abrupt and spurious forces near the edge of the cutoff region. Choosing the first root only includes the first repulsive part of the potential, which leads to incorrect pressures (as seen in Table 5). Choosing the second or third root includes the repulsive section(s) of the potential and the attractive contribution between them, which gives a correct pressure value. Because the third root results in a larger value of r_{cut} and therefore a more expensive simulation, we always choose the second root.

Let us now consider how the granularity of the CG model effects the value of N_{ints} and, consequently, the performance of the simulation. In general, r_{cut}

n_b	root	r_{cut} (Å)	N_{ints} (theory/sim.)	time (s)	P (atm)
3	1^{st}	67	300.3 / 403.6	24.2	395.9
3	2^{nd}	127	2,015 / 2,612	99.2	357.4
4	1^{st}	54.2	217.4 / 243.0	17.5	395.3
4	2^{nd}	103	1,471 / 1,567	71.0	356.6
6	1^{st}	40.7	137.6 / 159.1	19.2	391.5
6	2^{nd}	77.3	945.9 / 1,025	53.6	356.3
UA	N/A	50	(N/A) / 82.3	374.1	356 ± 1.0

TABLE 5. Tabulated data showing how granularity affects the performance and pressure correctness of CG simulations. The model includes 350 chains, 192 monomers per chain, R_g^2=541.034Å^2, c_0=−10.019. The N_{ints} theoretical value is calculated with V_{system} set to the LAMMPS box-size. The simulation value is the average neighbors / atom as reported by LAMMPS. The time is overall LAMMPS execution time for 15,000 steps, and the pressure is the average value after 100,000 steps.

decreases with n_b when the granularity is coarse enough such that the requirement $N_b \geq 30$ is satisfied. On the other hand, n_b obviously increases with itself. Therefore, Equation 3.12 suggests that chain granularity increases the value of N_{ints} as n_b^2, but decreases N_{ints} as r_{cut}^6. Because the value of r_{cut} itself depends on n_b, we tabulate several values of N_{ints} to quantify the overall effect of changing the granularity of the CG model. These values are shown in Table 5. Values that correspond to a LAMMPS simulation (the "N_{ints}(sim.)", "time", and "pressure" columns) were run with 96 MPI processes (12 processes per compute node) on the ACISS cluster.

Table 5 contains values for a model with 350 chains and 192 monomers per chain. We set n_b to 3, 4, and 6 because these are the only values for which $N_b \geq 30$ and the CG bond angle potential, $v_{angle}(\theta)$, is valid (see [103] for details

The claim that r_{cut} decreases with n_b is easily verified with a plot, but it is not included here for the sake of brevity. Table 5 shows this behavior for $n_b = 3, 4, 6$. On the other hand, r_{cut} eventually increases with n_b, but only in the region where the theory is no longer valid, i.e., when $N_b < 30$.

on $v_{angle}(\theta)$, which is analogous to $v_{bond}(r)$ from Equation 3.3.). Table 5 confirms that the r_{cut}^6 term dominates overall, because N_{ints} decreases as we coarsen the granularity of the CG model. The fourth column of Table 5 shows the theoretical lower bound for N_{ints} from Equation 3.12 (with V_{system} set to the size of the simulation box in LAMMPS) along with a measurement from LAMMPS that includes the average number of neighbor interactions (which also includes intra-molecular interactions). We see that the theoretical lower-bound predictions closely match (and never exceed) the actual simulation measurements.

Table 5 also displays a column that includes the total execution time of 15,000 simulation steps in a LAMMPS simulation. The data suggests that overall simulation performance is directly proportional to N_{ints}. This claim is further verified by observing that most of each simulation's execution time is spent doing pairwise calculations and communication (typically, over 90%). This behavior suggests that more fine-grained models exhibit better computational performance properties. While this is true for a constant simulation timestep, the next section will describe how the timestep can be dramatically increased as we coarsen the granularity of the polymer system.

The final column of Table 5 shows the pressure of the CG systems after 100,000 simulation steps with a timestep of 1 femtosecond. The corresponding UA simulation produces a final pressure of 356±1.0 atmospheres, which we establish as the correct pressure. We see that the 1^{st} root overestimates the pressure by over 10%, which affirms that the 2^{nd} root is the better choice for accuracy. The 3^{rd} roots (not shown) also reproduce the correct pressure, but are considerably more expensive due to the increase in r_{cut} and, consequently, in N_{ints}. The final take-

away is that the granularity of the CG model has no effect on the correctness of the pressure, despite the different computational costs.

3.3.2..2 Overall Convergence Performance of UA versus CG

The previous section presented and verified a performance model that confirms the computational efficiency of coarser models is lower than finer grained models (given a constant timestep). This effect is due to the increase in r_{cut} and neighbor interactions, N_{ints}. Here we study how the *timestep* may increase for coarser granularities. It is well known that CG dynamics are faster than the corresponding atomistic simulation or experiment [219, 220]. Using our analysis from Section 3.3.1., we can explore how increasing the timestep with granularity accordingly results in a faster convergence towards thermodynamic equilibrium.

We begin by verifying that the theoretical $v_{bond}(r)$ and $P(r)$ functions described by Equation 3.3 and 3.4, respectively, correspond to the actual CG simulation data. Once verified, we have substantial confidence that the shape of the harmonic potentials (specifically, the multiplicative spring constant) provide a model for selecting a reasonable timestep according to Equation 3.7. Figure 55 shows how the theoretical expression from Equation 3.4 compares to actual CG simulation data across several different granularities. We see good agreement, which is expected because the intra-molecular bond vibrations are dictated by Equation 3.3, which depends only on temperature, the polymer radius of gyration, and the granularity. From this data, we can also compare Equation 3.5 directly to the simulation data, which also shows good agreement in Figure 54.

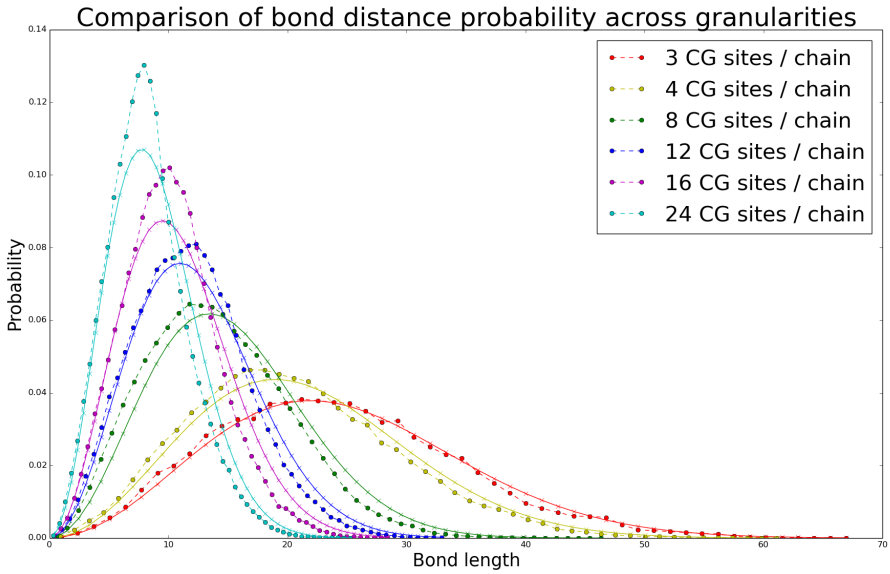


FIGURE 55. Theory versus simulation for the bond probability histogram ($P(r)$ from Equation 3.4) across several n_b granularities. The model parameters are the same as in Tables 4 and 5.

3.3.3. Related Work to CG Workflows

Most related CG models (such as within the MARTINI force field [107]) use numerically optimized potentials, with the problem that numerical errors in the optimization of the potential can propagate to other physical quantities, such as thermodynamic properties. In general, numerically optimized CG potentials have problems with transferability and correct representability. It has been observed that numerical CG potentials that are optimized to reproduce some physical quantity correctly, such as the structure, may not correctly reproduce other quantities, such as the pressure or the free energy of the system [221, 102, 222, 223]. This problem is not present in our analytically-based CG simulations, because the formal solution of the potential, as well as of the structural and thermodynamic quantities of interest, ensures their consistency across variable levels of resolution in the CG description.

Many others have studied the backmapping problem, and due to the non-unique nature of the solutions, many different methods have been proposed such as fragment-based libraries [224] and optimization procedures with gradually relaxing harmonic constraints between the UA and CG systems [225]. Furthermore, particular backmapping procedures tend to be specific to the problem under consideration, often by necessity. For example, backmapping polymer models have been applied to nonequilibrium shear flows by Chen et al. [226]. For CG models of polymers with rigid side groups, Ghanbari et al. use simple geometric properties of the molecular fragments to reconstruct the atomic system [227]. Our implementation in Section 3.2.3..2 supports homopolymers with a relatively constant bond distance (although, it would certainly be possible to generalize to heteropolymers or block copolymers in future implementations). The general sentiment in the CG research community is that backmapping procedures are important for supporting adaptive resolution capabilities, but we desire a general approach for backmapping. We believe that online construction of realistic configurations offers the most promising approach for a general and high performance backmapping procedure.

A plethora of related research and development focuses on optimizing the performance and productivity of scientific workflows. One of the most popular frameworks is Kepler, which has recently presented full-fledged workflows for drug design, and developments towards supporting popular distributed data-parallel patterns, such as MapReduce [212]. Other scientific workflow environments include Pegasus [213], and Taverna [214]. Instead of committing to a full-blown workflow framework up-front, we tap into the advantages of using a considerably lightweight system for managing input parameters while benefiting from simple

data provenance and template processing. For instance, our workflow is trivial to setup on any Linux based machine with Python and LAMMPS installed.

3.3.4. UA↔CG Workflow Conclusion

Coarse-graining methods benefit from reduced computational requirements, allowing us to simulate systems with a realistic number of atoms relative to laboratory bulk experiments. Few polymer simulation studies have explored long-chain configurations, despite their importance for studying real-world systems. This chapter has presented a customized set of software tools for running such large systems with LAMMPS as the primary molecular dynamics framework. We use the analytically based IECG potential, which has been shown to preserve both thermodynamic quantities and bulk properties, unlike other numerically based potentials. In addition to our collection of individual software tools, we have implemented a full-fledged scientific workflow that enables automatic transformation between UA and CG representations. The CG to UA backmapping problem is handled by randomly generating a polymer onto a regular Cartesian grid followed by a short UA equilibration.

The UA↔CG workflow enables four noteworthy features for conducting large-scale polymer studies:

1. Automated setup of CG systems based on a corresponding UA configuration
2. Excellent parallel performance when applying IECG theory
3. A backmapping procedure from the CG to the UA representation, restoring local molecular information
4. Potential to iterate through several cycles of the workflow loop

By having low-overhead tools between workflow phases, we can focus on the performance comparison of CG versus UA. Not only do our scaling experiment results show a general benefit of using our CG potential over straightforward UA, they also suggest the effectiveness of our new workflow for transforming between UA and CG. We have quantified what performance improvements to expect when in the CG component of the workflow, and we have presented a performance model which accurately estimates the number of particle interactions for a given granularity. Others can use this model to better understand how to choose the granularity factor to best exploit computational performance benefit and thermodynamic accuracy.

Now that simulation of long-chain polymer systems is possible, efficient, and dynamically transformable into either UA or CG representation, future work will formally validate the thermodynamic quantities similarly to what has been done for $\leq$1000 monomers per chain. By quickly switching back and forth from UA to CG, we open doors to new studies of polymeric systems while maintaining simulation accuracy and efficiency.

CHAPTER IV

QM/MM: HYBRID QUANTUM/MOLECULAR MECHANICS

This chapter explores the state of the art in the capabilities of existing QM/MM software applications, then presents and analyzes new features for improving their programmability and performance. Section 1.5 described the QM/MM concept and provided detailed background information. Recall that the goal of the QM/MM method is to simultaneously exploit the *accuracy* of QM and the *efficiency* of MM by carefully combining the two models into a multiscale simulation. Many other sources [114, 112, 228] describe the theoretical foundation for QM/MM, but this chapter instead studies software implementations of these methods. Several different software artifacts exist for accomplishing QM/MM calculations, and they all differ in very important ways. In this chapter, I categorize the different implementations while analyzing their software engineering patterns and how they improve (or impair) development productivity and parallel performance. In particular, I focus on issues that arise when *combining* different modules or libraries into complete QM/MM applications.

4.1 Existing QM/MM Implementations

This section surveys and classifies existing QM/MM software implementations according to their software design, architecture, and coupling features. Many of the most popular packages have software licenses with proprietary components, such as AMBER [76], CHARMM [77], and GROMOS [229]; however, this section focuses primarily on software with completely free and open source licenses, such as LAMMPS [230], GROMACS [85], CP2K [231], and NWChem [232]. It is

more practical to focus this study on open source projects because their software architectures and designs are more apparent, and there exist more leading-edge research publications related to these packages. Despite the availability of the source code, however, the software architecture and design is not always clearly documented, which means that we may require some reverse engineering to glean information about the software structure.

As illustrated in Sections 1.2 and 1.3, the calculations for the QM and MM modules are vastly different. For instance, QM applications are usually bottlenecked by the sheer number of electron integrals and/or the tensor contraction calculation costs, whereas MM applications are relatively more bottlenecked by communication latencies. It is commonplace for QM/MM code bases to be distinctly separated into independent modules, files, libraries, and/or binaries, according to their role as either QM, MM, or hybrid QM/MM kernels (but sometimes the components are more diverse and categorizing them within one of the three models is more dubious). Table 6 classifies several open source packages according to the relatively broad distinction between single/multiple binary architectures and internal/external library coupling. Sections 4.1.1. and 4.1.2. describe the differences between these architectures and justify the classification of each QM/MM package.

Before diving into the detailed descriptions of the software architecture and design of QM/MM implementations, it is pertinent to define and clarify these commonly overloaded and misinterpreted terms. In the subsections below, the phrase *software architecture* refers to a high-level description and/or abstraction of how software components interact and what programming concepts facilitate the big-picture software requirements. On the other hand, the phrase *software design*

Package	Architecture	Couples with
CP2K	Single-binary, internal libraries	N/A
NWChem	Single-binary, internal libraries	ChemShell, LICHEM
GROMACS	Single-binary, external libraries	GAMESS-UK, Gaussian, MOPAC, and ORCA
ChemShell	Single or Multiple-binary, external libraries	GAMESS-UK, NWChem, Gaussian, ORCA, and more
LICHEM	Multiple-binary, no linking	NWChem, Gaussian, TINKER, and PSI4
LAMMPS (QMMMW)	Multiple-binary, external libraries	Quantum ESPRESSO
LAMMPS (USER-QMMM)	Single-binary, external libraries	Quantum ESPRESSO

TABLE 6. A comparison of the software architecture and supported coupling of several QM/MM packages.

refers to a relatively lower-level description of exactly how API's are defined and what happens, both syntactically and semantically, when invoking these interfaces that connect individual components.

4.1.1. Single versus Multiple Binary Architectures

As far as this work is concerned, the most important division between QM/MM implementations is whether the compiled application runs as a single program binary or across multiple program binaries. It is trivial to determine whether a QM/MM package is a single or multiple-binary architecture: when executing the application (usually with a startup mechanism such as `mpirun`) how many different binary files must execute to complete the simulation? In the case of CP2K, NWChem, GROMACS, and LAMMPS (USER-QMMM), it is clear that only one binary (`cp2k`, `nwchem`, `gmx`, and `lmp`, respectively) is passed to `mpirun`. For others, such as LICHEM or LAMMPS-QMMMW, multiple binaries need to

be executed, either by invoking `mpirun` multiple times on the command line (as in LAMMPS-QMMW), or by a driver program that makes multiple system calls against existing binaries (as in LICHEM). Some packages may be capable of either single or multiple-binary execution, such as ChemShell, which has both a library-based "direct linking" mode and an "external binary" mode with various support for different calculations and methods across several software frameworks.

The single and multiple-binary architectures each have advantages and disadvantages. One clear advantage of a single binary is the convenience of executing and controlling its launch via job scheduling systems on various computer clusters. For multiple binaries, the procedure for launching multiple parallel jobs simultaneously with a workload manager can vary greatly, and sometimes support is partial. This can also be problematic on clusters with job schedulers that do not allow binaries to launch other binaries from backend compute nodes, which is common on today's largest supercomputer systems (like Titan at Oak Ridge and Edison at NERSC). On the other hand, if multiple binary execution is well-supported, runtime control and tuning is arguably more flexible, simply because separate parameters (e.g. the number of threads versus processes) can be more easily passed to the separated binaries based on what performs best for each parallel program.

While a single binary is generally preferable with regard to parallel execution, it may require too much development effort to couple code bases having different software stacks. For instance, the MPMD mechanism of MPI is not used in any of the QM/MM implementations in Figure 6, possibly because of the lack of a robust standard for how such applications execute across different MPI implementations and job schedulers. It is also relatively less desirable to couple applications with

different software stacks or programming languages into a single binary (or with MPMD MPI) compared to other mechanisms that support a higher level of data abstraction, such as through ADIOS, which I describe further in the next section.

Concerning data abstractions, we must consider that *either* the single *or* the multiple-binary approach may rely on the generation of framework-specific input files for the different software components. Few frameworks have support for automatically generating these input files (LICHEM and ChemShell), but others rely on the user to create *separate* input files for each component, usually with considerably different scripting paradigms and definitions. This can be quite burdensome on users, especially when using multiple binaries for frameworks which use completely different scripting definitions, which is essentially the case for *all* computational chemistry frameworks (though projects such as OpenBabel alleviate this issue, at least for QM codes). Data transfer itself is relatively more difficult with multiple binaries and requires passing files (slowly) or managing shared memory between applications with separate memory address spaces. LAMMPS-QMMMW supports a select few mechanisms such as POSIX shared memory (`shm_open`/`unlink` and `mmap`), files, and Python sockets. However, as of this writing these mechanisms only support sequential MM execution. Section 4.3 discusses why this is a serious detriment to scalability, and Section 4.2 presents a data abstraction that mitigates this issue.

The multiple-binary architecture are usually more maintainable and interoperable than a single binary, especially when dealing with disparate software stacks. Furthermore, as long as components are modularized with a preference towards functional design, independent software stacks are not as much of a problem compared to single binaries which apply tighter coupling. By definition,

a multiple-binary architecture requires two different linking steps, which actually coincides with the paradigm that QM and MM applications are best managed as separate software repositories with separate compilation procedures. This alludes to how coupling libraries are linked and organized (internally versus externally), which is the subject of the next section.

4.1.2. Internal vs External Coupling Designs

Another characteristic of QM/MM software architecture (and many other coupled physics applications for that matter) is whether their coupling is achieved via *internal* or *external* library interfaces. This distinction is somewhat related to the single/multiple binary categorization, but instead depends on how the API design of computational chemistry libraries are utilized to create overall applications. *Internally* linked libraries rely on a design in which components fall within a single software framework, and usually share consistent naming schemes, modular design patterns, and programming languages. *Externally* linked libraries simply combine the libraries of more than one framework to accomplish QM/MM calculations. An example of a framework utilizing an internal QM/MM design is NWChem, which implements separate Fortran modules to couple QM and MM regions using libraries that are built as a part of the overall software package. Specifically, these are the `libmm`, `libqmmm`, and other QM libraries (such as `libtce` and `libnwdft`). While NWChem implements QM/MM internally, its design sufficiently allows other frameworks to link with its libraries *externally* (for example, ChemShell supports external linking with NWChem). An example of a framework that uses an external design is LAMMPS, which can couple with Quantum ESPRESSO regardless of whether using the single or multiple binary

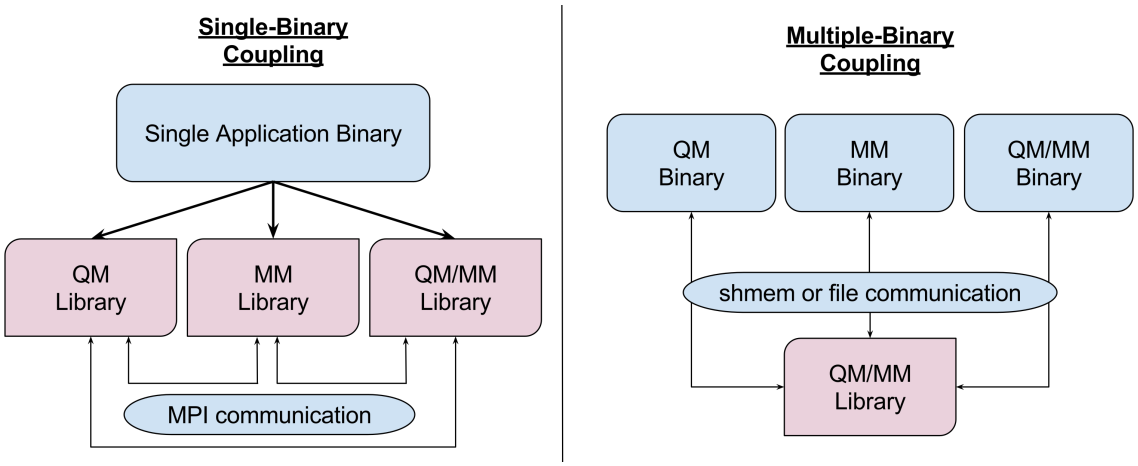


FIGURE 56. The software architecture of production-ready QM/MM applications fall into 3 categories. The first is a completely internal module-based application (not shown) in which a single binary implements the entire QM/MM application, usually through an object-oriented design. The second (left) is where a single application binary makes library calls to the QM, MM, and QM/MM components. The third (right) involves separate component binaries using a common coupling-library for data transfers and synchronization.

architecture of USER-QMMM or QMMMW. Other examples include GROMACS, CHARMM, and ChemShell.

The advantages and disadvantages somewhat mirror the discussion from the previous section. Internal libraries are usually easier to maintain because they often share a common software stack and API design patterns. Furthermore, in the experience of this author, it can be far more difficult to link together disparate libraries from external chemistry frameworks simply because of issues like compatibility of static/dynamic linking, link ordering on some Linux systems, and sometimes the need to pass linking flags through a compilation procedure that otherwise would not need them. This last hurdle can be especially apparent in QM/MM implementations that use external linking to compile a single binary. On the other hand, internal libraries are by definition less flexible in terms of supporting high interoperability between software frameworks. If there is no library

API for a chemistry framework that is designed for external interaction, there is little to no chance of it implementing QM/MM with other frameworks.

Figure 56 contains a graphic that represents two software architectures of interest in the QM/MM context. The left side shows an architecture with a single application binary that makes several different library calls to perform the required calculations (this diagram does not specify whether these libraries are linked internally or externally). On the other hand, the architecture on the right side of Figure 56 shows a multiple-binary arrangement, in which a single *library* somehow coordinates data transfer and execution coordination across multiple required binaries. This juxtaposition is not unique to QM/MM applications; in fact, the coupling of many multi-physics software frameworks can also be done with single or multiple program binaries [233].

4.2 A Novel Design for QM/MM/CG Coupling

The DataPrep library provides a common infrastructure for sending relevant atomic data between QM, MM, and CG application components, regardless of whether the hybrid simulation is compiled as a single or a multiple-binary application. An application initializes DataPrep before and after MPI:

```
/* QM_binary.c */
MPI_Init(&argc, &argv);
dataprep_init(&argc, &argv);
dataprep_init_QM("MM_binary");
/* Application work ... */
dataprep_finalize();
MPI_Finalize();
```

```
/* MM_binary.c */
MPI_Init(&argc, &argv);
dataprep_init(&argc, &argv);
dataprep_init_MM("QM_binary");
/* Application work ... */
dataprep_finalize();
MPI_Finalize();
```

Now consider the following C++ object which captures the relevant atomic data for QM/MM simulations (passing an equivalent C-style struct is also supported):

```
class QMM_Data {
  int natoms;
  double box_dimensions[6];
  double *atoms_positions;
  ... };
```

```
/* QMM_Data class methods */
void SetNatoms(int);
void SetBoxDims(double[6]);
void SetPositions(double *);
...
```

After packing this struct, passing data between components is simple. For example, the MM application may put coordinates:

```
QMM_Data MM_data;
/* Initialize and equilibrate atomic positions here ... */
dataprep_put(&MM_data);
```

and the QM application may get those coordinates:

```
dataprep_get(&MM_data);
```

Other key methods assist with sharing data in multiple-binary MPI applications by either yielding CPU resources and waiting for a UNIX signal or putting data then signaling another program. For example, one can easily accomplish a simple ping-pong style passing between QM and MM programs:

```
/* QM application work: */
for (int i=0; i<ITERS; i++) {
  /* Do QM calculations here... */
  /* Put data and wait for MM put */
  dataprep_put_then_signal(&qmmdata);
  dataprep_get_wait(&qmmdata); }
```

```
/* MM application work: */
for (int i=0; i<ITERS; i++) {
  dataprep_get_wait(&qmmdata);
  /* Do MM calculations here... */
  /* Put data and wait for QM put */
  dataprep_put_then_signal(&qmmdata); }
```

The DataPrep library makes use of the ADIOS read/write API [234] for packaging and transporting atomic simulation data. By leveraging the features

of ADIOS, DataPrep can support several languages, modern network interconnects, and data types with relatively high portability. DataPrep currently provides the `QMM_Data` structure as a C struct, a C++ object, and Fortran derived datatype, and it is straightforward to pass this data structure between multi-language applications - ADIOS takes care of the individual datatype sizes, byte alignments, and data transport mechanisms. The DataPrep abstraction hides the complexity of the ADIOS implementation: it specifies ADIOS transport configuration files that fit the QM/MM/CG paradigm and generates the necessary ADIOS read/write routines, buffer specifications, language particulars, and synchronization strategies. If users desire an augmented data structure, it is a relatively easy update to the DataPrep library to update the relevant ADIOS configuration XML files and necessary packing/unpacking to support the additional data.

4.3 QM/MM Experiments

This section describes a series of experiments that expose issues with the parallel performance of both single and multiple binary implementations of QM/MM applications. These experiments were run on two different computer clusters: the ACISS cluster located at the University of Oregon and the Comet cluster located at the San Diego Supercomputing Center. ACISS experiments were run on the generic compute nodes, each having 2 Intel X5650 2.67 GHz 6-core CPUs (12 processor cores total) per node and 72 GB of memory per node. ACISS has a 10 gigabit Ethernet interconnect based on a 1-1 nonblocking Voltaire 8500 10 GigE switch that connects all compute nodes and storage fabric. The operating system is RedHat Enterprise Linux 6.2, and MPICH 3.1 is used with the GCC 4.4.7 compiler collection and default optimizations. Comet experiments were run on the

"compute" queue nodes, each having 2 Intel Xeon E5-2680v3 12-core CPUs (24 processor cores total) per node and 128 GB of DDR4 DRAM per node. Comet has a 56 Gbps bidirectional InfiniBand FDR interconnect with a fat-tree topology.

As made clear in the previous section, there is a plethora of existing QM/MM implementations. Instead of including a performance analysis of *all* these software implementations, the following subsections instead focus on the LAMMPS and Quantum ESPRESSO coupling, as realized by both the USER-QMMM and QMMMW packages. These two implementations of QM/MM make for an interesting juxtaposition because they conduct the same calculation, yet have two very different software architectures and designs. USER-QMMM is a single-binary application with an external library design, and all distributions of the LAMMPS source code include it as an optional package. It relies on MPI intercommunicators (one each for QM, MM, and a "slave" component) for data passing and component synchronization. QMMMW is closely based on the USER-QMMM implementation, but instead *separates* the applications into two different binaries and exposes a useful API for supporting data transfers between the QM and MM components [235]. Both USER-QMMM and QMMMW follow a LAMMPS-master, LAMMPS-slave, and quantum component architecture in which data is shared across these three modules. In the opinion of this author, the "slave" component is a bit of a misnomer (it calculates the MM forces on the QM subsystem), so it is instead referred to as the MM→QM component in this section. For clarity and brevity, it is helpful to limit the scope of our study to these two packages, but also allows us to focus on *both* the single and multiple binary architectures, as described in section 4.1.1.. This also highlights the more

FIGURE 57. A ray-traced visualization of the 10^3 water system used in this experiment. The enlarged water molecule in the middle of the simulation box is modeled with QM, and all other molecules with MM.

preferable (yet more challenging to manage) external linking architecture, in hopes of motivating future work and collaboration on supporting these packages.

4.3.1. Water QM/MM Relative Size Experiment

This first experiment involves profiling the execution time within the LAMMPS/Quantum ESPRESSO QM/MM coupled application, and to determine how best to assign processor resources to the individual components. In other words, the goal of this experiment is to determine whether or not the QM component is always the computational bottleneck, or similarly, how big the MM system must be before needing to assign more threads/processes to it. The TAU performance measurement system [168] was used to measure the time spent in various functions of the coupled application, and expose the three most time-consuming functions (which completely dwarf all other functions). To keep things

simple and easy to reproduce, the chemical system studied in this experiment is a simple collection of water molecules, an example of which is included with both the USER-QMMM and QMMMW software packages source code repositories*. A visualization of this chemical system is shown in Figure 57, where one central molecule is modeled with QM and the rest are modeled with MM. Our experiments run mechanical coupling (but electrostatic coupling is also supported by USER-QMMM), the QM model is plane-wave DFT, and a simple Lennard-Jones intermolecular pair potential with harmonic bond and bond angle potentials. In addition, a very minor change was made to the source code to fix an MPI bug which resulted in a segmentation fault on certain systems (please contact the author if interested in the patch).

The results are shown in Figures 58 and 59. These experiments show profile data for three different allocations of MPI processes on a single compute node on the Comet cluster:

1. 22 QM processes, 1 MM process, 1 MM→QM process
2. 12 QM processes, 11 MM processes, 1 MM→QM process
3. 1 QM process, 22 MM processes, 1 MM→QM process

In Figure 58, the left panel shows allocation #1, the middle panel shows allocation #2, and the right panel shows allocation #3. Only 1 process is ever assigned to the MM→QM component because it is extremely inexpensive and the time spent there is always negligible compared to the other components. Although LAMMPS and Quantum ESPRESSO both support OpenMP, our measurements suggest that the best performance is achieved with 1-2 threads per MPI process, so we

*At the time of this writing, the input files and parameters for this water-cluster example can most easily be found here:
`https://github.com/lammps/lammps/tree/master/lib/qmmm/example-mc`

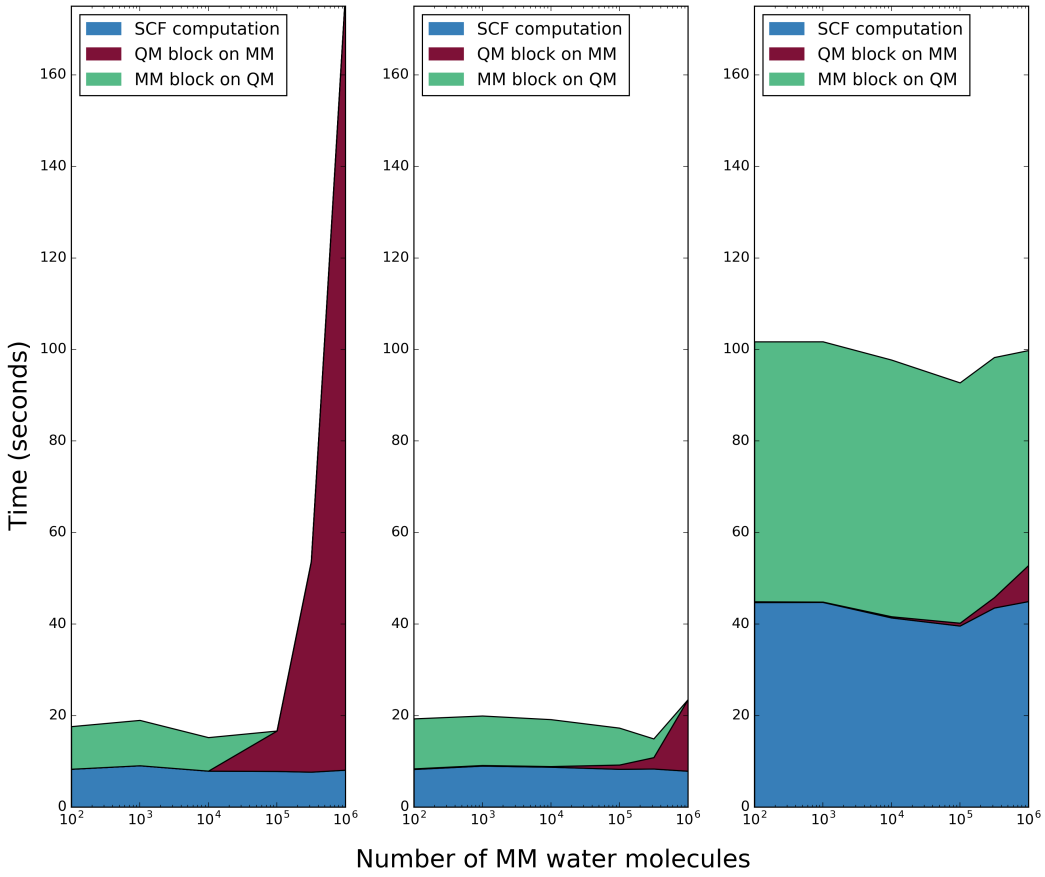


FIGURE 58. A comparison of the LAMMPS USER-QMMM bottleneck routines as the number of MM water molecules increases on a single Comet compute node. The left subplot shows simulations that allocated 22 processes for QM calculations and 1 process for MM. The center subplot allocated 12 QM processes and 11 MM processes, and the right subplot had 1 QM process and 22 MM processes. All runs reserve 1 MM→QM process and run 1 OpenMP thread per process.

choose `OMP_NUM_THREADS=1`. This data includes simulations from the USER-QMMM package only, but the performance of QMMMW is almost exactly the same as the single MM process case (because currently, having multiple MM processes is only supported in QMMMW via a shared networked file system).

The left panel of Figure 58 suggests that allocation #1 has good performance for small MM system sizes, but as more MM water molecules are added, an exorbitant amount of time is spent in QM processes blocking on MM calculations

(in USER-QMMM this is a single `MPI_Recv` per iteration). This occurs because a single MM process is not sufficient to track large MM system sizes, and the QM MPI communicator blocks until the MM calculations finish. The middle panel shows acceptable performance across all MM system sizes for the more balanced allocation #2, and only mild degradation for very small numbers of MM water molecules. At the very least, this result motivates the need for a parallel MM component (for which support was unfortunately *removed* in the latest version of USER-QMMM, as of this writing). Finally, the right panel shows that performance deteriorates severely when only 1 process is allocated to the QM component. If this is clear for only a single water molecule in the QM region, it would surely become more prominent for larger and more interesting QM systems.

Figure 59 shows data from the same simulations as Figure 58, but instead shows the total execution time of the QM/MM calculation from start to finish for each of the three allocations. As we might expect, the best overall performance across various MM system sizes is with the balanced allocation #2. Except for very large MM sizes, allocation #1 is better than allocation #3. Interestingly, allocation #1 shows better performance then allocation #2 only for instances with a very small MM size. This reaffirms the instinct that the QM component is the most important to parallelize when the number of MM molecules is relatively low. But this allocation also suffers from extreme performance degradation when the size of the MM system is relatively very large. This suggests that being able to control resource allocations in coupled QM/MM applications is paramount, and that the best allocation given the QM and MM system sizes is apparently not entirely straightforward - even for this simple water cluster.

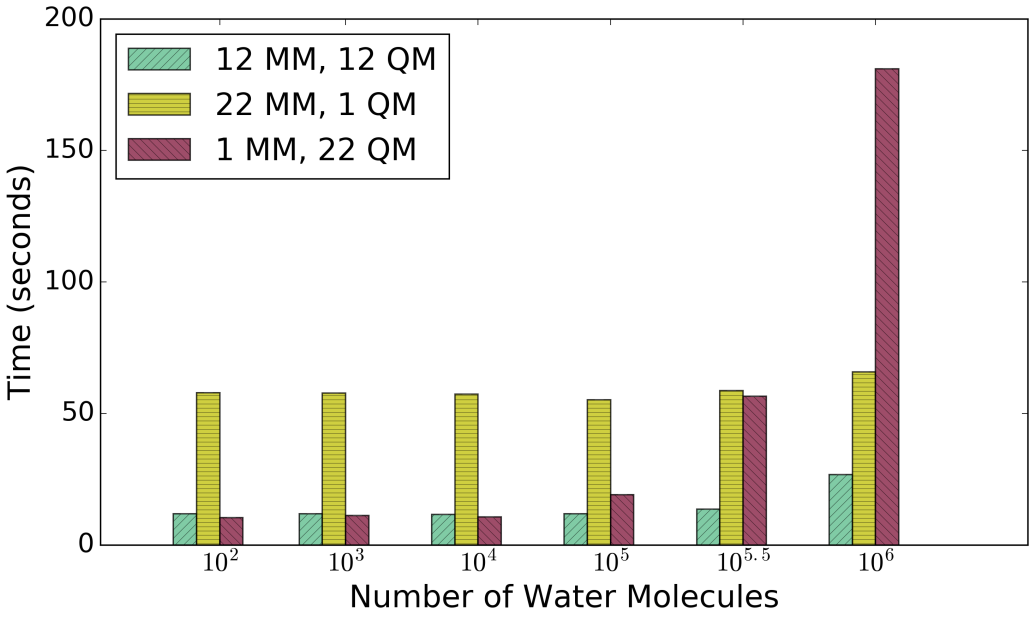


FIGURE 59. Total execution time for the experiments shown in Figure 58. The balanced process allocation shows the best performance with larger MM domains, but with fewer MM molecules, allocating more QM processes is advantageous.

4.3.2. Mini-App Barrier Experiment

The next experiment aims to expose a performance issue related to component coordination that is found directly in the USER-QMMM package, and is closely related to performance degradations seen in other QM/MM implementations, especially those that support iterative dynamics by passing control between externally linking QM and MM programs (such as GROMACS with GAMESS-UK). QMMMW does not exhibit this problem as readily, but this is because none of its transport mechanisms fully support a parallel MM component. The scaling measurements below also include an equivalent implementation that uses the DataPrep library (see section 4.2) to coordinate and facilitate data transfer between QM and MM components.

In the USER-QMMM package, for example, atomic position data is intermittently transferred between three MPI intercommunicators. While one component is busy with a calculation, a representative process from the other two intercommunicators blocks on an `MPI_Recv` while all other processes wait in an `MPI_Barrier`. Depending on the MPI implementation, runtime parameters, and the number of processes per compute node, this strategy may cause performance degradation in the active component because of busy-polling or intermittent-polling mechanisms in the barrier implementation. To expose this and analyze the consequences on performance, we consider a simple mini-app that launches two MPI applications, one that does a compute-intensive calculation. In this case, we solve Poisson's equation, $\left(\frac{d^2}{dx^2}+\frac{d^2}{dy^2}\right)u(x,y)=f(x,y)$, on a 2-dimensional domain using Jacobi iterations (which is meant to roughly resemble the QM calculation). The other application mimics a waiting MM application where one process waits for the Jacobi calculation to finish, while the other processes stall on an `MPI_Barrier`.

Figures 60 and 61 show the results of a scaling experiment on the ACISS and Comet clusters, respectively. Each line on the plot corresponds to a run with a number of processes per compute node waiting in a barrier while the other application performs a compute-intensive calculation. On ACISS, the compute-intensive application is always assigned 1 process and 12 OpenMP threads, and on Comet, 1 process and 24 OpenMP threads, because those assignments exhibit the best stand-alone performance. The vertical axis shows the slowdown of the Jacobi application defined by the measured time with the barrier application divided by the stand-alone time. Each data point corresponds to the minimum of 5 executions to reduce the noise of the measurements.

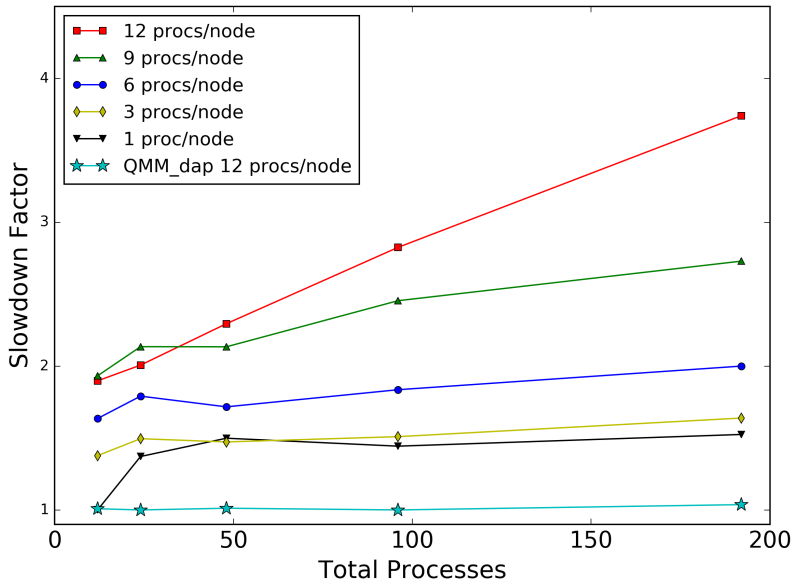


FIGURE 60. Slowdown of the multiple binary architecture when one application is blocked in a barrier consuming compute resources. The effect is exacerbated when there are a large number of QM processes (such as in the left side of Figure 58). At the largest measured scale, DataPrep provides a ∼3.7x speedup when using 12 MM processes.

There are three salient features worth noting in these plots. First, the performance slowdown is exacerbated by the presence of more processes in the barrier-blocked application. Second, the performance generally deteriorates as the scale increases, and the effect is worse when the number of blocked processes is high. The first observation is clearly expected, because there are more processes that must poll to make progress, which consumes valuable resources for the compute-intensive application. The second observation is relatively more surprising, because it suggests that having more *off-node* blocked processes results in larger synchronization delays. This is sensible because the global barrier finishes after the *final* representative process receives a notice, and all other representatives must wait in the barrier until that time. Finally, we observe that the DataPrep implementation completely avoids these performance degradations, because it relies

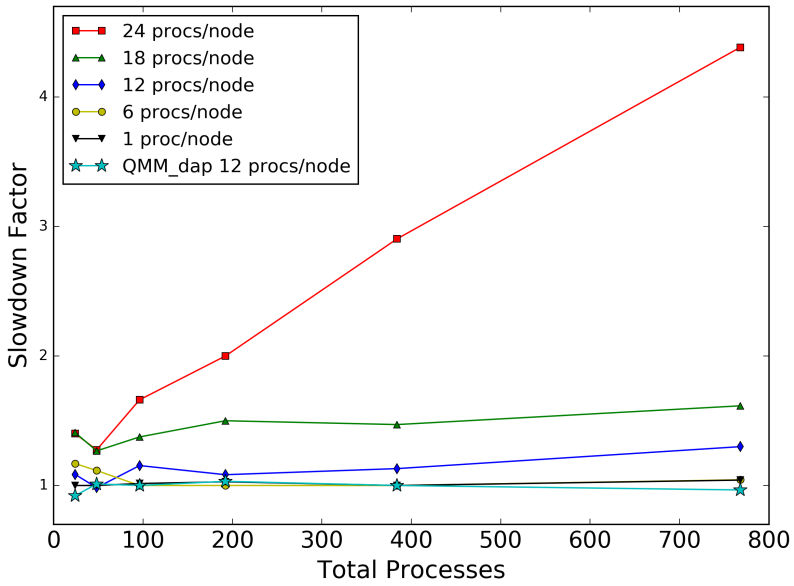


FIGURE 61. Similar experiment to Figure 60, but instead on the Comet cluster. Here, the effect is less detrimental than the Ethernet network for relatively comparable process counts, likely due to more efficient RDMA-based algorithms implemented in MVAPICH on this fabric [236, 237]. At the largest measured scale, DataPrep provides a ∼4.4x speedup when using 24 MM processes.

on synchronization via custom UNIX signal handling instead of polling. While the Jacobi calculations complete, the stalled application is completely suspended, and only awakens after the call to `dataprep_put_then_signal`. Also note that both plots only show the DataPrep result for the worst case - 12 suspended processes. All other numbers of processes also exhibit the same result, and they are excluded for clarity. In this worst case scenario, DataPrep shows a ∼3.7x speedup on ACISS and a ∼4.4x speedup on Comet.

CHAPTER V

CONCLUSION AND FUTURE WORK

Computational chemistry is naturally a multiscale problem: interesting length and time scales span many orders of magnitude, and bridging these scales remains an important unsolved problem. Chemical models of vastly different scale require vastly different theories, software implementations, parallel decompositions, algorithms, and parallel programming methods. This dissertation discussed the challenges within the most prevalent scales of modern molecular simulations: QM, MM, and CG. Software implementations and their utilization of HPC differ greatly across these scale domains, and new research is required to effectively bridge these scales in the face of extreme challenges as the community pursues the exascale computing objective.

In our consideration of QM codes, we learned that QM algorithms often suffer from extremely diverse communication requirements that worsen as we add more processes to the application. More work needs to be done to quantify the potential benefit of optimizing for locality, and how best to reduce communication overhead at scale. Furthermore, relatively little work has been done in QM on new computer architectures such as Intel's Knights Corner device. Exploring methods that optimize vector operations is remarkably important in light of the large acquisition of copious Xeon Phi resources (Cori, Aurora, and Trinity), which will all contain Knights Landing and Knights Hill architectures.

In our sections highlighting the characteristics of MD codes, we learned that many software frameworks are relatively mature, but may benefit from an updated perspective on their load balance strategies, their tuning of important parameters

such as the cutoff distance, and possibly utilizing data mining and machine learning techniques to alleviate the combinatorial explosion of simulation parameters and chemical system knowledge.

In the field of CG theory, we learned that there are several approaches to reducing the number of degrees of freedom from MD simulations. At the highest level, there are numerical and theoretical approaches. Broadly speaking, future work in numerical approaches should improve the accuracy of CG models while considering that introducing more parameters requires their tuning and sufficient training data. For theoretical methods, work remains to extend the models to more diverse chemical systems, such as block copolymers, and models that incorporate adaptive resolution.

In some sense, QM, MM, and CG codes have *individually* reached an appreciable state of maturity, but hybrid codes such as QM/MM/CG are relatively less supported, and the barrier of entry is too high for many computational chemists. Part of the reason for this is the novelty and diversity of many different multiresolution methods in chemistry. Much work remains to improve and eventually verify and validate these existing models, which requires supportive tools (such as VOTCA for CG) that allow for direct comparison of hybrid methods. The community would greatly benefit from more modular, extendible, and scalable software interfaces that enable exploration of different multiscale methods, and offer guidance in choosing the best fit for the chemistry problem at hand. Workflow systems management plays a crucial role in this regard, yet there is a general lack of support, adaptivity, performance metrics, and modularity in this field. The hardware infrastructure is available, we only require the committed development of new and extendible computational chemistry tools for multiscale simulations.

REFERENCES CITED

[1] D.P. Scarpazza, D.J. Ierardi, A.K. Lerer, K.M. Mackenzie, A.C. Pan, J.A. Bank, E. Chow, R.O. Dror, J.P. Grossman, D. Killebrew, M.A. Moraes, C. Predescu, J.K. Salmon, and D.E. Shaw. Extending the Generality of Molecular Dynamics Simulations on a Special-Purpose Machine. In *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 933–945, May 2013.

[2] D.E. Shaw, J.P. Grossman, J.A. Bank, B. Batson, J.A. Butts, J.C. Chao, M.M. Deneroff, R.O. Dror, A. Even, C.H. Fenton, A. Forte, J. Gagliardo, G. Gill, B. Greskamp, C.R. Ho, D.J. Ierardi, L. Iserovich, J.S. Kuskin, R.H. Larson, T. Layman, Li-Siang Lee, A.K. Lerer, C. Li, D. Killebrew, K.M. Mackenzie, S.Y.-H. Mok, M.A. Moraes, R. Mueller, L.J. Nociolo, J.L. Peticolas, T. Quan, D. Ramot, J.K. Salmon, D.P. Scarpazza, U.B. Schafer, N. Siddique, C.W. Snyder, J. Spengler, P.T.P. Tang, M. Theobald, H. Toma, B. Towles, B. Vitale, S.C. Wang, and C. Young. Anton 2: Raising the Bar for Performance and Programmability in a Special-Purpose Molecular Dynamics Supercomputer. In *High Performance Computing, Networking, Storage and Analysis, SC14: International Conference for*, pages 41–53, Nov 2014.

[3] Ian T. Foster, Jeffrey L. Tilson, Albert F. Wagner, Ron L. Shepard, Robert J. Harrison, Rick A. Kendall, and Rik J. Littlefield. Toward High-Performance Computational Chemistry: I. Scalable Fock Matrix Construction Algorithms. *Journal of Computational Chemistry*, 17(1):109–123, 1996.

[4] Robert J. Harrison, Martyn F. Guest, Rick A. Kendall, David E. Bernholdt, Adrian T. Wong, Mark Stave, James L. Anchell, Anthony C. Hess, Rik J. Littlefield, George L. Fann, Jaroslaw Nieplocha, Greg S. Thomas, David Elwood, Jeffrey L. Tilson, Ron L. Shepard, Albert F. Wagner, Ian T. Foster, Ewing Lusk, and Rick Stevens. Toward High-Performance Computational Chemistry: II. A Scalable Self-Consistent Field Program. *Journal of Computational Chemistry*, 17(1):124–132, 1996.

[5] Tim Meyer, Marco D'Abramo, Adam Hospital, Manuel Rueda, Carles Ferrer-Costa, Alberto Prez, Oliver Carrillo, Jordi Camps, Carles Fenollosa, Dmitry Repchevsky, Josep Lluis Gelp, and Modesto Orozco. MoDEL (Molecular Dynamics Extended Library): A Database of Atomistic Molecular Dynamics Trajectories. *Structure*, 18(11):1399 – 1409, 2010.

[6] Matthias Rupp, Alexandre Tkatchenko, Klaus-Robert Müller, and O. Anatole von Lilienfeld. Fast and Accurate Modeling of Molecular Atomization Energies with Machine Learning. *Phys. Rev. Lett.*, 108:058301, Jan 2012.

[7] Stefano Curtarolo, Dane Morgan, Kristin Persson, John Rodgers, and Gerbrand Ceder. Predicting Crystal Structures with Data Mining of Quantum Calculations. *Phys. Rev. Lett.*, 91:135503, Sep 2003.

[8] J Knap, C E Spear, O Borodin, and K W Leiter. Advancing a Distributed Multi-Scale Computing Framework for Large-scale High-throughput Discovery in Materials Science. *Nanotechnology*, 26(43):434004, 2015.

[9] Jianyin Shao et al. Clustering Molecular Dynamics Trajectories: 1. Characterizing the Performance of Different Clustering Algorithms. *Journal of Chemical Theory and Computation*, 3(6):2312–2334, 2007.

[10] V. Weber, C. Bekas, T. Laino, A. Curioni, A. Bertsch, and S. Futral. Shedding Light on Lithium/Air Batteries Using Millions of Threads on the BG/Q Supercomputer. In *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, pages 735–744, May 2014.

[11] Samyam Rajbhandari, Akshay Nikam, Pai-Wei Lai, Kevin Stock, Sriram Krishnamoorthy, and P. Sadayappan. A Communication-optimal Framework for Contracting Distributed Tensors. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '14, pages 375–386, Piscataway, NJ, USA, 2014. IEEE Press.

[12] A.G. Shet, W.R. Elwasif, R.J. Harrison, and D.E. Bernholdt. Programmability of the HPCS Languages: A Case Study with a Quantum Chemistry Kernel. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–8, April 2008.

[13] Carsten Hartmann and Luigi Delle Site. Scale Bridging in Molecular Simulation. *The European Physical Journal Special Topics*, 224(12):2173–2176, 2015.

[14] Billion-atom Lennard-Jones benchmarks, 2015. http://lammps.sandia.gov/bench.html#billion.

[15] Nobel Media AB 2014. Web. *The Nobel Prize in Chemistry 2013 - Press Release*. 2015.

[16] Pandian Sokkar, Eliot Boulanger, Walter Thiel, and Elsa Sanchez-Garcia. Hybrid Quantum Mechanics/Molecular Mechanics/Coarse Grained Modeling: A Triple-Resolution Approach for Biomolecular Systems. *Journal of Chemical Theory and Computation*, 11(4):1809–1818, 2015.

[17] Soroosh Pezeshki and Hai Lin. Recent Progress in Adaptive-Partitioning QM/MM Methods for Born-Oppenheimer Molecular Dynamics. *Quantum Modeling of Complex Molecular Systems*, 21:93, 2015.

[18] D. Ozog, J.R. Hammond, J. Dinan, P. Balaji, S. Shende, and A. Malony. Inspector-Executor Load Balancing Algorithms for Block-Sparse Tensor Contractions. In *Parallel Processing (ICPP), 2013 42nd International Conference on*, pages 30–39, Oct 2013.

[19] D. Ozog, A. Malony, J.R. Hammond, and P. Balaji. WorkQ: A Many-core Producer/Consumer Execution Model Applied to PGAS Computations. In *Parallel and Distributed Systems (ICPADS), 2014 20th IEEE International Conference on*, pages 632–639, Dec 2014.

[20] D. Ozog, A. Kamil, Y. Zheng, P. Hargrove, J. R. Hammond, A. Malony, W. d. Jong, and K. Yelick. A Hartree-Fock Application Using UPC++ and the New Darray Library. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 453–462, May 2016.

[21] David Ozog, Jay McCarty, Grant Gossett, Allen D. Malony, and Marina Guenza. Fast Equilibration of Coarse-Grained Polymeric Liquids. *2015 International Conference on Computational Science (ICCS 2015)*, 2015.

[22] David Ozog, Jay McCarty, Grant Gossett, Allen D. Malony, and Marina Guenza. Fast Equilibration of Coarse-Grained Polymeric Liquids. *Journal of Computational Science*, 9:33 – 38, 2015. Computational Science at the Gates of Nature.

[23] David Ozog, Allen Malony, and Marina Guenza. The UA/CG Workflow: High Performance Molecular Dynamics of Coarse-Grained Polymers. *Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, 2016.

[24] Hans Johansen, Lois Curfman McInnes, David E. Bernholdt, Jeffrey Carver, Michael Heroux, Richard Hornung, Phil Jones, Bob Lucas, Andrew Siegel, and Thomas Ndousse-Fetter. Software Productivity for Extreme-Scale Science. 2014.

[25] Hans Johansen, David E Bernholdt, Bill Collins, Michael Heroux SNL, Robert Jacob ANL, Phil Jones, Lois Curfman McInnes ANL, and J David Moulton. Extreme-Scale Scientific Application Software Productivity. 2013.

[26] Jack Dongarra et al. The International Exascale Software Project Roadmap. *International Journal of High Performance Computing Applications*, 2011.

[27] Saman Amarasinghe, Mary Hall, Richard Lethin, Keshav Pingali, Dan Quinlan, Vivek Sarkar, John Shalf, Robert Lucas, Katherine Yelick, Pavan Balaji, et al. Exascale Programming Challenges. In *Report of the 2011 Workshop on Exascale Programming Challenges, Marina del Rey*, 2011.

[28] Steve Ashby, P Beckman, J Chen, P Colella, B Collins, D Crawford, J Dongarra, D Kothe, R Lusk, P Messina, et al. The Opportunities and Challenges of Exascale Computing. *Summary Report of the Advanced Scientific Computing Advisory Committee (ASCAC) Subcommittee*, pages 1–77, 2010.

[29] Linus Pauling and Edgar Bright Wilson. *Introduction to Quantum Mechanics: with Applications to Chemistry*. Courier Corporation, 1985.

[30] Michel Dupuis and Antonio Marquez. The Rys Quadrature Revisited: A Novel Formulation for the Efficient Computation of Electron Repulsion Integrals over Gaussian Functions. *The Journal of Chemical Physics*, 114(5):2067–2078, 2001.

[31] Ivan S. Ufimtsev and Todd J. Martinez. Quantum Chemistry on Graphical Processing Units. 1. Strategies for Two-Electron Integral Evaluation. *Journal of Chemical Theory and Computation*, 4(2):222–231, 2008.

[32] Frank Jensen. *Introduction to Computational Chemistry*. John Wiley & Sons, 2013.

[33] Christopher J Cramer. *Essentials of Computational Chemistry: Theories and Models*. John Wiley & Sons, 2013.

[34] Hongzhang Shan, Brian Austin, Wibe De Jong, Leonid Oliker, N.J. Wright, and Edoardo Apra. Performance Tuning of Fock Matrix and Two-Electron Integral Calculations for Nwchem on Leading HPC Platforms. In Stephen A. Jarvis, Steven A. Wright, and Simon D. Hammond, editors, *High Performance Computing Systems. Performance Modeling, Benchmarking and Simulation*, Lecture Notes in Computer Science, pages 261–280. Springer International Publishing, 2014.

[35] Attila Szabo and Neil S Ostlund. *Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory*. Courier Corporation, 2012.

[36] Isaiah Shavitt and Rodney J Bartlett. *Many-body Methods in Chemistry and Physics: MBPT and Coupled-Cluster Theory*. Cambridge university press, 2009.

[37] Ken-ichi Nomura, Rajiv K. Kalia, Aiichiro Nakano, Priya Vashishta, Kohei Shimamura, Fuyuki Shimojo, Manaschai Kunaseth, Paul C. Messina, and Nichols A. Romero. Metascalable Quantum Molecular Dynamics Simulations of Hydrogen-on-demand. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '14, pages 661–673, Piscataway, NJ, USA, 2014. IEEE Press.

[38] Peter Deglmann, Ansgar Schfer, and Christian Lennartz. Application of Quantum Calculations in the Chemical Industry - An Overview. *International Journal of Quantum Chemistry*, 115(3):107–136, 2015.

[39] L. H. Thomas. The Calculation of Atomic Fields. *Proceedings of the Cambridge Philosophical Society*, 23:542, 1927.

[40] Pierre Hohenberg and Walter Kohn. Inhomogeneous Electron Gas. *Physical review*, 136(3B):B864, 1964.

[41] Wibe A. de Jong, Eric Bylaska, Niranjan Govind, Curtis L. Janssen, Karol Kowalski, Thomas Muller, Ida M. B. Nielsen, Hubertus J. J. van Dam, Valera Veryazov, and Roland Lindh. Utilizing High Performance Computing for Chemistry: Parallel Computational Chemistry. *Phys. Chem. Chem. Phys.*, 12:6896–6920, 2010.

[42] A. Bender, A. Poschlad, S. Bozic, and I. Kondov. A Service-oriented Framework for Integration of Domain-specific Data Models in Scientific Workflows. *2013 International Conference on Computational Science (ICCS 2013)*, 18(0):1087 – 1096, 2013.

[43] Kenneth P Esler, Jeongnim Kim, David M Ceperley, Wirawan Purwanto, Eric J Walter, Henry Krakauer, Shiwei Zhang, Paul RC Kent, Richard G Hennig, Cyrus Umrigar, et al. Quantum Monte Carlo Algorithms for Electronic Structure at the Petascale; the Endstation Project. In *Journal of Physics: Conference Series*, volume 125, page 012057. IOP Publishing, 2008.

[44] Amos Anderson, William A. Goddard, III, and Peter Schrder. Quantum Monte Carlo on Graphical Processing Units, 2007.

[45] A. Vishnu, J. Daily, and B. Palmer. Designing Scalable PGAS Communication Subsystems on Cray Gemini Interconnect. In *High Performance Computing (HiPC), 2012 19th International Conference on*, pages 1–10, Dec 2012.

[46] Edmond Chow, Xing Liu, Sanchit Misra, Marat Dukhan, Mikhail Smelyanskiy, Jeff R. Hammond, Yunfei Du, Xiang-Ke Liao, and Pradeep Dubey. Scaling up HartreeFock calculations on Tianhe-2. *International Journal of High Performance Computing Applications*, 2015.

[47] Graham D. Fletcher, Michael W. Schmidt, Brett M. Bode, and Mark S. Gordon. The Distributed Data Interface in GAMESS. *Computer Physics Communications*, 128(12):190 – 200, 2000.

[48] Xing Liu, A. Patel, and E. Chow. A New Scalable Parallel Algorithm for Fock Matrix Construction. In *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, pages 902–914, May 2014.

[49] Marat Valiev, Eric J Bylaska, Niranjan Govind, Karol Kowalski, Tjerk P Straatsma, Hubertus JJ Van Dam, Dunyou Wang, Jarek Nieplocha, Edoardo Apra, Theresa L Windus, et al. NWChem: a Comprehensive and Scalable Open-source Solution for Large Scale Molecular Simulations. *Computer Physics Communications*, 181(9):1477–1489, 2010.

[50] Y. Alexeev, A. Mahajan, S. Leyffer, G. Fletcher, and D.G. Fedorov. Heuristic Static Load-balancing Algorithm Applied to the Fragment Molecular Orbital Method. In *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, pages 1–13, Nov 2012.

[51] Pai-Wei Lai, Kevin Stock, Samyam Rajbhandari, Sriram Krishnamoorthy, and P. Sadayappan. A Framework for Load Balancing of Tensor Contraction Expressions via Dynamic Task Partitioning. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 13:1–13:10, New York, NY, USA, 2013. ACM.

[52] Curtis L Janssen and Ida MB Nielsen. *Parallel Computing in Quantum Chemistry*. CRC Press, 2008.

[53] Yili Zheng, A. Kamil, M.B. Driscoll, Hongzhang Shan, and K. Yelick. UPC++: A PGAS Extension for C++. In *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, pages 1105–1114, May 2014.

[54] Adam H. R. Palser and David E. Manolopoulos. Canonical Purification of the Density Matrix in Electronic-Structure Theory. *Phys. Rev. B*, 58:12704–12711, Nov 1998.

[55] E. Solomonik, D. Matthews, J.R. Hammond, and J. Demmel. Cyclops Tensor Framework: Reducing Communication and Eliminating Load Imbalance in Massively Parallel Contractions. In *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 813–824, May 2013.

[56] H. McCraw, A. Danalis, T. Herault, G. Bosilca, J. Dongarra, K. Kowalski, and T.L. Windus. Utilizing Dataflow-based Execution for Coupled Cluster Methods. In *Cluster Computing (CLUSTER), 2014 IEEE International Conference on*, pages 296–297, Sept 2014.

[57] Ivan S Ufimtsev and Todd J Martinez. Quantum Chemistry on Graphical Processing Units. 2. Direct Self-Consistent-Field Implementation. *Journal of Chemical Theory and Computation*, 5(4):1004–1015, 2009.

[58] Ivan S Ufimtsev and Todd J Martinez. Quantum Chemistry on Graphical Processing Units. 3. Analytical Energy Gradients, Geometry Optimization, and First Principles Molecular Dynamics. *Journal of Chemical Theory and Computation*, 5(10):2619–2628, 2009.

[59] Andreas W Götz, Thorsten Wölfle, and Ross C Walker. Quantum Chemistry on Graphics Processing Units. *Annual Reports in Computational Chemistry*, 6:21–35, 2010.

[60] Mark Berger. *NVIDIA Computational Chemistry and Biology*. November 2015.

[61] III A. Eugene DePrince and Jeff R. Hammond. Coupled Cluster Theory on Graphics Processing Units i. The Coupled Cluster Doubles Method. *Journal of Chemical Theory and Computation*, 7(5):1287–1295, 2011. PMID: 26610123.

[62] Wenjing Ma, Sriram Krishnamoorthy, Oreste Villa, Karol Kowalski, and Gagan Agrawal. Optimizing Tensor Contraction Expressions for Hybrid CPU-GPU Execution. *Cluster Computing*, 16(1):131–155, 2013.

[63] Edoardo Aprà, Michael Klemm, and Karol Kowalski. Efficient Implementation of Many-body Quantum Chemical Methods on the Intel®Xeon Phi®Coprocessor. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '14, pages 674–684, Piscataway, NJ, USA, 2014. IEEE Press.

[64] Hongzhang Shan, Samuel Williams, Wibe de Jong, and Leonid Oliker. Thread-level Parallelization and Optimization of Nwchem for the Intel MIC Architecture. In *Proceedings of the Sixth International Workshop on Programming Models and Applications for Multicores and Manycores*, PMAM '15, pages 58–67, New York, NY, USA, 2015. ACM.

[65] Alexey V. Titov, Ivan S. Ufimtsev, Nathan Luehr, and Todd J. Martinez. Generating Efficient Quantum Chemistry Codes for Novel Architectures. *Journal of Chemical Theory and Computation*, 9(1):213–221, 2013.

[66] John C. Linford, John Michalakes, Manish Vachharajani, and Adrian Sandu. Multi-core Acceleration of Chemical Kinetics for Simulation and Prediction. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 7:1–7:11, New York, NY, USA, 2009. ACM.

[67] S. Herres-Pawlis, A. Hoffmann, R. Grunzke, W.E. Nagel, L. De La Garza, J. Kruger, G. Terstyansky, N. Weingarten, and S. Gesing. Meta-Metaworkflows for Combining Quantum Chemistry and Molecular Dynamics in the MoSGrid Science Gateway. In *Science Gateways (IWSG), 2014 6th International Workshop on*, pages 73–78, June 2014.

[68] S. Herres-Pawlis, A. Hoffmann, L. De La Garza, J. Kruger, S. Gesing, and R. Grunzke. User-friendly Metaworkflows in Quantum Chemistry. In *Cluster Computing (CLUSTER), 2013 IEEE International Conference on*, pages 1–3, Sept 2013.

[69] Sonja Herres-Pawlis, Alexander Hoffmann, kos Balask, Peter Kacsuk, Georg Birkenheuer, Andr Brinkmann, Luis de la Garza, Jens Krger, Sandra Gesing, Richard Grunzke, Gabor Terstyansky, and Noam Weingarten. Quantum Chemical Meta-workflows in MoSGrid. *Concurrency and Computation: Practice and Experience*, pages n/a–n/a, 2014.

[70] Knordlun. *Molecular Dynamics Algorithm*. 2015.

[71] Mike P Allen and Dominic J Tildesley. *Computer Simulation of Liquids*. Oxford university press, 1989.

[72] Johannes Hachmann, Roberto Olivares-Amaya, Adrian Jinich, Anthony L. Appleton, Martin A. Blood-Forsythe, Laszlo R. Seress, Carolina Roman-Salgado, Kai Trepte, Sule Atahan-Evrenk, Suleyman Er, Supriya Shrestha, Rajib Mondal, Anatoliy Sokolov, Zhenan Bao, and Alan Aspuru-Guzik. Lead Candidates for High-Performance Organic Photovoltaics from High-throughput Quantum Chemistry - the Harvard Clean Energy Project. *Energy Environ. Sci.*, 7:698–704, 2014.

[73] Stefano Curtarolo, Gus LW Hart, Marco Buongiorno Nardelli, Natalio Mingo, Stefano Sanvito, and Ohad Levy. The High-throughput Highway to Computational Materials Design. *Nature materials*, 12(3):191–201, 2013.

[74] Tom M Mitchell. *Machine Learning*. McGraw-Hill Boston, MA:, 1997.

[75] James C Phillips, Rosemary Braun, Wei Wang, James Gumbart, Emad Tajkhorshid, Elizabeth Villa, Christophe Chipot, Robert D Skeel, Laxmikant Kale, and Klaus Schulten. Scalable Molecular Dynamics with NAMD. *Journal of computational chemistry*, 26(16):1781–1802, 2005.

[76] David A Pearlman, David A Case, James W Caldwell, Wilson S Ross, Thomas E Cheatham, Steve DeBolt, David Ferguson, George Seibel, and Peter Kollman. AMBER: A Package of Computer Programs for Applying Molecular Mechanics, Normal Mode Analysis, Molecular Dynamics and Free Energy Calculations to Simulate the Structural and Energetic Properties of Molecules. *Computer Physics Communications*, 91(1):1–41, 1995.

[77] Bernard R Brooks, Charles L Brooks, Alexander D MacKerell, Lennart Nilsson, Robert J Petrella, Benoît Roux, Youngdo Won, Georgios Archontis, Christian Bartels, Stefan Boresch, et al. CHARMM: The Biomolecular Simulation Program. *Journal of computational chemistry*, 30(10):1545–1614, 2009.

[78] Simpatico: Simulation Package for Polymer and Molecular Liquids (2015), 2015. `http://research.cems.umn.edu/morse/code/simpatico/home.php`.

[79] Jonathan D. Halverson, Thomas Brandes, Olaf Lenz, Axel Arnold, Sta Bevc, Vitaliy Starchenko, Kurt Kremer, Torsten Stuehn, and Dirk Reith. ESPResSo++: A Modern Multiscale Simulation Package for Soft Matter Systems. *Computer Physics Communications*, 184(4):1129 – 1149, 2013.

[80] Steve Plimpton. Fast Parallel Algorithms for Short-Range Molecular Dynamics. *Journal of Computational Physics*, 117(1):1 – 19, 1995.

[81] Simon J. Pennycook, Chris J. Hughes, M. Smelyanskiy, and S. A. Jarvis. Exploring SIMD for Molecular Dynamics, Using Intel®Xeon®Processors and Intel®Xeon Phi®Coprocessors. In *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, IPDPS '13, pages 1085–1097, Washington, DC, USA, 2013. IEEE Computer Society.

[82] Hossein Talebi, Mohammad Silani, Stphane P.A. Bordas, Pierre Kerfriden, and Timon Rabczuk. A Computational Library for Multiscale Modeling of Material Failure. *Computational Mechanics*, 53(5):1047–1071, 2014.

[83] Daan Frenkel and Berend Smit. *Understanding Molecular Simulation: From Algorithms to Applications*, volume 1. Academic press, 2001.

[84] Mark James Abraham, Teemu Murtola, Roland Schulz, Szilrd Pll, Jeremy C. Smith, Berk Hess, and Erik Lindahl. GROMACS: High Performance Molecular Simulations through Multi-level Parallelism from Laptops to Supercomputers. *SoftwareX*, 12:19 – 25, 2015.

[85] Sander Pronk, Szilrd Pll, Roland Schulz, Per Larsson, Pr Bjelkmar, Rossen Apostolov, Michael R. Shirts, Jeremy C. Smith, Peter M. Kasson, David van der Spoel, Berk Hess, and Erik Lindahl. GROMACS 4.5: a High-throughput and Highly Parallel Open Source Molecular Simulation Toolkit. *Bioinformatics*, 2013.

[86] L.V. Kal, M. Bhandarkar, R. Brunner, N. Krawetz, J. Phillips, and A. Shinozaki. NAMD: A Case Study in Multilingual Parallel Programming. In Zhiyuan Li, Pen-Chung Yew, Siddharta Chatterjee, Chua-Huang Huang, P. Sadayappan, and David Sehr, editors, *Languages and Compilers for Parallel Computing*, volume 1366 of *Lecture Notes in Computer Science*, pages 367–381. Springer Berlin Heidelberg, 1998.

[87] J Villareal, John Cortes, and W Najjar. Compiled Code Acceleration of NAMD on FPGAs. *Proceedings of the Reconfigurable Systems Summer Institute*, 2007.

[88] Lisandro D. Dalcin, Rodrigo R. Paz, Pablo A. Kler, and Alejandro Cosimo. Parallel Distributed Computing Using Python. *Advances in Water Resources*, 34(9):1124 – 1139, 2011. New Computational Methods and Software Tools.

[89] Frank S Bates. Block Copolymers Near the Microphase Separation Transition. 2. Linear Dynamic Mechanical Properties. *Macromolecules*, 17(12):2607–2613, 1984.

[90] Sanat K. Kumar, Nicolas Jouault, Brian Benicewicz, and Tony Neely. Nanocomposites with Polymer Grafted Nanoparticles. *Macromolecules*, 46(9):3199–3214, 2013.

[91] Anton Smessaert and Jörg Rottler. Recovery of Polymer Glasses from Mechanical Perturbation. *Macromolecules*, 45(6):2928–2935, 2012.

[92] J. Baschnagel, K. Binder, P. Doruker, A. Gusev, O. Hahn, K. Kremer, W. Mattice, F. Müller-Plathe, M. Murat, W. Paul, S. Santos, U. Suter, and V. Tries. Bridging the Gap Between Atomistic and Coarse-Grained Models of Polymers: Status and Perspectives. In *Viscoelasticity, Atomistic Models, Statistical Chemistry*, volume 152 of *Advances in Polymer Science*, pages 41–156. 2000.

[93] Victor Rhle, Christoph Junghans, Alexander Lukyanov, Kurt Kremer, and Denis Andrienko. Versatile Object-Oriented Toolkit for Coarse-Graining Applications. *Journal of Chemical Theory and Computation*, 5(12):3211–3223, 2009.

[94] Timothy C. Moore, Christopher R. Iacovella, and Clare McCabe. Derivation of Coarse-Grained Potentials via Multistate Iterative Boltzmann Inversion. *The Journal of Chemical Physics*, 140(22), 2014.

[95] Andrew Peacock. *Handbook of Polyethylene: Structures, Properties, and Applications*. CRC Press, 2000.

[96] E. J. Sambriski and M. G. Guenza. Theoretical Coarse-Graining Approach to Bridge Length Scales in Diblock Copolymer Liquids. *Phys. Rev. E*, 76:051801, Nov 2007.

[97] A. J. Clark and M. G. Guenza. Mapping of Polymer Melts onto Liquids of Soft-Colloidal Chains. *The Journal of Chemical Physics*, 132(4):–, 2010.

[98] G. Yatsenko, E. J. Sambriski, M. A. Nemirovskaya, and M. Guenza. Analytical Soft-Core Potentials for Macromolecular Fluids and Mixtures. *Phys. Rev. Lett.*, 93:257803, Dec 2004.

[99] I. Y. Lyubimov and M. G. Guenza. Theoretical Reconstruction of Realistic Dynamics of Highly Coarse-Grained cis-1,4-polybutadiene Melts. *The Journal of Chemical Physics*, 138(12):120000, March 2013.

[100] I. Lyubimov and M. G. Guenza. First-principle Approach to Rescale the Dynamics of Simulated Coarse-Grained Macromolecular Liquids. *Phys. Rev. E*, 84:031801, Sep 2011.

[101] J. McCarty and M. Guenza. Multiscale Modeling of Binary Polymer Mixtures: Scale Bridging in the Athermal and Thermal Regime. *The Journal of Chemical Physics*, 133(9):094904, 2010.

[102] A. J. Clark, J. McCarty, and M. G. Guenza. Effective Potentials for Representing Polymers in Melts as Chains of Interacting Soft Particles. *The Journal of Chemical Physics*, 139(12):–, 2013.

[103] J. McCarty, A. J. Clark, J. Copperman, and M. G. Guenza. An Analytical Coarse-Graining Method which Preserves the Free Energy, Structural Correlations, and Thermodynamic State of Polymer Melts from the Atomistic to the Mesoscale. *The Journal of Chemical Physics*, 140(20):–, 2014.

[104] A. Clark, J. McCarty, I. Lyubimov, and M. Guenza. Thermodynamic Consistency in Variable-Level Coarse Graining of Polymeric Liquids. *Phys. Rev. Lett.*, 109:168301, Oct 2012.

[105] James F. Dama, Anton V. Sinitskiy, Martin McCullagh, Jonathan Weare, Benot Roux, Aaron R. Dinner, and Gregory A. Voth. The Theory of Ultra-Coarse-Graining. 1. General Principles. *Journal of Chemical Theory and Computation*, 9(5):2466–2480, 2013.

[106] Dirk Reith, Mathias Pütz, and Florian Müller-Plathe. Deriving Effective Mesoscale Potentials from Atomistic Simulations. *Journal of Computational Chemistry*, 24(13):1624–1636, 2003.

[107] Siewert J. Marrink and D. Peter Tieleman. Perspective on the Martini Model. *Chem. Soc. Rev.*, 42:6801–6822, 2013.

[108] Luigi Delle Site. What is a Multiscale Problem in Molecular Dynamics? *Entropy*, 16(1):23, 2013.

[109] M.F. Horstemeyer. Multiscale Modeling: A Review. In Jerzy Leszczynski and Manoj K. Shukla, editors, *Practical Aspects of Computational Chemistry*, pages 87–135. Springer Netherlands, 2010.

[110] S.A. Baeurle. Multiscale Modeling of Polymer Materials using Field-theoretic Methodologies: A Survey about Recent Developments. *Journal of Mathematical Chemistry*, 46(2):363–426, 2009.

[111] Stefan Dapprich, Istvn Komromi, K.Suzie Byun, Keiji Morokuma, and Michael J Frisch. A New ONIOM Implementation in Gaussian98. Part i. The Calculation of Energies, Gradients, Vibrational Frequencies and Electric Field Derivatives. *Journal of Molecular Structure: Theoretical Chemistry*, 461462:1 – 21, 1999.

[112] Hai Lin and Donald G. Truhlar. QM/MM: What Have We Learned, Where Are We, and Where Do We Go from Here? *Theoretical Chemistry Accounts*, 117(2):185–199, 2007.

[113] Nandun M. Thellamurege, Dejun Si, Fengchao Cui, Hongbo Zhu, Rui Lai, and Hui Li. QuanPol: A Full Spectrum and Seamless QM/MM Program. *Journal of Computational Chemistry*, 34(32):2816–2833, 2013.

[114] Gerrit Groenhof. Introduction to QM/MM Simulations. In Luca Monticelli and Emppu Salonen, editors, *Biomolecular Simulations*, volume 924 of *Methods in Molecular Biology*, pages 43–66. Humana Press, 2013.

[115] Arieh Warshel and Michael Levitt. Theoretical Studies of Enzymic Reactions: Dielectric, Electrostatic and Steric Stabilization of the Carbonium Ion in the Reaction of Lysozyme. *Journal of molecular biology*, 103(2):227–249, 1976.

[116] Sebastian Metz, Johannes Kstner, Alexey A. Sokol, Thomas W. Keal, and Paul Sherwood. ChemShell: A Modular Software Package for QM/MM Simulations. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 4(2):101–110, 2014.

[117] Michael Levitt and Arieh Warshel. Computer Simulation of Protein Folding. *Nature*, 253(5494):694–698, 1975.

[118] Matej Praprotnik, Kurt Kremer, and Luigi Delle Site. Adaptive Molecular Resolution via a Continuous Change of the Phase Space Dimensionality. *Phys. Rev. E*, 75:017701, Jan 2007.

[119] Matej Praprotnik, Luigi Delle Site, and Kurt Kremer. Adaptive Resolution Molecular-Dynamics Simulation: Changing the Degrees of Freedom on the Fly. *The Journal of Chemical Physics*, 123(22):–, 2005.

[120] Matej Praprotnik, Luigi Delle Site, and Kurt Kremer. Multiscale Simulation of Soft Matter: From Scale Bridging to Adaptive Resolution. *Annual Review of Physical Chemistry*, 59(1):545–571, 2008. PMID: 18062769.

[121] Julija Zavadlav, Manuel N. Melo, Ana V. Cunha, Alex H. de Vries, Siewert J. Marrink, and Matej Praprotnik. Adaptive Resolution Simulation of MARTINI Solvents. *Journal of Chemical Theory and Computation*, 10(6):2591–2598, 2014.

[122] Julija Zavadlav, Manuel Nuno Melo, Siewert J. Marrink, and Matej Praprotnik. Adaptive Resolution Simulation of an Atomistic Protein in MARTINI Water. *The Journal of Chemical Physics*, 140(5):–, 2014.

[123] L. Delle Site, A. Agarwal, C. Junghans, and H. Wang. Adaptive Resolution Simulation as a Grand Canonical Molecular Dynamics Scheme: Principles, Applications and Perspectives. *ArXiv e-prints*, December 2014.

[124] Alexander B. Kuhn, Srinivasa M. Gopal, and Lars V. Schfer. On Using Atomistic Solvent Layers in Hybrid All-Atom/Coarse-Grained Molecular Dynamics Simulations. *Journal of Chemical Theory and Computation*, 11(9):4460–4472, 2015.

[125] Sebastian Fritsch, Christoph Junghans, and Kurt Kremer. Structure Formation of Toluene around C60: Implementation of the Adaptive Resolution Scheme (AdResS) into GROMACS. *Journal of Chemical Theory and Computation*, 8(2):398–403, 2012.

[126] Cameron Abrams, Luigi Delle Site, and Kurt Kremer. Multiscale Computer Simulations for Polymeric Materials in Bulk and Near Surfaces. In Peter Nielaba, Michel Mareschal, and Giovanni Ciccotti, editors, *Bridging Time Scales: Molecular Simulations for the Next Decade*, volume 605 of *Lecture Notes in Physics*, pages 143–164. Springer Berlin Heidelberg, 2002.

[127] H. Wang and A. Agarwal. Adaptive Resolution Simulation in Equilibrium and Beyond. *The European Physical Journal Special Topics*, 224(12):2269–2287, 2015.

[128] S. Fritsch, S. Poblete, C. Junghans, G. Ciccotti, L. Delle Site, and K. Kremer. Adaptive Resolution Molecular Dynamics Simulation through Coupling to an Internal Particle Reservoir. *Phys. Rev. Lett.*, 108:170602, Apr 2012.

[129] Animesh Agarwal and Luigi Delle Site. Path Integral Molecular Dynamics within the Grand Canonical-like Adaptive Resolution Technique: Simulation of Liquid Water. *The Journal of Chemical Physics*, 143(9):–, 2015.

[130] Marcus Bockmann, Dominik Marx, Christine Peter, Luigi Delle Site, Kurt Kremer, and Nikos L. Doltsinis. Multiscale Modelling of Mesoscopic Phenomena Triggered by Quantum Events: Light-driven Azo-materials and Beyond. *Phys. Chem. Chem. Phys.*, 13:7604–7621, 2011.

[131] Simn Poblete, Matej Praprotnik, Kurt Kremer, and Luigi Delle Site. Coupling Different Levels of Resolution in Molecular Simulations. *The Journal of Chemical Physics*, 132(11), 2010.

[132] Tsjerk A. Wassenaar, Kristyna Pluhackova, Rainer A. Böckmann, Siewert J. Marrink, and D. Peter Tieleman. Going Backward: A Flexible Geometric Approach to Reverse Transformation from Coarse Grained to Atomistic Models. *Journal of Chemical Theory and Computation*, 10(2):676–690, 2014.

[133] Christine Peter and Kurt Kremer. Multiscale Simulation of Soft Matter Systems - from the Atomistic to the Coarse-Grained Level and Back. *Soft Matter*, 5:4357–4366, 2009.

[134] Zahid Mahimwalla, KevinG. Yager, Jun-ichi Mamiya, Atsushi Shishido, Arri Priimagi, and ChristopherJ. Barrett. Azobenzene Photomechanics: Prospects and Potential Applications. *Polymer Bulletin*, 69(8):967–1006, 2012.

[135] K. Jorissen, F.D. Vila, and J.J. Rehr. A High Performance Scientific Cloud Computing Environment for Materials Simulations. *Computer Physics Communications*, 183(9):1911 – 1919, 2012.

[136] Robert J. Harrison. Portable Tools and Applications for Parallel Computers. 40(6):847–863, 1991.

[137] E. J. Bylaska et. al. NWChem, A Computational Chemistry Package for Parallel Computers, Version 6.1.1, 2012.

[138] Jiří Čížek. On the Correlation Problem in Atomic and Molecular Systems. Calculation of Wavefunction Components in Ursell-Type Expansion Using Quantum-Field Theoretical Methods. 45(11):4256–4266, December 1966.

[139] T. Daniel Crawford and Henry F. Schaefer III. An Introduction to Coupled Cluster Theory for Computational Chemists. volume 14, chapter 2, pages 33–136. VCH Publishers, New York, 2000.

[140] Rodney J. Bartlett and Monika Musiał. Coupled-Cluster Theory in Quantum Chemistry. 79(1):291–352, 2007.

[141] Rodney J. Bartlett. Coupled-Cluster Approach to Molecular Structure and Spectra: A Step Toward Predictive Quantum Chemistry. 93(5):1697–1708, 1989.

[142] George D. Purvis III and Rodney J. Bartlett. A Full Coupled-Cluster Singles and Doubles Model: The Inclusion of Disconnected Triples. 76(4):1910–1918, 1982.

[143] Miroslav Urban, Jozef Noga, Samuel J. Cole, and Rodney J. Bartlett. Towards a Full CCSDT Model for Electron Correlation. 83(8):4041–4046, 1985.

[144] Krishnan Raghavachari, Gary W. Trucks, John A. Pople, and Martin Head-Gordon. A Fifth-order Perturbation Comparison of Electron Correlation Theories. 157:479–483, May 1989.

[145] John F. Stanton. Why CCSD(T) Works: a Different Perspective. 281:130–134, 1997.

[146] Jozef Noga and Rodney J. Bartlett. The Full CCSDT Model for Molecular Electronic Structure. 86(12):7041–7050, 1987.

[147] John D. Watts and Rodney J. Bartlett. The Coupled-Cluster Single, Double, and Triple Excitation Model for Open-shell Single Reference Functions. 93(8):6104–6105, 1990.

[148] S.A. Kucharski and Rodney J. Bartlett. Recursive Intermediate Factorization and Complete Computational Linearization of the Coupled-Cluster Single, Double, Triple, and Quadruple Excitation Equations. 80:387–405, 1991. 10.1007/BF01117419.

[149] Stanislaw A. Kucharski and Rodney J. Bartlett. The Coupled-Cluster Single, Double, Triple, and Quadruple Excitation Method. 97(6):4282–4288, 1992.

[150] Nevin Oliphant and Ludwik Adamowicz. Coupled-Cluster Method Truncated at Quadruples. *The Journal of Chemical Physics*, 95(9):6645–6651, 1991.

[151] Stanislaw A. Kucharski and Rodney J. Bartlett. Coupled-Cluster Methods that Include Connected Quadruple Excitations, T_4: CCSDTQ-1 and Q(CCSDT). 158(6):550–555, 1989.

[152] Stanislaw A. Kucharski and Rodney J. Bartlett. An Efficient Way to Include Connected Quadruple Contributions into the Coupled Cluster Method. 108(22):9221–9226, 1998.

[153] Yannick J. Bomble, John F. Stanton, Mihály Kállay, and Jürgen Gauss. Coupled-Cluster Methods Including Noniterative Corrections for Quadruple Excitations. 123(5):054101, 2005.

[154] Mihály Kállay and Jürgen Gauss. Approximate Treatment of Higher Excitations in Coupled-Cluster Theory. 123(21):214105, 2005.

[155] John Stanton. This remark is attributed to Devin Matthews.

[156] J. Fuchs and C. Schweigert. *Symmetries, Lie Algebras and Representations: A Graduate Course for Physicists*. Cambridge University Press, 2003.

[157] F.A. Cotton. *Chemical Applications of Group Theory*. John Wiley & Sons, 2008.

[158] John F. Stanton, Jürgen Gauss, John D. Watts, and Rodney J. Bartlett. A Direct Product Decomposition Approach for Symmetry Exploitation in Many-body Methods, I: Energy Calculations. 94(6):4334–4345, 1991.

[159] Jürgen Gauss, John F. Stanton, and Rodney J. Bartlett. Coupled-Cluster Open-Shell Analytic Gradients: Implementation of the Direct Product Decomposition Approach in Energy Gradient Calculations. 95(4):2623–2638, 1991.

[160] Jaroslaw Nieplocha, Robert J. Harrison, and Richard J. Littlefield. Global Arrays: A Portable "Shared-memory" Programming Model for Distributed Memory Computers. In *Supercomputing '94: Proceedings of the 1994 ACM/IEEE conference on Supercomputing*, pages 340–349, New York, 1994. ACM.

[161] Jaroslaw Nieplocha, Robert J. Harrison, and Richard J. Littlefield. Global Arrays: A Non-Uniform-Memory-Access Programming Model for High-Performance Computers. *The Journal of Supercomputing*, 10:10–197, 1996.

[162] Jarek Nieplocha and Bryan Carpenter. ARMCI: A Portable Remote Memory Copy Library for Distributed Array Libraries and Compiler Run-time Systems. In *Parallel and Distributed Processing*, pages 533–546, London, UK, 1999. Springer-Verlag.

[163] James Dinan, D. Brian Larkins, P. Sadayappan, Sriram Krishnamoorthy, and Jarek Nieplocha. Scalable Work Stealing. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 53:1–53:11, New York, 2009. ACM.

[164] Humayun Arafat, P. Sadayappan, James Dinan, Sriram Krishnamoorthy, and Theresa L. Windus. Load Balancing of Dynamical Nucleation Theory Monte Carlo Simulations through Resource Sharing Barriers. In *IPDPS*, pages 285–295, 2012.

[165] So Hirata. Tensor Contraction Engine: Abstraction and Automated Parallel Implementation of Configuration-Interaction, Coupled-Cluster, and Many-Body Perturbation Theories. 107:9887–9897, 2003.

[166] Alexander A. Auer, Gerald Baumgartner, David E. Bernholdt, Alina Bibireata, Venkatesh Choppella, Daniel Cociorva, Xiaoyang Gao, Robert Harrison, Sriram Krishnamoorthy, Sandhya Krishnan, Chi-Chung Lam, Qingda Lu, Marcel Nooijen, Russell Pitzer, J. Ramanujam, P. Sadayappan, and Alexander Sibiryakov. Automatic Code Generation for Many-body Electronic Structure Methods: The Tensor Contraction Engine. 104(2):211–228, 2006.

[167] Karol Kowalski, Jeff R. Hammond, Wibe A. de Jong, Peng-Dong Fan, Marat Valiev, Dunyou Wang, and Niranjan Govind. Coupled Cluster Calculations for Large Molecular and Extended Systems. In Jeffrey R. Reimers, editor, *Computational Methods for Large Systems: Electronic Structure Approaches for Biotechnology and Nanotechnology*. Wiley, 2011.

[168] Sameer S. Shende and Allen D. Malony. The TAU Parallel Performance System. *Int. J. High Perform. Comput. Appl.*, 20(2):287–311, May 2006.

[169] Gagan Agrawal, Alan Sussman, and Joel Saltz. An Integrated Runtime and Compile-Time Approach for Parallelizing Structured and Block Structured Applications. *IEEE Transactions on Parallel and Distributed Systems*, 6:747–754, 1995.

[170] Stephen J. Fink, Scott B. Baden, and Scott R. Kohn. Efficient Run-Time Support for Irregular Block-Structured Applications. *Journal of Parallel and Distributed Computing*, 50(1–2):61–82, 1998.

[171] Stephen Fink and Scott Baden. Runtime Support for Multi-tier Programming of Block-structured Applications on SMP Clusters. In Yutaka Ishikawa, Rodney Oldehoeft, John Reynders, and Marydell Tholburn, editors, *Scientific Computing in Object-Oriented Parallel Environments*, volume 1343 of *Lecture Notes in Computer Science*, pages 1–8. Springer Berlin / Heidelberg, 1997.

[172] Elmar Peise and Paolo Bientinesi. Performance Modeling for Dense Linear Algebra. In *Proceedings of the 3rd International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS12)*, November 2012.

[173] S. H. Bokhari. On the Mapping Problem. *IEEE Trans. Comput.*, 30(3):207–214, March 1981.

[174] Karen Devine, Erik Boman, Robert Heaphy, Bruce Hendrickson, and Courtenay Vaughan. Zoltan Data Management Services for Parallel Dynamic Applications. *Computing in Science and Engineering*, 4(2):90–97, 2002.

[175] Sriram Krishnamoorthy, Ümit V. Çatalyürek, Jarek Nieplocha, and Atanas Rountev. Hypergraph Partitioning for Automatic Memory Hierarchy Management. In *Supercomputing (SC06)*, 2006.

[176] R. L. Graham. Bounds on Multiprocessing Timing Anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.

[177] Jeff R. Hammond, Sriram Krishnamoorthy, Sameer Shende, Nichols A. Romero, and Allen D. Malony. Performance Characterization of Global Address Space Applications: A Case Study with NWChem. 24(2):135–154, 2012.

[178] Yuri Alexeev, Ashutosh Mahajan, Sven Leyffer, Graham Fletcher, and Dmitri Fedorov. Heuristic Static Load-Balancing Algorithm Applied to the Fragment Molecular Orbital Method. *Supercomputing*, 2012.

[179] Edgar Solomonik. Cyclops Tensor Framework. http://www.eecs.berkeley.edu/ solomon/cyclopstf/index.html.

[180] R. A. Van De Geijn and J. Watts. SUMMA: Scalable Universal Matrix Multiplication Algorithm. *Concurrency: Practice and Experience*, 9(4):255–274, 1997.

[181] W. Richard Stevens. *UNIX Network Programming, Volume 2 (2nd ed.): Interprocess Communications*. Prentice Hall PTR, Upper Saddle River, NJ, 1999.

[182] T. Hoefler, J. Dinan, D. Buntinas, P. Balaji, B. Barrett, R. Brightwell, W. Gropp, V. Kale, and R. Thakur. MPI + MPI: A New Hybrid Approach to Parallel Programming with MPI plus Shared Memory. *Computing*, 95(12):1121–1136, 2013.

[183] D. Ozog. TCE mini-app source code repository. `https://github.com/davidozog/NWChem-mini-app`.

[184] J. Nieplocha, R.J. Harrison, and R.J. Littlefield. Global Arrays: A Nonuniform Memory Access Programming Model for High-Performance Computers. *The Journal of Supercomputing*, 10(2):169–189, 1996.

[185] E. Aprà, A.P. Rendell, R.J. Harrison, V. Tipparaju, W.A. deJong, and S.S. Xantheas. Liquid Water: Obtaining the Right Answer for the Right Reasons. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 66:1–66:7, New York, NY, USA, 2009. ACM.

[186] William Gropp and Ewing Lusk. Reproducible Measurements of MPI Performance Characteristics. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, volume 1697 of *Lecture Notes in Computer Science*, pages 11–18. Springer Berlin Heidelberg, 1999.

[187] Laxmikant V. Kale and Gengbin Zheng. *Charm++ and AMPI: Adaptive Runtime Strategies via Migratable Objects*, pages 265–282. John Wiley & Sons, Inc., 2009.

[188] A. Tabbal, M. Anderson, M. Brodowicz, H. Kaiser, and T. Sterling. Preliminary Design Examination of the Parallex System from a Software and Hardware Perspective. *SIGMETRICS Perform. Eval. Rev.*, 38(4):81–87, March 2011.

[189] A. Nikodem, A. V. Matveev, T. M. Soini, and N. Rösch. *International Journal of Quantum Chemistry*, 114(12):813–822, 2014.

[190] Alex Kogan and Erez Petrank. Wait-free Queues with Multiple Enqueuers and Dequeuers. In *Proceedings of the 16th ACM Symposium on Principles and Practice of Parallel Programming*, PPoPP '11, pages 223–234, New York, 2011. ACM.

[191] Yili Zheng, Amir Kamil, Michael B. Driscoll, Hongzhang Shan, and Katherine Yelick. UPC++: A PGAS Extension for C++. In *28th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2014.

[192] S. French, Yili Zheng, B. Romanowicz, and K. Yelick. Parallel Hessian Assembly for Seismic Waveform Inversion Using Global Updates. In *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*, pages 753–762, May 2015.

[193] Amir Kamil, Yili Zheng, and Katherine Yelick. A Local-View Array Library for Partitioned Global Address Space C++ Programs. In *Proceedings of ACM SIGPLAN International Workshop on Libraries, Languages, and Compilers for Array Programming*, ARRAY'14, pages 26:26–26:31, New York, NY, USA, 2014. ACM.

[194] GASNet home page. `http://gasnet.lbl.gov`.

[195] Min Si, A.J. Peña, J. Hammond, P. Balaji, M. Takagi, and Y. Ishikawa. Casper: An Asynchronous Progress Model for MPI RMA on Many-Core Architectures. In *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*, pages 665–676, May 2015.

[196] Norbert Flocke and Victor Lotrich. Efficient Electronic Integrals and Their Generalized Derivatives for Object Oriented Implementations of Electronic Structure Calculations. *Journal of computational chemistry*, 29(16):2722–2736, 2008.

[197] J. Milthorpe, V. Ganesh, A.P. Rendell, and D. Grove. X10 as a Parallel Language for Scientific Computation: Practice and Experience. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1080–1088, May 2011.

[198] J. Dinan, P. Balaji, J.R. Hammond, S. Krishnamoorthy, and V. Tipparaju. Supporting the Global Arrays PGAS Model Using MPI One-Sided Communication. In *Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pages 739–750, May 2012.

[199] William Gropp, Torsten Hoefler, Rajeev Thakur, and Ewing Lusk. *Using Advanced MPI: Modern Features of the Message-Passing Interface*. MIT Press, 2014.

[200] Michael Rubinstein and Ralph H. Colby. *Polymers Physics*. Oxford, 2003.

[201] Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, and J. Myers. Examining the Challenges of Scientific Workflows. *IEEE Computer*, 40(12):26–34, December 2007.

[202] K.S. Schweizer and J.G. Curro. PRISM Theory of the structure, Thermodynamics, and Phase Transitions of Polymer Liquids and Alloys. 116:319–377, 1994.

[203] E. David and K. Schweizer. Integral Equation Theory of Block Copolymer Liquids. II. Numerical Results for Finite Hard-Core Diameter Chains. *The Journal of Chemical Physics*, 100:7784, 1994.

[204] I. Y. Lyubimov, J. McCarty, A. Clark, and M. G. Guenza. Analytical Rescaling of Polymer Dynamics from Mesoscale Simulations. 132(22):224903, June 2010.

[205] C. N. Likos, A. Lang, M. Watzlawek, and H. Löwen. Criterion for Determining Clustering Versus Reentrant Melting Behavior for Bounded Interaction Potentials. *Phys. Rev. E*, 63, Feb 2001.

[206] Dragos-Anton Manolescu, Markus Voelter, and James Noble. *Pattern Languages of Program Design 5*, volume 5. Addison-Wesley Professional, 2006.

[207] Terence John Parr. Enforcing Strict Model-View Separation in Template Engines. In *Proceedings of the 13th International Conference on World Wide Web*, WWW '04, pages 224–233. ACM, 2004.

[208] K. Anderson, R. Mukherjee, J. Critchley, J. Ziegler, and S. Lipton. POEMS: Parallelizable Open-source Efficient Multibody Software. *Engineering with Computers*, 23(1):11–23, 2007.

[209] D.A. McQuarrie. *Statistical Mechanics*. University Science Books, 2000.

[210] R. Auhl, R. Everaers, G. Grest, K. Kremer, and S. Plimpton. Equilibration of Long Chain Polymer Melts in Computer Simulations. *The Journal of Chemical Physics*, 119(24):12718–12728, 2003.

[211] Pek U. Ieong, Jesper Srensen, Prasantha L. Vemu, Celia W. Wong, zlem Demir, Nadya P. Williams, Jianwu Wang, Daniel Crawl, Robert V. Swift, Robert D. Malmstrom, Ilkay Altintas, and Rommie E. Amaro. Progress Towards Automated Kepler Scientific Workflows for Computer-aided Drug Discovery and Molecular Simulations. *Procedia Computer Science*, 29:1745 – 1755, 2014. 2014 International Conference on Computational Science.

[212] Jianwu Wang, Daniel Crawl, and Ilkay Altintas. A Framework for Distributed Data-Parallel Execution in the Kepler Scientific Workflow System. *2012 International Conference on Computational Science (ICCS 2012)*, 9(0):1620 – 1629, 2012.

[213] Ewa Deelman, Gurmeet Singh, Mei-Hui Su, et al. Pegasus: A Framework for Mapping Complex Scientific Workflows Onto Distributed Systems. *Scientific Programming*, 13(3):219–237, July 2005.

[214] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, et al. Taverna: a Tool for the Composition and Enactment of Bioinformatics Workflows. *Bioinformatics*, 20:3045–3054, 2004.

[215] J.P. Hansen and I. R. McDonald. *Theory of Simple Liquids*. Academic Press, London, 1991.

[216] Kenneth S Schweizer and John G Curro. Integral Equation Theories of the Structure, Thermodynamics, and Phase Transitions of Polymer Fluids. *Advances in Chemical Physics, Volume 98*, pages 1–142, 1997.

[217] Pierre-Gilles De Gennes. *Scaling Concepts in Polymer Physics*. Cornell university press, 1979.

[218] Fast-Equilibration-Workflow source code repository, 2015.

[219] Praveen Depa, Chunxia Chen, and Janna K. Maranas. Why are Coarse-Grained Force Fields Too Fast? A Look at Dynamics of Four Coarse-Grained Polymers. *The Journal of Chemical Physics*, 134(1), 2011.

[220] Moritz Winger, Daniel Trzesniak, Riccardo Baron, and Wilfred F. van Gunsteren. On Using a Too Large Integration Time Step in Molecular Dynamics Simulations of Coarse-Grained Molecular Models. *Phys. Chem. Chem. Phys.*, 11:1934–1941, 2009.

[221] M. Guenza. Thermodynamic Consistency and Other Challenges in Coarse-Graining Models. *The European Physical Journal Special Topics*, 224(12):2177–2191, 2015.

[222] A A Louis. Beware of Density Dependent Pair Potentials. *Journal of Physics: Condensed Matter*, 14(40):9187, 2002.

[223] Margaret E. Johnson, Teresa Head-Gordon, and Ard A. Louis. Representability Problems for Coarse-Grained Water Potentials. *The Journal of Chemical Physics*, 126(14), 2007.

[224] Phillip J. Stansfeld and Mark S.P. Sansom. From Coarse Grained to Atomistic: A Serial Multiscale Approach to Membrane Protein Simulations. *Journal of Chemical Theory and Computation*, 7(4):1157–1166, 2011.

[225] Andrzej J. Rzepiela, Lars V. Schäfer, Nicolae Goga, H. Jelger Risselada, Alex H. De Vries, and Siewert J. Marrink. Reconstruction of Atomistic Details from Coarse-Grained Structures. *Journal of Computational Chemistry*, 31(6):1333–1343, 2010.

[226] Xiaoyu Chen, Paola Carbone, Giuseppe Santangelo, Andrea Di Matteo, Giuseppe Milano, and Florian Muller-Plathe. Backmapping Coarse-Grained Polymer Models under Sheared Nonequilibrium Conditions. *Phys. Chem. Chem. Phys.*, 11:1977–1988, 2009.

[227] Azadeh Ghanbari, Michael C. Böhm, and Florian Müller-Plathe. A Simple Reverse Mapping Procedure for Coarse-Grained Polymer Models with Rigid Side Groups. *Macromolecules*, 44(13):5520–5526, 2011.

[228] Hideaki Takahashi and Nobuyuki Matubayasi. Development of a Massively Parallel QM/MM Approach Combined with a Theory of Solutions. In Jean-Louis Rivail, Manuel Ruiz-Lopez, and Xavier Assfeld, editors, *Quantum Modeling of Complex Molecular Systems*, volume 21 of *Challenges and Advances in Computational Chemistry and Physics*, pages 153–196. Springer International Publishing, 2015.

[229] Walter RP Scott, Philippe H Hünenberger, Ilario G Tironi, Alan E Mark, Salomon R Billeter, Jens Fennen, Andrew E Torda, Thomas Huber, Peter Krüger, and Wilfred F van Gunsteren. The GROMOS Biomolecular Simulation Program Package. *The Journal of Physical Chemistry A*, 103(19):3596–3607, 1999.

[230] Steve Plimpton, Paul Crozier, and Aidan Thompson. LAMMPS-Large-Scale Atomic/Molecular Massively Parallel Simulator. *Sandia National Laboratories*, 18, 2007.

[231] Jürg Hutter, Marcella Iannuzzi, Florian Schiffmann, and Joost VandeVondele. CP2K: Atomistic Simulations of Condensed Matter Systems. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 4(1):15–25, 2014.

[232] E. J. Bylaska, W. A. de Jong, N. Govind, K. Kowalski, T. P. Straatsma, M. Valiev, D. Wang, E. Aprà, T. L. Windus, J. Hammond, J. Autschbach, P. Nichols, S. Hirata, M. T. Hackler, Y. Zhao, P-D Fan, R. J. Harrison, M. Dupuis, D. M. A. Smith, J. Nieplocha, V. Tipparaju, M. Krishnan, A. Vazquez-Mayagoitia, Q. Wu, T. Van Voorhis, A. A. Auer, M. Nooijen, L. D. Crosby, E. Brown, G. Cisneros, G. I. Fann, H. Früchtl, J. Garza, K. Hirao, R. Kendall, J. A. Nichols, K. Tsemekhman, K. Wolinski, J. Anchell, D. Bernholdt, P. Borowski, T. Clark, D. Clerc, H. Dachsel, M. Deegan, K. Dyall, D. Elwood, E. Glendening, M .Gutowski, A Hess, J. Jaffe, B. Johnson, J. Ju, R. Kobayashi, R. Kutteh, Z. Lin, R. Littlefield, X. Long, B. Meng, T. Nakajima, S. Niu, L. Pollack, M. Rosing, G. Sandrone, M. Stave, H. Taylor, G. Thomas, J. van Lenthe, A. Wong, and Z. Zhang. NWChem, A Computational Chemistry Package for Parallel Computers, Version 5.1.1, 2009.

[233] P Chow and C Addison. An Overview on Current Multiphysics Software Strategies for Coupled Applications with Interacting Physics on Parallel and Distributed Computers. In *Proceedings of the 11th International Conference on Domain Decomposition Methods in Sciences & Engineering*, 1999.

[234] Jay F Lofstead, Scott Klasky, Karsten Schwan, Norbert Podhorszki, and Chen Jin. Flexible IO and Integration for Scientific Codes through the Adaptable IO System (ADIOS). In *Proceedings of the 6th international workshop on Challenges of large applications in distributed environments*, pages 15–24. ACM, 2008.

[235] Changru Ma, Layla Martin-Samos, Stefano Fabris, Alessandro Laio, and Simone Piccinin. QMMMW: A Wrapper for QM/MM Simulations with Quantum ESPRESSO and LAMMPS. *Computer Physics Communications*, 195:191 – 198, 2015.

[236] Sushmitha P Kini, Jiuxing Liu, Jiesheng Wu, Pete Wyckoff, and Dhabaleswar K Panda. Fast and Scalable Barrier Using RDMA and Multicast Mechanisms for InfiniBand-based Clusters. In *European Parallel Virtual Machine/Message Passing Interface Users Group Meeting*, pages 369–378. Springer, 2003.

[237] T. Hoefler, T. Mehlan, F. Mietke, and W. Rehm. Fast Barrier Synchronization for InfiniBand. In *Proceedings 20th IEEE International Parallel Distributed Processing Symposium*, page pp.7, April 2006.