OPTIMIZING AND IMPROVING THE FIDELITY OF REACTIVE, POLARIZABLE MOLECULAR DYNAMICS SIMULATIONS ON MODERN HIGH PERFORMANCE COMPUTING ARCHITECTURES

By

Kurt A. O'Hearn

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Computer Science – Doctor of Philosophy

2022

ABSTRACT

OPTIMIZING AND IMPROVING THE FIDELITY OF REACTIVE, POLARIZABLE MOLECULAR DYNAMICS SIMULATIONS ON MODERN HIGH PERFORMANCE COMPUTING ARCHITECTURES

By

Kurt A. O'Hearn

Reactive, polarizable molecular dynamics simulations are a crucial tool for the high-fidelity study of large systems with chemical reactions. In support of this, several approaches have been employed with varying degrees of computational cost and physical accuracy. One of the more successful approaches in recent years, the reactive force field (ReaxFF) model, was designed to fill the gap between traditional classical models and quantum mechanical models by incorporating a dynamic bond order potential term. When coupling ReaxFF with dynamic global charges models for electrostatics, special considerations are necessary for obtaining highly performant implementations, especially on modern high-performance computing architectures.

In this work, we detail the performance optimization of the PuReMD (PuReMD Reactive Molecular Dynamics) software package, an open-source, GPLv3-licensed implementation of ReaxFF coupled with dynamic charge models. We begin by exploring the tuning of the iterative Krylov linear solvers underpinning the global charge models in a shared-memory parallel context using OpenMP, with the explicit goal of minimizing the mean combined preconditioner and solver time. We found that with appropriate solver tuning, significant speedups and scalability improvements were observed. Following these successes, we extend these approaches to the solvers in the distributed-memory MPI implementation of PuReMD, as well as broaden the scope of optimization to other portions of the ReaxFF potential such as the bond order computations. Here again we find that sizable performance gains were achieved for large simulations numbering in the hundreds of thousands of atoms.

With these performance improvements in hand, we next change focus to another important use of PuReMD – the development of ReaxFF force fields for new materials. The high fidelity inherent in ReaxFF simulations for different chemistries oftentimes comes at the expense of a steep learning curve for parameter optimization, due in part to complexities in the high dimensional parameter space and due in part to the necessity of deep domain knowledge of how to adequately control the ReaxFF functional forms. To diagnose and combat these issues, a study was undertaken to optimize parameters for Li-O systems using the OGOLEM genetic algorithms framework coupled with a modified shared-memory version of PuReMD. We found that with careful training set design, sufficient optimization control with tuned genetic algorithms, and improved polarizability through enhanced charge model use, higher accuracy was achieved in simulations involving ductile fracture behavior, a difficult phenomena to hereto model correctly.

Finally, we return to performance optimization for the GPU-accelerated distributed-memory PuReMD codebase. Modern supercomputers have recently achieved exascale levels of peak arithmetic rates due in large part to the design decision to incorporate massive numbers of GPUs. In order to take advantage of such computing systems, the MPI+CUDA version of PuReMD was re-designed and benchmarked on modern NVIDIA Tesla GPUs. Performance on-par with or exceeding the LAMMPS Kokkos, a ReaxFF implementation developed at Scandia National Laboratories, with PuReMD typically out-performing LAMMPS Kokkos at larger scales.

This thesis is dedicated to my family for their unconditional support and investment in my educational aspirations.

ACKNOWLEDGEMENTS

The work in Chapter 2 was supported through an MSU Foundation Strategic Partnership Grant, and NSF Grants ACI-1566049 and OAC-1807622. This research used computational resources provided by the Institute for Cyber-Enabled Research at Michigan State University.

The work in Chapter 3 was supported through a Michigan State University Foundation Strategic Partnership Grant, NSF Grants ACI-1566049 and OAC-1807622, and the computing resources at the National Energy Research Scientific Computing Center (NERSC).

The work in Chapter 4 was supported in part by a Strategic Partnership Grant from the Michigan State University Foundation, an NSF CDS&E grant (award number 1807622), an NSF DMR grant (award number 1832808), and an U.S. DOE grant number DE-FG02-01ER15228. This work used computational resources provided by the Institute for Cyber-Enabled Research at Michigan State University. Thanks to Stan Moore, Jason Koski, and the LAMMPS group for sharing

The work in Chapter 5 used computational resources provided by the Institute for Cyber-Enabled Research at Michigan State University.

TABLE OF CONTENTS

LIST OF TABLES x

LIST OF FIGURES xi

LIST OF ALGORITHMS xiii

CHAPTER 1 BACKGROUND AND RELATED MATERIAL 1
1.1 Description of the Reactive Force Field Method 1
1.1.1 Bond Orders 2
1.1.2 Bonding Term 3
1.1.3 Lone Pair Term 3
1.1.4 Over- and Under-coordination Term 4
1.1.5 Valence Angle Term 4
1.1.6 Torsion Term 5
1.1.7 Additional Bonding Terms 5
1.1.8 Hydrogen Bonding Term 6
1.1.9 van der Waal's Term 6
1.1.10 Coulomb Term 7

CHAPTER 2 FAST SOLVERS FOR CHARGE DISTRIBUTION MODELS ON SHARED-MEMORY PLATFORMS 8
2.1 Introduction 9
2.2 Dynamic Charge Distribution Models and Linear Solvers 10
2.2.1 Charge Equilibriation (QEq) 10
2.2.2 Electronegativity Equalization (EE) 11
2.2.3 Atom-Condensed Kohn-Sham Approximated to Second Order (ACKS2) 12
2.3 Preconditioning Techniques for Linear Solvers 13
2.3.1 Approaches 14
2.3.1.1 Jacobi (**Jacobi**) 15
2.3.1.2 Incomplete LU (ILU) based Techniques 15
2.3.1.3 Sparse Approximate Inverse (**SAI(τ)**)) 19
2.3.2 Dynamic Determination of Preconditioner Recomputation 20
2.3.3 Solver Implementation 23
2.4 Numerical Experiments 24
2.4.1 Computing Environment and Benchmark Systems 24
2.4.2 Preconditioner Longevity 26
2.4.3 Tuning of Preconditioner Parameters for Cost and Effectiveness . . . 27
2.4.4 Scalability of Preconditioned Solvers 31
2.4.5 Preconditioned Solver Performance 33
2.5 Conclusions 35

CHAPTER 3 PERFORMANCE OPTIMIZATION OF REACTIVE MOLECULAR DYNAMICS SIMULATIONS WITH DYNAMIC CHARGE DISTRIBUTION MODELS ON DISTRIBUTED MEMORY PLATFORMS . 37
3.1 Introduction . . . 37
3.2 Background . . . 39
3.2.1 Parallelization . . . 40
3.2.2 Neighbor Generation . . . 41
3.2.3 Initialization of Interaction Lists . . . 41
3.2.4 Bonded Interactions . . . 42
3.2.5 Charge Distribution . . . 42
3.2.6 Nonbonded Interactions . . . 44
3.3 Methods . . . 44
3.3.1 Reduction of Global Communication Overheads in Charge Solvers . . . 44
3.3.1.1 Baseline Preconditioned Conjugate Gradient . . . 44
3.3.1.2 Preconditioned Pipelined Conjugate Gradient . . . 45
3.3.2 Acceleration of QEq through preconditioning . . . 47
3.3.2.1 SAI Preconditioning . . . 47
3.3.2.2 Parallelization of the SAI Preconditioner . . . 49
3.3.2.3 Structure of the Hamiltonian . . . 50
3.3.2.4 SAI Preconditioner Sparsity Pattern Selection . . . 50
3.3.2.5 SAI Preconditioner Computation . . . 51
3.3.3 Optimization at the Ghost Regions . . . 54
3.4 Numerical Results . . . 56
3.4.1 Benchmarking Systems and Hardware . . . 56
3.4.2 Impact of Charge Solver Improvements . . . 57
3.4.2.1 SAI Longevity . . . 57
3.4.2.2 SAI Preconditioning Convergence Rates . . . 58
3.4.2.3 Strong Scaling Tests . . . 58
3.4.2.4 Weak Scaling . . . 61
3.4.3 Impact of Ghost Region Optimizations . . . 63
3.4.4 Overall Simulation Performance . . . 63
3.5 Conclusions . . . 64

CHAPTER 4 OPTIMIZATION OF THE REAX FORCE FIELD FOR THE LITHIUM-OXYGEN SYSTEM USING A HIGH FIDELITY CHARGE MODEL 67
4.1 Introduction . . . 67
4.2 Methodology . . . 71
4.2.1 Training Data Set . . . 71
4.2.2 High Fidelity Charge Model . . . 72
4.2.3 Optimization of the Force Field Parameters . . . 75
4.3 Results and Discussion . . . 77
4.3.1 Fitness to the Training Data Set . . . 78
4.3.2 Evaluation using Bulk Lithium Oxide . . . 82
4.3.3 Evaluation using Deformation and Fracture . . . 83
4.4 Conclusion . . . 88

CHAPTER 5 PERFORMANCE OPTIMIZATION OF LARGE-SCALE DISTRIBUTED GPU-ACCELERATED REAXFF SIMULATIONS 90
5.1 Introduction 90
5.2 Methods 90
5.2.1 Improving Performance Portability and Scalability via Software Modernization 90
5.2.2 Increasing Parallelism through Kernel-Level Restructuring 92
5.2.3 Exploiting Task-Level Parallelism via CUDA Streams 93
5.2.4 Global Charge Solver Optimizations 95
5.3 Performance Studies 96
5.3.1 Computing Environment and Benchmark Systems 96
5.3.2 Scalability Studies 97
5.4 Conclusions 98

APPENDICES 102
APPENDIX A SUPPLEMENTAL DATA FOR GLOBAL CHARGE MODEL OPTIMIZATIONS IN SHARED-MEMORY 103
APPENDIX B SUPPLEMENTAL DATA FOR OPTIMIZATION OF THE REAX FORCE FIELD FOR THE LITHIUM-OXYGEN SYSTEM USING A HIGH FIDELITY CHARGE MODEL 105

BIBLIOGRAPHY 114

LIST OF TABLES

Table 2.1 Molecular systems used in shared-memory performance evaluation. 25

Table 2.2 Qualitative behavior of the preconditioning techniques. 25

Table 2.3 Number of recomputations for the shared-memory preconditioned solvers. 28

Table 2.4 Shared-memory preconditioner parameter comparison using the QEq charge method. 29

Table 2.5 Comparison of the cost and effectiveness of the shared-memory preconditioned solvers for the bulk water system. 30

Table 3.1 Mean solver iterations for the Jacobi and SAI distributed preconditioned QEq solvers. 59

Table 3.2 Average number of bond order calculations per simulation step. 63

Table 4.1 Materials properties as predicted by DFT, the fitted ACKS2 FF, and the 2016 QEq FF. 81

Table 4.2 Properties of the notched slabs used as a starting point for the fracture simulations. 85

Table A.1 Force field parameter values utlized for simulations with the molecular systems in Table 2.1. 103

Table A.2 Charge matrix condition numbers for the charge distribution models. . . . 103

Table A.3 Tuned preconditioned solver parameters for shared-memory experiments. . 104

Table B.1 The ReaxFF parameters adjusted to create the new ACKS2-based force field for Li-O. 105

Table B.2 Nuclear geometries used in the calculations for the Li-O-Li system, along with the MRCI+Q total electronic energies and MRCI atomic charges on Li and O atoms. 107

LIST OF FIGURES

Figure 2.1 Longevity of preconditioners with the QEq model. 27

Figure 2.2 Strong scaling plots of shared-memory preconditioned solvers with QEq. 32

Figure 2.3 Speedups for shared-memory solvers for the benchmark systems by charge distribution method. 34

Figure 3.1 ReaxFF's computational workflow includes bonded, non-bonded, and system-specific interactions, each with different physical formulations and cut-offs. 40

Figure 3.2 Structure of the local charge matrix in distributed-memory simulations. . 51

Figure 3.3 The global view of the process of constructing and solving one of the least-squares problems for the distributed SAI preconditioning approach with an example QEq matrix. 53

Figure 3.4 Longevity of the **CG+SAI(0.15)** distributed preconditioned solver for systems using the QEq model. 57

Figure 3.5 Strong scaling plots for mean QEq time of the distributed preconditioned solvers for the silica and water systems. 60

Figure 3.6 Speedups and parallel efficiencies for the distributed strong scaling study. 61

Figure 3.7 Weak scaling plots for the distributed solver sub-kernels for QEq. 62

Figure 3.8 Strong scaling plots for total simulation time of distributed preconditioned solvers for the benchmark systems. 64

Figure 3.9 Weak scaling plots for main simulation sub-kernels in distributed memory using the benchmark systems. 65

Figure 4.1 Fitness of the best performing ReaxFF parameter set during optimization. 77

Figure 4.2 Charge during dissociation of the Li−O−Li molecule for asymmetric symmetric bond stretching. 79

Figure 4.3 Energy vs. hydrostatic strain of Li_2O and Li_2O_2 crystals along with energy vs. separation of the Li_2O molecule for bond stretching. 80

Figure 4.4 NPT simulation and material properties of crystalline Li_2O with the ACKS2 FF and the 2016 QEq FF. 84

Figure 4.5 Simulation results and material properties for Li_2O slabs before and after NPT equilibration with ACKS2 and 2016QEq FF. 85

Figure 4.6 Simulation results for fracture in Li_2O slabs. 86

Figure 5.1 Task dependency graph for the ReaxFF potential coupled with a global charge model (QEq) within a single MD step. 94

Figure 5.2 Execution profile of the MPI+CUDA version of PuReMD using a single CUDA stream. 96

Figure 5.3 Weak scaling study for the MPI+CUDA PuReMD implementation and LAMMPS Kokkos. 100

Figure 5.4 Strong scaling study for the MPI+CUDA PuReMD implementation and LAMMPS Kokkos. 101

Figure B.1 Deformation simulation using the fitted QEq force field for Li-O. 110

Figure B.2 Detailed information on oxygen charge during cracking simulation. 111

Figure B.3 Second cracking experiment with identical parameters and starting structures. 112

Figure B.4 Evolution of the RDF during NPT simulation with the ACKS2 FF and the 2016 QEq FF. 113

LIST OF ALGORITHMS

Algorithm 2.1 Dynamic Determination of Preconditioner Recomputation. 21

Algorithm 3.1 Preconditioned Pipelined Conjugate Gradient. 46

Algorithm 3.2 Distributed SAI(τ) Preconditioner Sparsity Pattern Selection. 52

Algorithm 3.3 Distributed SAI(τ) Preconditioner Computation. 54

Algorithm 5.1 Atomic Transactional Approach for GPU Memory Management. . . . 92

CHAPTER 1

BACKGROUND AND RELATED MATERIAL

1.1 Description of the Reactive Force Field Method

The reactive force field (ReaxFF) method is a relatively recent model (developed in early 2000s [1]) and is similar to the classical molecular dynamics (MD) model in the sense that it models atomic nuclei together with their electrons as a point mass. Unlike classical MD models, ReaxFF mimics bond formation and breakage observed in quantum mechanics (QM) methods by replacing the static harmonic bond models with the bond order concept, which is a quantity indicating bond strength between a pair of atoms based on the types of the atoms and the distance between them. Consequently, ReaxFF can overcome many of the limitations inherent to conventional MD. While the bond order concept dates back to 1980s and has been exploited in other force fields before (such as COMB [2] and AIREBO [3]), the distinguishing aspect of ReaxFF is the flexibility and transferability of its force field that allows ReaxFF to be applied to diverse systems of interest [1, 4, 5, 6].

ReaxFF is currently implemented by major open source (PuReMD Reactive Molecular Dynamics (PuReMD) [7], large-scale atomic/molecular massively parallel simulator (LAMMPS) [8], RXMD [9]) and commercial (ADF, Material Studio) software with an estimated userbase of over 1,000 groups. In this thesis, the work builds off and utilizes the PuReMD software, as PuReMD and its LAMMPS integrations, i.e., the User-ReaxC and User-ReaxC/OMP packages [8], represent the most widely used implementations of ReaxFF. PuReMD uses novel algorithms and data structures to achieve high performance while retaining a small memory footprint. An optimized neighbor generation scheme, elimination of the bond order derivatives list in bonded interactions, lookup tables to accelerate non-bonded interaction computations, and efficient iterative solvers for charge distribution are the major algorithmic innovations in PuReMD [10, 11].

In ReaxFF, the total energy of the system is comprised of partial energy contributions according to Eq. (1.1), where summation over atomic indices is implied for each term.

$$\begin{aligned} E_{\text{system}} = {} & E_{\text{bond}} + E_{\text{lp}} + E_{\text{over}} + E_{\text{under}} \\ & + E_{\text{val}} + E_{\text{pen}} + E_{\text{3conj}} \\ & + E_{\text{tors}} + E_{\text{4conj}} + E_{\text{H-bond}} + E_{\text{vdW}} + E_{\text{Coulomb}} \end{aligned} \tag{1.1}$$

While there are many similarities in ReaxFF to classical MD methods, one major difference is that bond orders are calculated system-wide at the beginning of every MD step before computing energies and gradients. Moreover, due to the dynamic bonding scheme of ReaxFF, these potentials must be modified to ensure smooth potential energy curves as bonds form or break. In the following subsections, these details are elucidated.

1.1.1 Bond Orders

In ReaxFF, the bond order between a pair of atoms i and j signifies the strength of the bond between the two atoms. Eq. (1.2) properly quantifies this notion for specific types of atoms i and j at at distance apart of r_{ij}.

$$\text{BO}_{ij}^{\alpha'}(r_{ij}) = \exp\left[a_\alpha \left(\frac{r_{ij}}{r_{0\alpha}}\right)^{b_\alpha}\right] \tag{1.2}$$

In the above equation, α corresponds to $\sigma-\sigma$, $\sigma-\pi$, or $\pi-\pi$ bonds; a_α and b_α are parameters specific to the bond type; and $r_{0\alpha}$ is the optimal length for this bond type. With this in mind, the total bond order BO'_{ij} is expressed as the summation of these bond types in the equation below.

$$\text{BO}'_{ij} = \text{BO}_{ij}^{\sigma'} + \text{BO}_{ij}^{\pi'} + \text{BO}_{ij}^{\pi\pi'} \tag{1.3}$$

In order to model complex bonding behavior in real-life systems, additional considerations beyond pairwise bond orders are necessary including the total coordination number of each atom and 1–3 bond corrections in valence angles. Eq. (1.4) describes these such corrections.

$$\text{BO}_{ij} = \text{BO}'_{ij} \cdot f_1\left(\Delta'_i, \Delta'_j\right) \cdot f_4\left(\Delta'_i, \text{BO}'_{ij}\right) \cdot f_5\left(\Delta'_j, \text{BO}'_{ij}\right) \tag{1.4}$$

In the above equation, Δ'_i denotes the deviation of atom i from its optimal coordination number, $f_1\left(\Delta'_i, \Delta'_j\right)$ applies an over-coordination correction, and $f_4(\Delta'_i, \mathrm{BO}'_{ij})$, together with $f_5(\Delta'_j, \mathrm{BO}'_{ij})$ constitute 1–3 bond order corrections. Only corrected bond orders are used in energy and gradient computations within ReaxFF.

1.1.2 Bonding Term

Classical MD methods adopt a spring model in which the energy of a bond is determined solely by its deviation from the optimal bond distance, thereby ignoring the effects of neighboring bonds. In contrast, ReaxFF tabulates the energy incident on a bond from all bond order constituents. The higher the bond order, the lower the energy and the stronger the force associated with the bond. Eq. (1.5) formalizes this notion, and ensures that the energy and force due to a bond smoothly go to zero as the bond dissipates.

$$\begin{aligned} E_{\mathrm{bond}} = & -D_e^{\sigma} \cdot \mathrm{BO}_{ij}^{\sigma} \cdot \exp\left\{p_{\mathrm{be1}}\left(1-\left(\mathrm{BO}_{ij}^{\sigma}\right)^{p_{\mathrm{be2}}}\right)\right\} \\ & -D_e^{\pi} \cdot \mathrm{BO}_{ij}^{\pi} - D_e^{\pi\pi} \cdot \mathrm{BO}_{ij}^{\pi\pi} \end{aligned} \tag{1.5}$$

1.1.3 Lone Pair Term

Classical MD methods do not need to resolved bond defects due to their aforementioned modeling. In contrast, the bonding in ReaxFF must account for unpaired electrons of an atom via an explicit lone pair term. In a nicely formed and equilibrated system, lone pair energy does not have a significant contribution to the total energy, however, it is important for describing atoms with defective bonds. Lone-pair energy is computed using the following equation:

$$E_{\mathrm{lp}} = \frac{p_{\mathrm{lp2}} \cdot \Delta_i^{\mathrm{lp}}}{1+\exp\{-75 \cdot \Delta_i^{\mathrm{lp}}\}} \tag{1.6}$$

In Eq. (1.6), $\Delta_i^{\mathrm{lp}} = n_{opt}^{\mathrm{lp}} - n_i^{\mathrm{lp}}$ essentially corresponds to the number of unpaired electrons.

1.1.4 Over- and Under-coordination Term

Despite the valence correction applied during bond order corrections, there may still remain some over- or under-coordinated atoms in the system. Over-coordination energy penalizes over-coordinated atoms, while under-coordination energy accounts for the energy due to a resonant π-electron between atomic centers in the presence of a π-bond between atoms i and j. Section 1.1.4 and Section 1.1.4 describe over- and under-coordination, resepctively.

$$E_{\text{over}} = \Delta_i^{\text{lpcorr}} \cdot \frac{\sum_{j \in \text{nbrs}(i)} p_{\text{ovun1}} \cdot D_e^{\sigma} \cdot \text{BO}_{ij}}{\left(\Delta_i^{\text{lpcorr}} + Val_i\right)\left(1 + \exp\{p_{\text{ovun2}} \cdot \Delta_i^{\text{lpcorr}}\}\right)} \tag{1.7}$$

$$E_{\text{under}} = -p_{\text{ovun5}} \cdot f_6(i, p_{\text{ovun7}}, p_{\text{ovun8}}) \cdot \frac{1 - \exp\{p_{\text{ovun6}} \cdot \Delta_i^{\text{lpcorr}}\}}{1 + \exp\{-p_{\text{ovun2}} \cdot \Delta_i^{\text{lpcorr}}\}} \tag{1.8}$$

$$\Delta_i^{\text{lpcorr}} = \Delta_i - \Delta_i^{\text{lp}} \cdot f_6(i, p_{\text{ovun3}}, p_{\text{ovun4}})$$

$$f_6(i, p_1, p_2) = \left(1 + p_1 \cdot \exp\left\{p_2 \cdot \left[\sum_{j \in nbrs(i)} \left(\Delta_j - \Delta_j^{\text{lp}}\right) \cdot \left(\text{BO}_{ij}^{\pi} + \text{BO}_{ij}^{\pi\pi}\right)\right]\right\}\right)^{-1}$$

1.1.5 Valence Angle Term

The energy associated with vibration about the optimum valence angle between a triplet of atoms i, j, and k is computed from according to Section 1.1.5.

$$\begin{aligned} E_{\text{val}} \;=\; & f_7(\text{BO}_{ij}, p_{\text{val3}}, p_{\text{val4}}) \cdot f_7(\text{BO}_{jk}, p_{\text{val3}}, p_{\text{val4}}) \cdot f_8(\Delta_j, p_{\text{val5}}, p_{\text{val6}}, p_{\text{val7}}) \cdot \\ & \left(p_{\text{val1}} - p_{\text{val1}} \cdot \exp\left\{-p_{\text{val2}} \cdot \left(\Theta_0 - \Theta_{ijk}\right)^2\right\}\right) \end{aligned} \tag{1.9}$$

$$f_7(\text{BO}, p_1, p_2) = 1 - \exp\left\{-p_1 \cdot \text{BO}^{p_2}\right\}$$

$$f_8(\Delta, p_1, p_2, p_3) = p_1 - (p_1 - 1) \cdot f_9(\Delta, p_2, p_3)$$

$$f_9(\Delta, p_1, p_2) = \frac{2 + \exp\left\{p_1 \cdot \Delta\right\}}{1 + \exp\left\{p_1 \cdot \Delta\right\} + \exp\left\{-p_2 \cdot \Delta\right\}}$$

Similar to its classical counterparts, the energy on Θ_{ijk} increases as it moves away from its corrected optima Θ_0, which is obtained from the theoretical optima Θ_{00}, by accounting for

the effects of over- and under-coordination on the central atom j as well as the influence of any lone electron pairs. Valence angle energy further depends on the strength of bonds BO_{ij} and BO_{jk}; $f_7(\mathrm{BO}_{ij})$ and $f_7(\mathrm{BO}_{jk})$ terms in Section 1.1.5 ensure that valence angle energy goes smoothly to zero as either bond dissociates.

1.1.6 Torsion Term

Section 1.1.6 accounts for the energy resulting from torsions in a molecule.

$$\begin{aligned} E_{\mathrm{tors}} &= \frac{1}{2} \cdot f_{10}(\mathrm{BO}_{ij}, \mathrm{BO}_{jk}, \mathrm{BO}_{kl}, p_{\mathrm{tor2}}, 1) \cdot \sin(\Theta_{ijk}) \cdot \sin(\Theta_{jkl}) \cdot \\ &\quad [\, V_1 \cdot (1 + \cos(\omega_{ijkl})) + \\ &\quad V_2 \cdot \exp\left\{ p_{\mathrm{tor1}} \cdot \left(2 - \mathrm{BO}_{jk}^{\pi} - f_9(\Delta_j + \Delta_k, p_{\mathrm{tor3}}, p_{\mathrm{tor4}})\right)^2 \right\} \cdot (1 - 2\cos(2\omega_{ijkl})) + \\ &\quad V_3 \cdot (1 + \cos(3\omega_{ijkl}))\,] \end{aligned} \tag{1.10}$$

$$f_{10}(\mathrm{BO}_1, \mathrm{BO}_2, \mathrm{BO}_3, p_1, p_2) = f_7(\mathrm{BO}_1, p_1, p_2) \cdot f_7(\mathrm{BO}_2, p_1, p_2) \cdot f_7(\mathrm{BO}_3, p_1, p_2)$$

As in the valence angle energy term, the torsional conribution from a four-body structure should vanish as any of its bonds dissociate. Here, $f_{10}(\mathrm{BO}_{ij}, \mathrm{BO}_{jk}, \mathrm{BO}_{kl}, p_{\mathrm{tor2}}, 1)$ enforce this constraint. If either of the two valence angles defined by these four atoms approaches π, torsional energy should again disappear; this is accomplished by the term $\sin(\Theta_{ijk}) \cdot \sin(\Theta_{jkl})$.

1.1.7 Additional Bonding Terms

In ReaxFF, there are other bonded interaction terms shown in Eq. (1.1) for which details have been omitted for brevity. The stability of 3-body structures in which the central atom has two double bonds is achieved through a penalty energy term E_{pen}. Three-body conjugation energy E_{3conj} and four-body conjugation energy E_{4conj} terms, as their names imply, encapsulate the energy contribution from conjugated systems. Full details regarding these terms can be found in [1].

1.1.8 Hydrogen Bonding Term

The energy term attached to a hydrogen bond in ReaxFF is given by Eq. (1.11).

$$E_{\text{hbond}} = p_{\text{hb1}} \cdot f_7(\text{BO}_{XH}, p_{\text{hb2}}, 1) \cdot \sin^4\left(\frac{\Theta_{XHZ}}{2}\right) \cdot \exp\left\{-p_{\text{hb3}} \cdot \left(\frac{r_{\text{hb}}^0}{r_{HZ}} + \frac{r_{HZ}}{r_{\text{hb}}^0} - 2\right)\right\} \quad (1.11)$$

In this equation, a bond is defined as existing between an electronegative atom (denoted Z) in the vicinity of a Hydrogen atom covalently bonded to a Nitrogen, Oxygen, or Fluorine atom (denoted by X). Similar to the equations for valency and torsion, the $f_7\left(\text{BO}_{XH}, p_{\text{hb2}}, 1\right)$ term ensures that contributions from hydrogen bonding smoothly approach zero as the covalent bond breaks. For hydrogen bonding to be strong, it is crucial that all three atoms are geometrically aligned on a line. This fact is evident by observing that $\sin^4\left(\frac{\Theta_{XHZ}}{2}\right)$ is maximized when $\Theta_{XHZ} = \pi$.

1.1.9 van der Waal's Term

A distance-corrected Morse-potential term is used for van der Waal's interactions as shown in Section 1.1.9.

$$\begin{aligned} E_{\text{vdWaals}} &= T(r_{ij}) \cdot D_{ij} \cdot \\ &\quad \left[\exp\left\{\alpha_{ij} \cdot \left(1 - \frac{f_{13}(r_{ij})}{r_{\text{vdW}}}\right)\right\} - 2 \cdot \exp\left\{\frac{1}{2} \cdot \alpha_{ij} \cdot \left(1 - \frac{f_{13}(r_{ij})}{r_{\text{vdW}}}\right)\right\}\right] \end{aligned} \quad (1.12)$$

$$\begin{aligned} T(r_{ij}) &= t_7 \cdot r_{ij}^7 + t_6 \cdot r_{ij}^6 + t_5 \cdot r_{ij}^5 + t_4 \cdot r_{ij}^4 \\ &\quad + t_3 \cdot r_{ij}^3 + t_2 \cdot r_{ij}^2 + t_1 \cdot r_{ij} + t_0 \end{aligned} \quad (1.13)$$

$$f_{13}(r_{ij}) = \left(r_{ij}^{p_{\text{vdW1}}} + \gamma_w^{-p_{\text{vdW1}}}\right)^{\frac{1}{p_{\text{vdW1}}}} \quad (1.14)$$

In contrast to classical force fields where van der Waal's interactions are computed only between non-bonded atom pairs, in ReaxFF all atom pairs – bonded or non-bonded – contribute to the energy term. The rationale behind this is that neglecting bonded pairs from contributing would result in discontinuities on the potential energy surface as bonds are formed or broken. To prevent extremely high repulsion forces between pairs at short distances, a shielding term is included in Section 1.1.9. Additionally, the tapering function

T in Section 1.1.9 is a seventh order polynomial with coefficients t_i, $1 \leq i \leq 7$, chosen to ensure that the van der Waal's energy smoothly goes to zero for pairs at distances beyond the non-bonded interaction cut-off distance, r_{nonb}.

1.1.10 Coulomb Term

Coulomb interactions are defined pairwise for all atoms akin to van der Waal's interactions. For corrections, shielding and taper terms are included in the potential as shown in Eq. (1.15).

$$E_{\text{Coulomb}} = C \cdot T(r_{ij}) \cdot \frac{q_i \cdot q_j}{\left[r_{ij}^3 + \gamma_{ij}^{-3}\right]^{\frac{1}{3}}} \tag{1.15}$$

Coulomb interactions are truncated within a specified cut-off r_{nonb}, typically valued in the 10 to 12 Å range. Moreover, no long-range electrostatic interactions in included in ReaxFF.

CHAPTER 2

FAST SOLVERS FOR CHARGE DISTRIBUTION MODELS ON SHARED-MEMORY PLATFORMS

This chapter presents previously published work on optimization of global charge models typically coupled with ReaxFF [12]. This work is reproduced with the permission of SIAM.

Chronologically, the first avenues explored for optimizing PuReMD focused on the shared-memory implementation. This version of the software, which prior to this work was a serial-only implementation, serves several purposes including acting as an optimization and development framework of ReaxFF parameter sets, and providing a codebase which is used for integration with external molecular software packages. Moreover, the relative simplicity of the serial nature of this implementation due to the presence of all simulation data within the same memory space enables rapid prototyping and exploration of new algorithms and avenues for performance optimization.

With the above points in mind, the shared-memory implementation was used for developing and optimizing several global charge solver models which are instrumental to polarizability [13, 12]. These models, which are described in detail in the following section, require the determination of partial atomic charges at a given instant in time. Algorithmically, this requires solving large sparse linear systems for equations at each MD timestep. From previous analysis of the performance of PuReMD, these linear solvers constitute a significant portion of the execution time – 50% and upward depending on the particular system under simulation. Thus, these kernels were clear top candidates for initial optimization efforts. In the following sections, the charge models themselves and the techniques used to optimize the linear solvers applied to these models are described.

2.1 Introduction

Molecular dynamics (MD) simulations are utilized across a wide range of fields including physics, chemistry, biology, and materials science. On one hand, the *force field*s (set of parameterized mathematical equations describing the interactions between atoms) used in these simulations must be high fidelity so that simulations can have scientific impact. On the other hand, the force field formulations need to be computationally inexpensive such that simulations of millions to billions of timesteps can be routinely performed, allowing scientists to probe into chemical or biological phenomena that take place in the microseconds to milliseconds ranges. In this regard, over the past few decades several highly successful force fields have been developed to model liquids, proteins, and materials [14, 15, 16]. The efficiency of these force fields is mainly due to the use of static bonds between atom pairs and fixed charges on each atom. While suitable for various applications, these simplifications render classical force fields unsuitable in applications where polarization effects or chemical reactions are important.

The impact of modeling charge distribution on the fidelity of MD simulations is well known [17, 18, 19, 20]. As such, to this day several charge distribution models have been developed for atomistic systems [21, 22, 23, 24]. While each of these models basically aims to find the charge distribution that minimizes the total electrostatic energy, the set of physical/chemical constraints they impose in this optimization problem is essentially what separates them. As discussed in Section 2.2.1, minimization of the electrostatic energy subject to the set of constraints defined by a particular model requires the solution of large systems of linear equations. It is important to note that trying to incorporate the long-range Coulomb effects into the dynamic charge distribution models would significantly increase their computational costs, making them impractical for actual simulations. Hence, force fields utilizing dynamic charge models adopt tapering functions which limit the effect of charges only within a certain radius (which is typically 10-12 Å). As a result, linear systems arising in the formulations of dynamic charge models are sparse, and iterative solvers are used to find sufficiently

accurate solutions.

Development of fast and efficient iterative solvers for dynamic charge models constitutes the main objective of this paper. Previous work on this topic has explored the use of Jacobi preconditioners and good initial guesses which have indeed been shown to be highly effective [11, 10]. Nevertheless, charge distribution in actual simulations still takes up a significant amount of computation time and represents an important scalability bottleneck, as demonstrated for instance for the Reax Force Field (ReaxFF) [1, 4] in Refs. [11, 8]. It can be argued that dynamic charge distribution would represent an even more significant bottleneck for other force fields if it were to be incorporated into their formulations, because ReaxFF is known to be a complex and computationally expensive force field.

In this paper, we focus on the development of effective, yet inexpensive preconditioners to accelerate the solvers used in dynamic charge distribution models, as well as high performance implementation of the resulting solvers in the open source PuReMD package [11, 10]. Presented techniques are shared memory parallel, but an efficient shared memory parallel solver constitutes the critical building block for extending our techniques to distributed memory systems. We demonstrate through extensive numerical tests that these carefully designed efficient preconditioned solvers and their efficient implementations can significantly accelerate solvers for dynamic charge models.

2.2 Dynamic Charge Distribution Models and Linear Solvers

2.2.1 Charge Equilibriation (QEq)

We define and discuss the charge distribution models and their formulations as linear systems of equations beginning with the QEq model [22]. In this model, the distribution of charges over atoms is determined by minimizing the electrostatic energy E_{ele} given the atomic positions. Let $\mathbf{R} = (\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_n)$ signify the positions of the system of n atoms, where $\mathbf{r}_i \in \mathbb{R}^3$ for $1 \leq i \leq n$. Atomic charges $\mathbf{q} = (q_1, q_2, \ldots, q_n)$, $q_i \in \mathbb{R}$, are thus defined by solving the

following problem:

$$\begin{aligned} \min_{\mathbf{q}} \quad & E_{\text{ele}}(\mathbf{q}) = \sum_i \chi_i q_i + \frac{1}{2} \sum_{i,j} H_{ij} q_i q_j \\ \text{subject to} \quad & q_{\text{net}} = \sum_i q_i. \end{aligned} \tag{2.1}$$

In Eq. (2.1), we define H_{ij} as $\delta_{ij}\eta_i + (1 - \delta_{ij}) \cdot F_{i,j}$, where δ_{ij} denotes the Kronecker delta operator; χ_i and η_i denote the atomic electronegativity and idempotential; $r_{ij} = ||\mathbf{r}_j - \mathbf{r}_i||_2$ signifies the distance between the atomic pair i and j; and $\gamma_{ij} = \sqrt{\gamma_i \cdot \gamma_j}$ denotes a pairwise shielding term tuned for element types of atoms i and j to avoid unbounded electrostatic energy at short distances. Additionally, $F_{i,j}$ is defined as

$$F_{i,j} = \begin{cases} \frac{1}{\sqrt[3]{r_{ij}^3 + \gamma_{ij}^{-3}}}, & r_{ij} \leq r_{\text{nonb}} \text{ (which is typically 10 to 12 Å)} \\ 0, & \text{otherwise.} \end{cases} \tag{2.2}$$

Applying the method of Lagrange multipliers to Eq. (2.1), we obtain the sets of linear equations below [25, 10].

$$\begin{aligned} \sum_{i=1}^{n} H_{ki} s_i &= -\chi_k, \quad k = 1, \ldots, n \\ \sum_{i=1}^{n} H_{ki} t_i &= -1, \quad k = 1, \ldots, n \end{aligned} \tag{2.3}$$

Here, the values s_i and t_i can be thought of as pseudo-charges used during the Lagrangian method. From these, partial atomic charges q_i can be computed as follows:

$$q_i = s_i - \frac{\sum_{j=1}^{n} s_j}{\sum_{j=1}^{n} t_j} \cdot t_i. \tag{2.4}$$

We henceforth refer to the coefficient matrix in Eq. (2.3) as $\mathbf{H}_{\text{QEq}} \in \mathcal{R}^{n \times n}$. We note that $\mathbf{H}_{\text{QEq}}$ is symmetric and has been observed to be positive definite across numerous atomic systems we have worked with.

2.2.2 Electronegativity Equalization (EE)

EE is another commonly used approach for partial charge calculation; it relies on the principle that charges should be distributed to atoms in order to satisfy constraints for both net

system charge and equalized atom electronegativity [26, 21]. For the latter, we represent the electronegativity of atom i as ϵ_i, and thus the electronegativity equalization constraint is formalized as

$$\epsilon_1 = \epsilon_2 = \cdots = \epsilon_n = \bar{\epsilon}. \tag{2.5}$$

Using the same definition of electrostatic energy (*i.e.*, E_{ele}) and empirical parameters in Eq. (2.1) in conjunction with the constraints in Eq. (2.5), we can represent the system of linear equations to be solved in EE in block form as a superset of those from the QEq model:

$$\begin{bmatrix} \mathbf{H}_{\text{QEq}} & \mathbf{1}_n \\ \mathbf{1}_n^T & 0 \end{bmatrix} \begin{bmatrix} \mathbf{q} \\ \bar{\epsilon} \end{bmatrix} = \begin{bmatrix} -\chi \\ q_{\text{net}} \end{bmatrix} \tag{2.6}$$

In Eq. (2.6), q_{net} again represents the net charge of the atomic system, while the EE charge matrix $\mathbf{H}_{\text{EE}} \in \mathcal{R}^{(n+1)\times(n+1)}$ is symmetric, indefinite, and sparse.

2.2.3 Atom-Condensed Kohn-Sham Approximated to Second Order (ACKS2)

The recently proposed ACKS2 model [24] can be regarded as an extension of the EE model. Essentially, ACKS2 was developed with the aim of correcting issues with accurately modeling dipole polarizability and charges during bond formation/dissociation via additional empirically fitted parameters. A block matrix representation of the linear system arising in the ACKS2 model is as follows:

$$\begin{bmatrix} \mathbf{H}_{\text{QEq}} & \mathbf{I}_n & \mathbf{1}_n & \mathbf{0}_n \\ \mathbf{I}_n & \mathbf{X} & \mathbf{0}_n^T & \mathbf{1}_n \\ \mathbf{1}_n^T & \mathbf{0}_n^T & 0 & 0 \\ \mathbf{0}_n^T & \mathbf{1}_n^T & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{q} \\ \mathbf{U} \\ \mu_{\text{mol}} \\ \lambda_U \end{bmatrix} = \begin{bmatrix} -\chi \\ \mathbf{0} \\ q_{\text{net}} \\ 0 \end{bmatrix}. \tag{2.7}$$

In Eq. (2.7), $\mathbf{U}$ contains Kohn-Sham potential coefficients, μ_{mol} and λ_U are Lagrangian multipliers from the underpinning optimization problem, and X contains terms for the linear response kernel of the Kohn-Sham potential. Similarly to the EE model, the ACKS2 matrix $\mathbf{H}_{\text{ACKS2}} \in \mathcal{R}^{(2n+2)\times(2n+2)}$ is symmetric, indefinite, and sparse.

In Table A.1 of Appendix A, we present the empirically-fitted values utilized for studying the computational behavior of the charge models discussed in this section. In addition to the parameters mentioned in Section 2.2.1, σ_i and Λ are ACKS2-specific parameters which control entries in the linear response kernel X, with the former being tuned for thresholding entries based on element types and the latter being used as a global bond softness parameter.

2.3 Preconditioning Techniques for Linear Solvers

Two commonly used techniques to accelerate the convergence of iterative solvers are using good initial guesses, and preconditioning; the latter of which essentially amounts to leveraging better spectral properties of a transformed linear system. In the context of charge distribution models, the good initial guesses approach was previously explored in the form of spline extrapolations for QEq [10] which capitalized on the key observation that in a molecular simulation environment, atomic positions change slowly due to short timestep lengths. Thus, extrapolations from solutions to the linear systems in previous timesteps are likely to provide good initial guesses.

While initial guesses are easy to apply and can be very effective, a good preconditioner is crucial to ensure fast convergence. Aktulga *et al.* have also proposed an incomplete Cholesky based preconditioner for the QEq model, albeit in a purely sequential context [10]. Due to the difficulties in attaining a scalable implementation of the incomplete Cholesky technique, actual simulations carried out on distributed memory systems (such as those with PuReMD and LAMMPS [27]) still rely on the simple Jacobi preconditioning technique in the QEq solver. As an illustration of the performance impact of this situation, 60% or more of the total runtime in large-scale ReaxFF simulations is spent in the QEq kernel, as the cost of inter-node communications in the QEq solver dominate the execution time [8].

When one considers the EE and ACKS2 models, the need for efficient solvers is exacerbated. As shown in Table A.2 of Appendix A for the systems considered in this study (and seems to hold true in general), condition numbers of the coefficient matrices for the EE

model tend to be slightly worse than those of the QEq model. For ACKS2 (where we only had empirically fitted force field parameters for the water system), the condition number is significantly worse than either QEq or EE.

Motivated by the factors above, identifying effective and efficient preconditioning techniques for the charge distribution models constitutes the main focus of this paper. One issue that prevents well-known preconditioners to be directly useful in this context is the fast execution requirement of molecular dynamics simulations. The key enabler for our use of high-quality preconditioners is the same observation that allows good initials guesses – leveraging the slowly-evolving characteristic of a molecular simulation, we amortize the cost of a preconditioner construction by re-using the same preconditioner over several timesteps. However, as we discuss below and demonstrate through numerical experiments in Section 2.4, there still exist important trade-offs that must be considered in building a practical solver:

- *Effectiveness*: How much a preconditioner can increase the convergence rate of the solver,
- *Cost*: Time to compute and apply a preconditioner,
- *Longevity*: Number of subsequent timesteps that a preconditioner can be used effectively without having to recompute it, and
- *Parallelizability*: How scalable a preconditioner is in a parallel environment.

2.3.1 Approaches

In the subsections that follow, we describe the preconditioning techniques explored, which we refer to in the remainder of the paper by the bolded notation in parentheses. The symbol **H** represents the generic coefficient matrix of the sparse linear system to be solved for a specified charge model (QEq, EE or ACKS2).

2.3.1.1 Jacobi (Jacobi)

A simple approach is to form a preconditioning matrix $\mathbf{P}^{-1}$ by taking the multiplicative inverses of the diagonal entries of $\mathbf{H}$. The ill-defined cases of zero entries on the diagonals of $\mathbf{H}_{\text{EE}}$ and $\mathbf{H}_{\text{ACKS2}}$ matrices can be treated by simply assuming those zeros to be ones for the Jacobi preconditioner.

Despite being an inexpensive and easily parallelizable preconditioner, the Jacobi preconditioner yields limited improvements in the solver convergence rate, because $\mathbf{P}^{-1}$ is a poor approximation to $\mathbf{H}^{-1}$ for the charge models considered due to the presence of a large number of off-diagonal entries (which is on the order of a few hundred per row) with not-so-insignificant nonzero values. Nevertheless, given its simplicity, Jacobi preconditioned solver performance serves as our baseline to measure the improvements obtained by the other preconditioning techniques explored.

2.3.1.2 Incomplete LU (ILU) based Techniques

The idea behind incomplete LU preconditioning is to compute and leverage an approximate $\mathbf{LU}$ decomposition $\mathbf{H} \approx \mathbf{L}'\mathbf{U}'$, where $\mathbf{L}'$ and $\mathbf{U}'$ are approximate lower and upper triangular factors, respectively. ILU techniques are known to be among the most effective preconditioners [28], but an important question is how to determine the factors $\mathbf{L}'$ and $\mathbf{U}'$, *i.e.*, which sparsity pattern to select so that the factors are computationally inexpensive to compute and apply, yet effective as a preconditioner. A common choice for the sparsity pattern is the 0-fillin approach, $\mathbf{ILU(0)}$, where $\mathbf{L}' + \mathbf{U}'$ has the same sparsity pattern as $\mathbf{H}$ [28]. Another effective choice utilizes the thresholding approach, $\mathbf{ILUT}(t)$, where the sparsity patterns in $\mathbf{L}'$ and $\mathbf{U}'$ are a subset of the 0-fillin approach in which entries smaller than a prescribed threshold t are dropped during factorization [29]. Note that since the matrices in the QEq model are symmetric positive-definite, ILU preconditioning essentially amounts to computing an incomplete Cholesky ($\mathbf{IC}$) factorization with $\mathbf{L}' = (\mathbf{U}')^T$.

Sparsification Strategies for Factor Computation: Plain **ILU(0)** and **ILUTP(t)** preconditioning turn out to be too costly to be useful in practice for our purposes. As such, we investigated a number of techniques to reduce the computational costs associated with construction and application of ILU based preconditioning techniques and improve their scaling. In particular, we developed two custom sparsification schemes for matrices arising in dynamic charge models:

- **Numeric dual drop strategy:** In this scheme, we start with a zero-fillin preconditioner (*i.e.*, **ILU(0)**, **IC(0)**), but as in **ILUTP(t)**, we apply a numerical threshold to drop small entries in the incomplete factors. Specifically, after a row in a factor is computed, the 1-norm of the row is computed, followed by a thresholding operation where *zero-fillin* entries that are less than a predetermined threshold t times the 1-norm of that row are discarded. This hybrid scheme allows us to easily determine the location of the nonzeros (by virtue of not allowing fillins), and obtain sparser factors that contain only the most significant entries (as in an **ILUTP** factorization). As such, it is an inexpensive, yet effective and easy-to-implement preconditioner for the *spd* matrices of QEq, which we refer to as IC with dual drop (**ICDD(t)**), where the parameter t denotes the numerical threshold to be used. In the cases of EE and ACKS2 models, to prevent numerical breakdowns during factorization, we replace zeros on the diagonal with ones and perform an ILU decomposition because EE and ACKS2 matrices are not spd. We denote this scheme as ILU with dual drop (**ILUDD(t)**).

- **Distance drop strategy:** A number of other works have utilized insights in the underlying application in order to select which entries to threshold [30, 31]. We choose to follow these approaches and look to the physics underpinning the charge model problem. From the definition of the coefficient matrices for the charge distribution models in Eq. (2.1), we observe that as the distance between a pair of atoms increases, the corresponding off-diagonal entry in **H** decreases proportionately to the inverse of the

distance. Said differently, small off-diagonal entries in $\mathbf{H}$ are contributed by atomic pairs separated by large distances. Intuitively, these entries contribute relatively less to the preconditioner quality; thus, they are good candidates for elimination. We denote the ILU/IC factors sparsified using the distance drop technique with the suffix **DS(d)**, *e.g.*, **ICDD(t)+DS(d)** and **ILUDD(t)+DS(d)**, where d is a real parameter in the range $(0, 1]$ that is applied as a scaling of the distance cutoff of the nonbonded interaction radius r_{nonb}. The distance drop sparsification technique can be implemented simply by constructing a sparser $\mathbf{H}$ given a prescribed parameter d, and then performing an ICDD or ILUDD factorization. As will be demonstrated through numerical experiments, distance based sparsification is highly preferred across a large majority of our benchmark cases.

Parallelization of Factor Computation and Application: In addition to the cost and effectiveness of the incomplete factors utilized, another important consideration is exploiting high degrees of parallelism available on high performance computing systems. Due to the need for eliminating data race conditions in a parallel ILU computation, the nonzero pattern of the sparse matrix plays an important role during both computation and application of the approximate factors. In this regard, it can be argued that the above sparsification schemes increase the degree of available parallelism by virtue of reducing the overlap between different rows of the coefficient matrices. To further improve the scalability of our ILU preconditioners, we implemented the techniques below:

- **Balanced Level Scheduling using Graph Coloring**: Fundamentally, this approach is based on the idea of level scheduling. Using the row dependency graph, groups of independent rows (i.e., levels) are determined. During the subsequent factorization, threads perform the scaling and elimination operations in parallel within a level, and computations between subsequent levels proceed in a lock-step fashion. Therefore the degree of parallelism is heavily dependent on the sparsity pattern.

As shown in our previous work [13], a pure level scheduling approach can produce poor thread scalability in situations where there is a large number of dependencies between the rows. In order to improve the performance of level scheduling, many other works have explored the use of reorderings using graph coloring [32, 33, 34]. For the solvers in this work, we utilize an iterative, graph coloring approach [35] to break data dependencies through an explicit permutation of $\mathbf{H}$ [36]. In this method, a permutation matrix $\mathbf{Q}$ – which increases the amount of independent nodes per level – is determined by first finding an approximate coloring of the dependency graph. This permutation matrix can be applied as $\mathbf{Q}^T\mathbf{H}\mathbf{Q}$, and subsequently factored as $\mathbf{Q}^T\mathbf{H}\mathbf{Q} \approx \mathbf{L'_P}\mathbf{U'_P}$ using level scheduling [37]. The graph coloring approach can be used with both the numerical and distance drop strategies (or their combination) discussed above.

- **Fine-grained ILU (FG-ILUDD(t, s))**: This iterative approach, recently proposed for computing the incomplete factors, seeks to address the issue of limited parallelism in ILU preconditioning [38]. Here, the factorization problem is posed as a constraint optimization problem where one seeks to satisfy a series of equations of the form

$$\sum_{k=1}^{\min(i,j)} L_{ik}U_{kj} = H_{ij} \tag{2.8}$$

by asynchronously updating individual nonzeros in the sparse factors. A single pass over all nonzeros in the factors is termed as a "sweep"; at each sweep, the factors computed by this algorithm with fine-grained parallelism get asymptotically closer to the actual $\mathbf{L'}$ and $\mathbf{U'}$ factors. We use the parameter s to denote the number of sweeps used to compute the factors in our implementation. To achieve convergence of the factors, $\mathbf{H}$ must have a unit diagonal, which may require diagonal scaling. Additionally, we assume the convention that the factor $\mathbf{L'}$ has unit diagonal entries.

Numerical dual dropping strategy can still be applied with fine grained ILU. In this scheme, which we refer to as **FG-ILUDD(t, s)**, after all sweeps are completed, we perform a post-processing step where all nonzeros are scanned and those below the

specified numerical threshold t times the 1-norm of that row are dropped. On the other hand, distance drop sparsification is straight-forward to use here because it applies the sparsification before factorization, exactly as required by the fine-grained ILU method, and it can be highly effective because our numerical experiments show that contruction of the fine-grained ILU preconditioner can be very costly despite being easily parallelizable.

2.3.1.3 Sparse Approximate Inverse (SAI(τ)))

A technique that is generally less effective than ILU preconditioning, but lends itself more easily to parallelization, is sparse approximate inverse (SAI) preconditioning. Given $\mathbf{H}$, SAI aims to find an approximate inverse matrix $\mathbf{M}$ such that the right preconditioned system $\mathbf{HMy} = \mathbf{b}$, with $\mathbf{x} = \mathbf{My}$, (or the left preconditioned system $\mathbf{MHx} = \mathbf{Mb}$) can be solved with fewer solver iterations than the original sparse linear system, and ideally with a lower overall execution time. Generally, there are three popular categories of SAI techniques: Frobenius norm minimization, factorized sparse approximate inverse, and incomplete biconjugation [39]. In this work, we focus on the first group. Hence, the objective is to minimize $||\mathbf{I} - \mathbf{HM}||$ for right preconditioning (or to minimize $||\mathbf{I} - \mathbf{MH}||$ for left preconditioning).

To control the cost of constructing as well as applying the SAI preconditioner, often $\mathbf{M}$ is chosen to be no denser than the coefficient matrix $\mathbf{H}$ itself [40]. Given the relatively large number of nonzeros per row in our problems, it is crucial that the number of nonzeros in the approximate inverses is significantly lower than that of the charge distribution matrix. Our experiments on a number of sample problems have shown that the positions of the numerically large nonzeros in the charge matrix constitute good candidates for the sparsity pattern of $\mathbf{M}$. In accordance with this observation, we select only the τ percent largest nonzeros for inclusion when computing the approximate inverse matrix.

During construction of the SAI preconditioner, a benefit of using the Frobenius form is

that we can easily exploit parallelism because the SAI factorization is formulated as:

$$||\mathbf{I} - \mathbf{HM}||_F^2 = \sum_{j=1}^{n} ||\mathbf{e}_j - \mathbf{Hm}_j||_2^2 . \tag{2.9}$$

Here, $\mathbf{e}_j$ is the j-th column of the identity matrix and $\mathbf{m}_j$ is the j-th column of $\mathbf{M}$. Leveraging the fact that $\mathbf{H}$ is sparse, we build least squares problems with smaller dense matrices $\hat{\mathbf{H}}_j$ and vectors $\hat{\mathbf{m}}_j$, where $\hat{\mathbf{H}}_j$ is obtained by eliminating the rows and columns of $\mathbf{H}$ that do not correspond to the set of nonzeros found in the row of $\mathbf{M}$ currently being computed (see Section 2.1 of [41] for details of this construction). Thus, the resulting least squares problems $\left|\left|\hat{\mathbf{e}}_j - \hat{\mathbf{H}}_j\hat{\mathbf{m}}_j\right|\right|_2^2$ can be solved independently through QR decomposition of $\hat{\mathbf{H}}_j$.

We note that the distance based sparsification of the coefficient matrix can be easily applied for SAI preconditioning as well. In fact, distance based sparsification can be highly effective in this context by reducing the cost of QR factorizations without much impact in effectiveness of the SAI preconditioner. Additionally, we utilize optimized LAPACK libraries (such as Intel MKL or Cray LibSci) to perform the QR decomposition tasks each of which are executed sequentially by different threads using dynamic assignment for load balancing purposes.

2.3.2 Dynamic Determination of Preconditioner Recomputation

One important insight considered when designing our preconditioned solver is that the molecular systems (hence the corresponding linear systems) change slowly over the course of a simulation. To leverage this insight, preconditioners are only occasionally computed and reused for several solves to amortize the preconditioner construction costs. The natural question which arises from this is exactly when to recompute the preconditioner. In Algorithm 2.1, we give a heuristic approach that adopts a gain-loss comparison perspective for recomputing the preconditioner versus reusing it.

Algorithm 2.1 Dynamic Determination of Preconditioner Recomputation.

1: Select $\alpha \geq 0$
2: **for** each simulation step **do**
3: **if** $T_{\text{loss}} \geq \alpha \cdot T_{\text{PreComp}}$ or first simulation step **then**
4: (Re)compute the preconditioner
5: Define T_{PreComp} as the preconditioner computation time from Line 3
6: Perform solve(s)
7: Define T_{ideal} as the solve time from Line 5
8: Update $T_{\text{loss}} = 0$
9: Check if the preconditioner quality is satisfactory and act accordingly
10: **else**
11: Perform solve(s)
12: Define T_{actual} as the solve time from Line 10
13: Update $T_{\text{loss}} \mathrel{+}= T_{\text{actual}} - T_{\text{ideal}}$
14: **end if**
15: **end for**

In this approach, we make the assumption that a preconditioner is most effective in terms of improving solver convergence at the simulation step in which it was computed. The preconditioned solve time required right after a preconditioner is recomputed is recorded as the ideal solve time for this preconditioner (denoted T_{ideal}). From this point onward, we accumulate the excess solve time compared to this ideal solve time (T_{loss}); this is considered the loss incurred by reusing the preconditioner. When the aggregated loss time exceeds the time it would take to recompute the preconditioner (T_{PreComp}) times α, we proceed to recompute it, otherwise we continue reusing it.

This dynamic scheme essentially justifies the reconstruction of a preconditioner by detecting when the loss of effectiveness of the preconditioner would exceed α times the cost of recomputing it, and tries to balance the gains and losses in preconditioner computation time versus other solver time (preconditioner application, solver operations including sparse matrix-dense vector multiplications and vector operations). We further study the consequences of this heuristic in further detail later in Section 2.4.2.

To provide insight into how to select the value for parameter α in Algorithm 2.1, consider the following proof for the α which minimizes total solver execution time under a fixed

preconditioner cost and linear degradation model for solve time with preconditioner reuse.

Theorem 1 (Optimal Parameterization for Fixed Preconditioner Recomputation)**.** *Suppose that for a sequence of n linear systems, a preconditioned solver is utilized where the preconditioner is recomputed and reused according to the Algorithm 2.1. Assuming that the solver performance degradation due to preconditioner reuse across solves is linear (solve time increases linearly with reuse) and the cost is identical for each time the preconditioner is recomputed, the value which minimizes total solve execution time is $\alpha = 1 \pm \sqrt{\frac{d}{T_{PreComp}}}$ where d is the increased solve time per step due to reusing the preconditioner.*

Proof. Let r be the number of solves for which the preconditioner is used. First, observe that from Line 3 of Algorithm 2.1 with our linear degradation assumption, we have

$$\begin{aligned}\alpha \cdot T_{\text{PreComp}} &= T_{\text{loss}} \\ &= 0 + d + 2d + \ldots + (r-1) \cdot d \\ &= d \cdot \frac{r^2 - r}{2}.\end{aligned}$$

Equivalently, we have $r = \sqrt{\frac{2\alpha \cdot T_{\text{PreComp}}}{d} + \frac{1}{4}} + \frac{1}{2}$. Next, considering the total solve time, we observe the following (where, for simplicity, we assume r divides n)

$$\begin{aligned}T_{\text{total}} &= \sum_{i=1}^{n} T_{\text{actual}}^{(i)} + \frac{n}{r} \cdot T_{\text{PreComp}} \\ &= n \cdot T_{\text{ideal}} + \sum_{i=1}^{n} T_{\text{loss}}^{(i)} + \frac{n}{r} \cdot T_{\text{PreComp}} \\ &= n \cdot T_{\text{ideal}} + \frac{nd}{2}\left[\sqrt{\frac{2\alpha \cdot T_{\text{PreComp}}}{d} + \frac{1}{4}} - \frac{1}{2}\right] \\ &\qquad + n \cdot T_{\text{PreComp}}\left[\sqrt{\frac{2\alpha \cdot T_{\text{PreComp}}}{d} + \frac{1}{4}} + \frac{1}{2}\right]^{-1}\end{aligned}$$

Finally, we see that when $\frac{\partial T_{\text{total}}}{\partial \alpha} = 0$, the minimum of T_{total} is achieved when $\alpha = 1 \pm \sqrt{\frac{d}{T_{\text{PreComp}}}}$. □

We note that in practice, $\sqrt{\frac{d}{T_{\text{PreComp}}}} \approx 0$, so we select $\alpha = 1$ for our experiments in Section 2.4.

2.3.3 Solver Implementation

As mentioned above, through construction of the preconditioners only occasionally, costs associated with computing the approximate inverses in SAI or the incomplete factors in ILU/IC schemes become negligible in practice. However, despite utilizing good initial guesses and the described preconditioning techniques, our solvers typically take tens to hundreds of iterations to converge depending on the charge model and the convergence tolerance used. Hence, implementation of the preconditioned solver is important from a performance point of view.

Since EE and ACKS2 models produce indefinite coefficient matrices, our implementation uses a restarted GMRES solver [42], which has in fact been observed to yield better convergence for QEq compared to a conjugate gradient and minimum residual solvers, despite the coefficient matrices being symmetric semi-positive definite in QEq. In a left preconditioned restarted GMRES(k) implementation, with k being the maximum number of iterations before restarting, the main parts are the sparse matrix-dense vector multiplications (SpMVs), the application of a preconditioner (PreApp), and the dense vector operations (VecOps). In PuReMD, the sparse coefficient matrix **H** is stored in CSR format, as such solver SpMV's are implemented by using loop parallelization over matrix rows. In the **ICDD/ILUDD** and **FG-ILUDD** approaches' PreApp phase, parallelization is applied across rows within each level of the level scheduling with graph coloring technique. For **SAI** preconditioner, PreApp essentially amounts to an SpMV operation, again using CSR format with simple parallelization of the loop over matrix rows. Finally, VecOps are also implemented by using simple loop parallelization.

Overall, with a simple Jacobi preconditioner, SpMV costs are expected to dominate the overall computation, while VecOps or PreApp times are negligible. However, for **SAI**, **ICDD/ILUDD** or **FG-ILUDD** preconditioners, PreApp time is likely to be significant.

2.4 Numerical Experiments

2.4.1 Computing Environment and Benchmark Systems

Results from numerical experiments presented in the following subsections have been obtained on Laconia, a cluster with over 400 compute nodes at Michigan State University's High Performance Computing Center [1]. Each of the base compute nodes on Laconia has 28 cores, consisting of two fourteen-core Intel Xeon E5-2680v4 Broadwell 2.4 GHz processors, and has 128 GB DDR3 2133 MHz ECC memory. Each core possesses a 64 KB L_1 cache (32 KB instruction, 32 KB data), a 256 KB L_2 cache, and the capability of running one or two user threads (i.e., hyperthreading). Between the fourteen cores on a single "Broadwell" processor, a 35 MB L_3 cache is shared. At the time of the experiments, the Laconia nodes ran CentOS version 7.6.1810 distribution of GNU/linux for x86_64 architectures, kernel version 3.10.0-957.5.1, and glibc version 2.17-260.el7_6.3.

The preconditioned solvers detailed in Section 2.3 were implemented in the shared memory version of the PuReMD ReaxFF software [10, 11]. The software was built using the GNU Compiler Collection version 8.2.0 with the optimizations enabled from the `-O3` flag. For numeric libraries, Intel MKL version 2019.1.144 was utilized. For experiments, we restricted our simulations to a single socket on Laconia to avoid performance issues due to non-uniform memory access (NUMA) effects. In these cases, the maximum number of OpenMP threads was set to 14 with no hyperthreading enabled.

For benchmarking purposes, four characteristically varied molecular systems from various application scenarios of reactive and polarizable force fields were selected. These systems, which were comprised of bulk water (H_2O), amorphous silica (SiO_2), pentaerythritol tetranitrate (PETN, $C_5H_8N_4O_{12}$), and phospholipid bilayer (not solvated in water), represent liquids, amorphous materials, perfect crystals, and soft matters, respectively. Quantitatively, each system contained between 48,000 and 78,000 atoms; thus, these systems represented

[1] https://docs.icer.msu.edu/2018_cluster_resources/

moderately sized molecular structures. Table 2.1 summarizes the systems used for performance evaluation.

Table 2.1 **Molecular systems used in shared-memory performance evaluation.** The third column indicates the number of atoms in the system (N) and the fourth column denotes the dimensions of the rectilinear simulation boxes in $\mathbb{R}^3$ in Angstroms (Å).

Name	**Chem. Rep.**	**N**	**Sim. Box Dims.** (Å)	**Category**
Water (W)	H_2O	78480	$120.9 \times 80.6 \times 80.6$	Liquid
Silica (S)	SiO_2	72000	$109.4 \times 100.4 \times 104.2$	Amorphous
PETN (P)	$C_5H_8N_4O_{12}$	48256	$114.5 \times 114.5 \times 155.5$	Crystal
Bilayer (B)	Phospholipid	56800	$82.7 \times 81.5 \times 80.0$	Soft matter

Unless indicated otherwise, all simulations were conducted with the following parameters: periodic boundary conditions enabled, 0.25 fs timestep lengths, NVE ensembles, $r_{\text{nonb}} = 10\,\text{Å}$, and 2000 total simulation timesteps. For the charge distribution solvers, the initial guesses were extrapolated from solutions in earlier timesteps, with cubic and quadratic spline extrapolation, respectively, used for the two linear systems arising in the QEq model solvers in Eq. (2.3), while cubic spline extrapolation was used for EE and ACKS2 models. All reported data have been averaged over the 2000 simulation timesteps.

Table 2.2 **Qualitative behavior of the preconditioning techniques.**

Preconditioner	**Cost**	**Longevity**	**Effectiveness**	**Parallelism**
Jacobi	Very Low	High	Low	High
ICDD(t)	Moderate	Moderate	High	Low
ILUDD(t)	High	Moderate	High	Low
FG-ICDD(t, s)	Very High	Moderate	High	High
FG-ILUDD(t, s)	Very High	Moderate	High	High
SAI(τ)	Moderate	High	Moderate/High	High

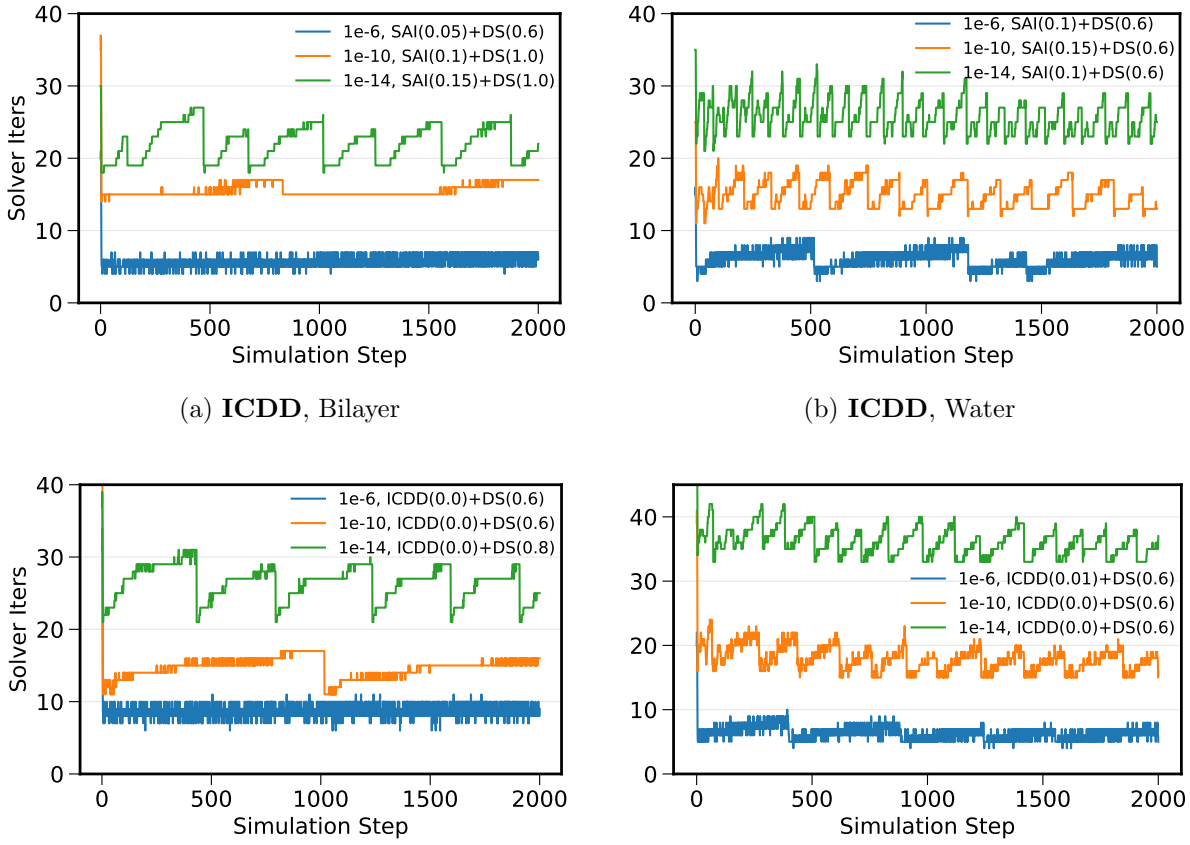
Before moving onto numerical results, in Table 2.2, we give a qualitative summary of the trade-offs involved with the preconditioning approaches we explore. As indicated in this table and demonstrated quantitatively in the subsections that will follow, it is crucial to identify preconditioners that are i) effective at reducing the number of solver iterations, ii) computationally inexpensive, and iii) easily parallelizable, especially in the PreApp phase.

In the following, we examine each of these trade-off points to identify preconditioners with the optimal overall performance.

2.4.2 Preconditioner Longevity

As discussed previously and indicated in Table 2.2, construction of ILU and SAI based preconditioners constitute a significant cost in terms of computational time. To reduce this cost, we leverage the fact that atomic positions, and hence the coefficient matrices, evolve slowly over the course of a simulation; as such a preconditioner, once computed, is reused over several timesteps. An important consideration is then for how many subsequent timesteps a preconditioner can be effective. In Fig. 2.1, we show the longevity of the **ICDD** and **SAI** preconditioners (the former of which behaves very similarly to its **FG-ICDD** and ILU-based counterparts) for the bilayer and bulk water systems under our dynamic refactorization scheme. For systems where atoms have limited mobility like the bilayer system, we observe that both preconditioners remain effective for hundreds to thousands of steps. For systems like water where atoms can move more freely, we observe that the heuristic employed in Algorithm 2.1 leads to significantly more frequent preconditioner recomputations to decrease the overall solve time. Additionally, we observe that as solver tolerances decrease, the frequency of preconditioner recomputation tends to increase as the potential losses – in terms of solve time – from using a poor quality preconditioner tend to increase (while the preconditioner construction time remains constant). To precisely quantify the frequency, we report in Table 2.3 the total number of preconditioner recomputations during the course of the 2000 step simulations.

Overall, these results provide a strong basis that preconditioner construction costs can be amortized by reusing ILU and SAI based preconditioners over hundreds of steps for high tolerances (*e.g.*, 10^{-6}) and over tens of steps for lower tolerances (*e.g.*, 10^{-10} or 10^{-14}). As will be shown in subsequent results (see Fig. 2.2), such reuse rates are sufficient to reduce the preconditioner construction costs to only a small percent of the overall solver time. As



(a) **ICDD**, Bilayer

(b) **ICDD**, Water

(c) **SAI**, Bilayer

(d) **SAI**, Water

Figure 2.1 **Longevity of preconditioners with the QEq model.** Results for the **ICDD** and **SAI** preconditioners with the bilayer (left column) and bulk water (right column) systems using the QEq model are presented at various solver tolerances.

such, the term "preconditioner cost" will solely refer to the preconditioner application cost (but not its construction cost) in the remainder of this section.

2.4.3 Tuning of Preconditioner Parameters for Cost and Effectiveness

With our proposed sparsification techniques, there is a large space of possible configurations to choose from for each preconditioner (beyond the Jacobi method). Since there exist important trade-offs regarding the effectiveness (convergence rate) and cost of different configurations, we conducted a parameter search study to determine which parameters to utilize for each scheme. In Table 2.4, we present a small snapshot from this study in which parameters

Table 2.3 **Number of recomputations for the shared-memory preconditioned solvers.** Results for the **ICDD**, **ILUDD**, **FG-ICDD**, **FG-ILUDD**, and **SAI** preconditioned solvers are presented over 2000 simulation steps.

Solver Tolerance		10^{-6}				10^{-10}				10^{-14}			
Systems		B	P	S	W	B	P	S	W	B	P	S	W
CM	Prec.												
QEq	**ICDD**	1	7	1	4	2	28	2	16	3	45	2	27
	FG-ICDD	1	2	1	2	2	9	2	4	3	13	2	6
	SAI	1	3	1	5	2	12	2	13	6	20	2	17
EE	**ILUDD**	2	4	1	2	2	9	2	7	2	13	2	11
	FG-ILUDD	1	1	1	1	1	4	1	3	1	6	1	4
	SAI	5	16	1	3	3	22	2	8	16	35	5	12
ACKS2	**ILUDD**	-	-	-	23	-	-	-	35	-	-	-	55
	FG-ILUDD	-	-	-	5	-	-	-	8	-	-	-	13
	SAI	-	-	-	9	-	-	-	12	-	-	-	22

that control preconditioner quality and sparsity for various schemes were varied for the QEq model. To illustrate the aforementioned trade-offs, focusing on **ICDD(0.0)+DS(1.0)** and **ICDD(0.01)+DS(0.8)**, we notice that the former yields fewer solver iterations (5.4 versus 5.7 for a tolerance of 10^{-6}, widening to 29.6 versus 32.0 for 10^{-14}), but the latter produces the lower mean solve times (0.01 versus 0.03 for 10^{-6}, widening to 0.03 versus 0.08 for 10^{-14}). Note that during application of the incomplete Cholesky preconditioner, each nonzero requires a single multiply/add operation, and therefore fewer number of nonzeros translates to less floating point operations. The better solve time for **ICDD(0.01)+DS(0.8)** can then be attributed to a lower preconditioner application cost as it has an order of magnitude less number of nonzeros than the **ICDD(0.0)+DS(1.0)** variant. Also, as we show in more detail in the next subsection, fewer number of nonzeros allows higher parallelizability during preconditioner application. Similar trade-offs for other techniques can be observed in Table 2.4. Overall, we notice that sparsification strategies hit the "sweet spot" in between the two extremes of achieving high effectiveness and keeping costs as low as possible.

Using insights from the above study, high quality parameters were selected for each pre-

Table 2.4 **Shared-memory preconditioner parameter comparison using the QEq charge method.** Results from bulk water simulations (6540 atoms for 100 steps) are presented for cost (number of non-zeros), mean solver iterations (SI), and mean solver time (ST) in seconds. Preconditioners were computed only once at step 0.

Solver Tolerance		10^{-6}		10^{-10}		10^{-14}	
	Cost	SI	ST	SI	ST	SI	ST
Preconditioner							
ICDD(0.0)+DS(0.6)	2.98E5	5.7	0.01	17.5	0.02	32.2	0.03
ICDD(0.0)+DS(0.8)	7.05E5	5.4	0.02	16.5	0.03	30.3	0.05
ICDD(0.0)+DS(1.0)	1.37E6	5.3	0.03	16.1	0.05	29.6	0.08
ICDD(0.01)+DS(0.6)	1.46E6	5.7	0.01	18.1	0.02	32.6	0.03
ICDD(0.01)+DS(0.8)	1.57E5	5.6	0.01	17.6	0.02	32.0	0.03
ICDD(0.01)+DS(1.0)	1.59E5	5.6	0.01	17.4	0.02	31.7	0.04
FG-ICDD(0.0,3)+DS(0.6)	2.98E5	5.6	0.01	17.4	0.03	32.0	0.05
FG-ICDD(0.0,3)+DS(0.8)	7.05E5	5.4	0.02	16.4	0.05	30.2	0.09
FG-ICDD(0.0,3)+DS(1.0)	1.37E6	5.3	0.05	16.3	0.11	29.9	0.18
FG-ICDD(0.1,3)+DS(0.6)	2.28E4	10.9	0.01	37.9	0.02	72.8	0.07
FG-ICDD(0.1,3)+DS(0.8)	2.29E4	10.8	0.02	37.9	0.02	72.6	0.07
FG-ICDD(0.1,3)+DS(1.0)	2.32E4	10.8	0.03	37.9	0.02	72.5	0.08
SAI(0.01)+DS(0.6)	6.54E3	18.9	0.01	62.2	0.04	134.0	0.12
SAI(0.01)+DS(0.8)	7.58E3	18.9	0.01	63.0	0.04	135.3	0.12
SAI(0.01)+DS(1.0)	2.09E4	10.6	0.01	35.0	0.02	72.3	0.06
SAI(0.5)+DS(0.6)	2.33E4	10.1	0.01	33.0	0.02	67.5	0.05
SAI(0.5)+DS(0.8)	6.40E4	6.7	0.01	21.0	0.02	40.4	0.03
SAI(0.5)+DS(1.0)	1.31E5	14.5	0.01	30.3	0.03	58.4	0.05
SAI(0.1)+DS(0.6)	5.36E4	6.8	0.01	21.5	0.02	41.1	0.03
SAI(0.1)+DS(0.8)	1.35E5	14.4	0.01	29.9	0.02	58.7	0.05
SAI(0.1)+DS(1.0)	2.68E5	8.4	0.01	23.0	0.01	43.2	0.04

conditioning scheme for a given molecular system and convergence tolerance by scanning the potential values of preconditioner parameters. These parameters are the dual drop threshold $t \in \{0.0, 0.1, 0.01\}$ in **ICDD**(t), **ILUDD**(t), **FG-ICDD**(s, t), and **FG-ILUDD**(s, t); the additional parameter s for the latter two methods has been the number of sweeps (ranged from 2 to 4). For SAI, the percentage of the top nonzeros to be kept (in terms of magnitude) was varied from 3% to 15% in 1.5% increments. For all solvers, distance based pruning was also investigated for values of $d \in \{0.6, 0.8, 1.0\}$. In the results presented below, the combination that yields the best execution time for each preconditioner is used. We report

these parameters in Table A.3 of Appendix A. As can be seen in that table, both the dual dropping and distance based pruning is heavily utilized across the board by different preconditioning schemes, providing evidence on their merit. We note that such a parameter search can be made an integral part of the PuReMD software itself, so that an automated tuning is performed for the particular simulation and architecture for the first few hundred to thousand steps of a simulation. Considering the fact that a production simulation typically takes millions to billions of timesteps, overheads incurred would be negligible.

Table 2.5 **Comparison of the cost and effectiveness of the shared-memory preconditioned solvers for the bulk water system.** Results for the **Jacobi**, **ICDD**, **ILUDD**, **FG-ILUDD**, and **SAI** preconditioned solvers are presented. For cost, the number of nonzeros in the preconditioner is reported for the chosen parameters at a solver tolerance of 10^{-6} (sum of factors for **ILU**-based methods). For effectiveness, the mean solver iterations (SI) and mean solve time (ST) in seconds are given for various solver tolerances (top row). For enabling easier comparisons, the solver iterations for the preconditioners have been normalized as improvement over the Jacobi preconditioner (bolded and unnormalized).

Solver Tolerance			10^{-6}		10^{-10}		10^{-14}	
CM	Prec.	Cost	SI	ST	SI	ST	SI	ST
QEq	**None**	0	0.8x	0.12	0.8x	0.51	0.9x	0.97
	Jacobi	7.84E4	**16.3**	0.09	**62.7**	0.37	**128.4**	0.80
	ICDD	3.46E6	2.7x	0.06	4.3x	0.14	5.0x	0.25
	FG-ICDD	7.16E6	2.3x	0.08	3.3x	0.21	3.8x	0.38
	SAI	6.42E5	2.6x	0.05	3.5x	0.13	3.6x	0.24
EE	**None**	0	0.8x	0.10	0.8x	0.30	0.9x	0.55
	Jacobi	7.84E4	**12.2**	0.08	**36.2**	0.23	**80.9**	0.55
	ILUDD	1.97E6	2.4x	0.05	3.9x	0.10	4.4x	0.17
	FG-ILUDD	7.32E6	1.8x	0.06	3.0x	0.16	3.4x	0.26
	SAI	6.53E5	2.8x	0.03	3.5x	0.08	3.5x	0.15
ACKS2	**None**	0	0.1x	14.98	0.1x	27.58	0.1x	64.60
	Jacobi	1.56E5	**125.6**	1.62	**216.1**	3.01	**425.7**	5.94
	ILUDD	5.07E6	5.4x	0.43	5.0x	0.81	4.5x	1.74
	FG-ILUDD	8.48E6	2.4x	1.02	2.9x	1.55	2.6x	3.38
	SAI	6.34E5	2.2x	0.72	2.3x	1.25	2.1x	3.06

Next, in Table 2.5, we demonstrate the trade-offs between preconditioner cost and ef-

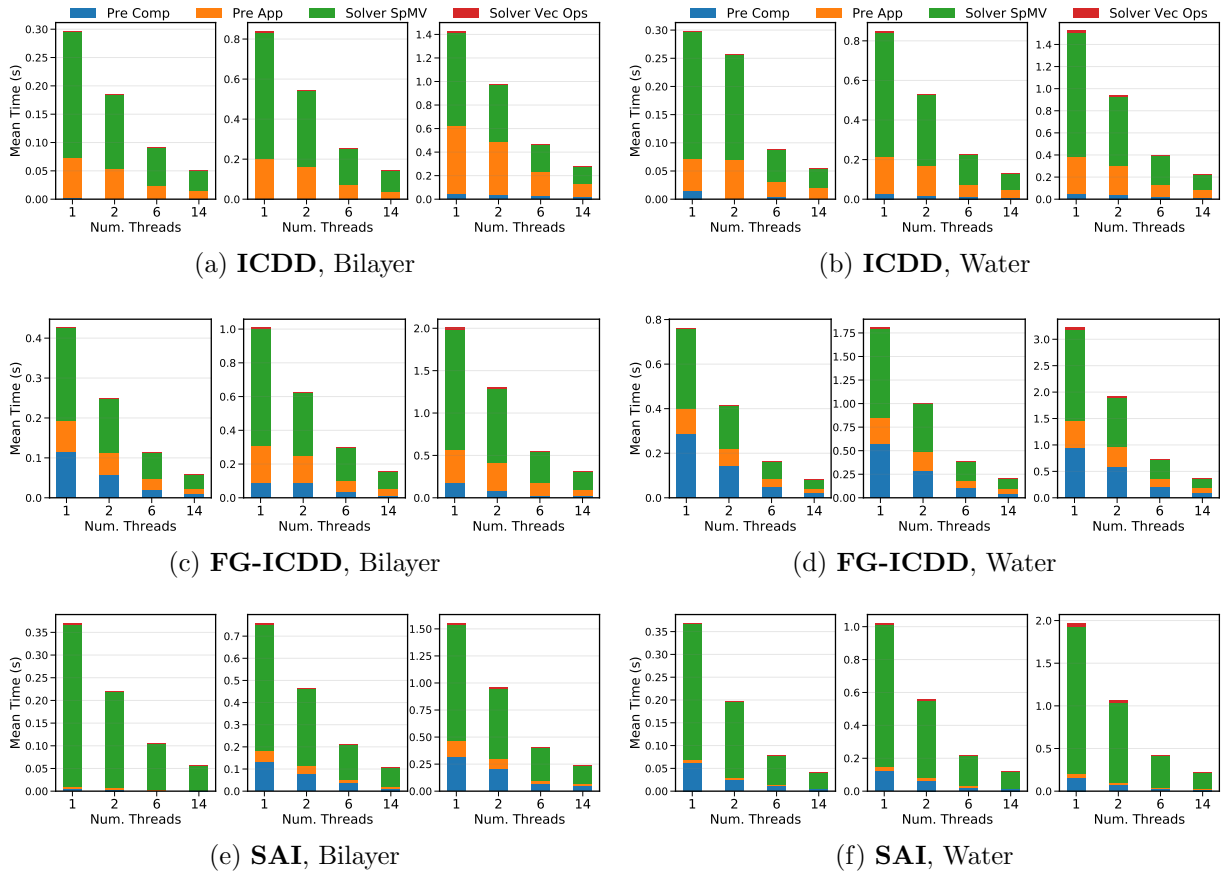
fectiveness for parameters determined as a result of the above search procedure across all charge models. As expected, despite its low cost, **Jacobi** is the least effective in improving the convergence rate, while still yielding important improvements over the unpreconditioned case, especially for the ACKS2 charge model. Incomplete factorization methods **ICDD**, **ILUDD**, **FG-ICDD**, and **FG-ILUDD** are the most effective ones. The effectiveness gap between **Jacobi** and ILU-based schemes widens as the convergence tolerance decreases from 10^{-6} to 10^{-10}, with lesser gains when moving to 10^{-14}. However, this effectiveness comes at the expense of increased preconditioner costs for which the number of nonzeros are about the same order of magnitude as the charge matrices themselves. Comparatively, the cost of **SAI** is between that of **Jacobi** and the ILU schemes, but the effectiveness of this method is closer to the latter. In addition to the cost and effectiveness examined here, parallel efficiency of applying the preconditioners represents an important consideration for overall performance in actual simulations. We inspect this issue next.

2.4.4 Scalability of Preconditioned Solvers

In order to analyze the scalability of our preconditioned solvers, strong scaling experiments were performed on a single socket of a dual socket 28-core node on Laconia. To better comprehend the trade-offs of the preconditioning approaches, the mean solver time was broken down by the major computational kernels in the solver: sparse matrix-dense vector multiplications (SpMV), preconditioner computation (PreComp), preconditioner application (PreApp), and the remaining vector operations (VecOps).

As shown in Fig. 2.2, the percentage of time spent in the preconditioner computation is insignificant for **ICDD** and **SAI** methods as a result of preconditioner reuse. However, the iterative approach used for asynchronously computing the **FG-ICDD** preconditioner takes a significant percentage of the overall solver execution time, despite reusing the preconditioners and keeping the number of sweeps very low (2-4 sweeps). The high cost of preconditioner computation renders the **FG-ICDD** method less competitive, at least on the multi-core

architectures on which we have experimented. But the fact that preconditioners generated by **FG-ICDD** exhibit convergence rates similar to those of the regular **IC** techniques shows that it can provide advantages on massively parallel architectures such as GPUs or Xeon Phis.



(a) **ICDD**, Bilayer

(b) **ICDD**, Water

(c) **FG-ICDD**, Bilayer

(d) **FG-ICDD**, Water

(e) **SAI**, Bilayer

(f) **SAI**, Water

Figure 2.2 **Strong scaling plots of shared-memory preconditioned solvers with QEq.** Results are presented for three preconditioned solvers with the bilayer (left column) and bulk water (right column) systems. Figures from left to right within a grouping show results at convergence tolerance levels of 10^{-6}, 10^{-10}, and 10^{-14}.

In terms of parallel scaling, **ICDD** achieves strong scaling efficiencies ranging from 36% to 46% through 14 threads for the bilayer and bulk water systems, as shown in Fig. 2.2. Improvements over **ICDD** are observed for **FG-ICDD** with scaling efficiencies ranging between 45% and 65%. For the **SAI** preconditioned solver, we see efficiencies between 45% and 50% for the bilayer system, and between 58% and 59% for the bulk water system.

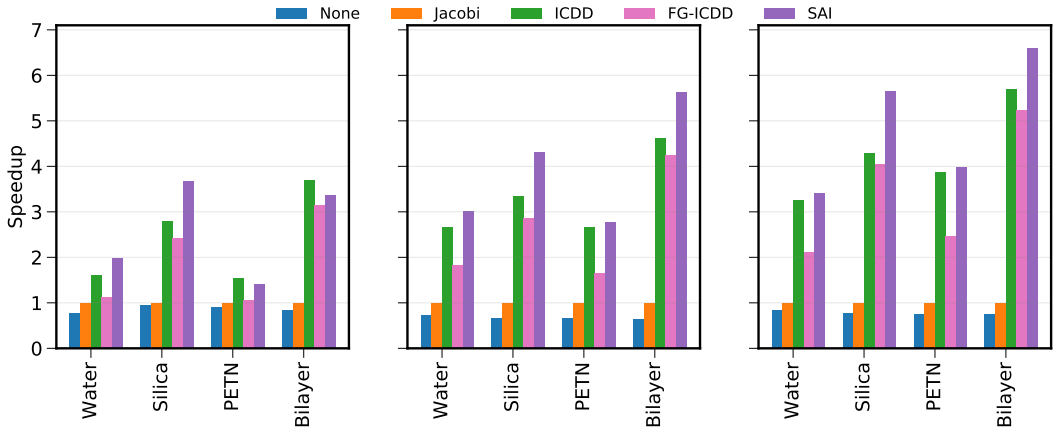
Given that the vast majority of operations involved in the preconditioned solvers have low arithmetic intensities and are therefore memory-bound, the observed efficiencies are in the expected regime.

An important observation in Fig. 2.2 is the relatively higher percentage of time spent in the PreApp phase for **ICDD** and **FG-ICDD** when compared to the **SAI** preconditioner. One would normally expect the PreApp phase of the ILU-based methods to exhibit poorer scaling than that of the **SAI** method, as the latter boils down to a simple SpMV operation, whereas the former requires level-scheduled forward and backward solves. However, we observe that the PreApp phase for the ILU-based methods (which exploits graph coloring based reordering of matrix rows) and **SAI** methods scale roughly at the same rate, which is also similar to the scaling efficiency of the SpMV kernel. As such, the reason for the high percentage of time spent by ILU-based methods in the PreApp phase is directly related to the cost of applying the preconditioners, which is essentially characterized by the number of nonzeros they contain (as given in Table 2.4 and Table 2.5).

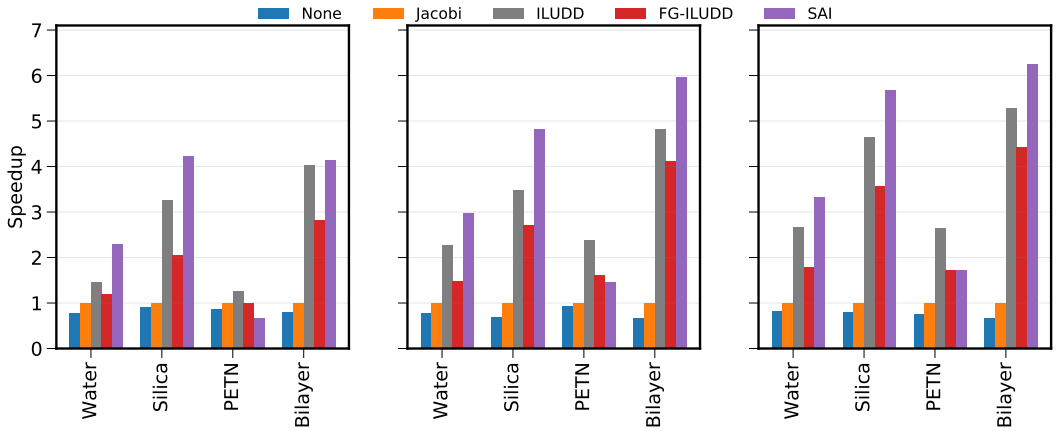
2.4.5 Preconditioned Solver Performance

In this subsection, we analyze the overall benefits of the preconditioning techniques we explored (on a single socket, *i.e.*, 14 cores, of a node on Laconia). Fig. 2.3 shows the speedups over the Jacobi preconditioner for our preconditioned solvers on the four benchmark systems. Note that we take Jacobi preconditioner as our base case due to its simplicity, but we also show the unpreconditioned solver performance for reference. We observe that for QEq, the preconditioners **ICDD**, **FG-ICDD**, and **SAI** result in speedups of 1.1 to 6.7 times the mean solve time using the Jacobi preconditioner, with speedups generally improving as the solver tolerance decreases. A similar trend is observed for the these preconditioners with the EE model.

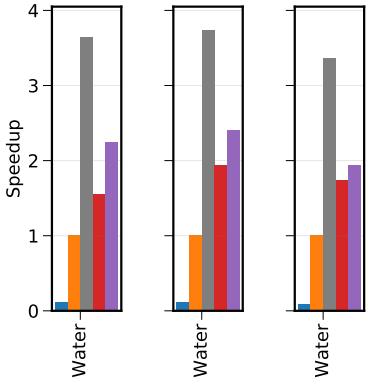
In the case of the ACKS2 model, only the force field parameters for the bulk water system were available to us. For bulk water, **SAI** produces speedups ranging from 1.8 to 2.4



(a) QEq



(b) EE



(c) ACKS2

Figure 2.3 **Speedups for shared-memory solvers for the benchmark systems by charge distribution method.** Figures from left to right show results at convergence tolerance levels of 10^{-6}, 10^{-10} and 10^{-14}, respectively. Speedups achieved are over the Jacobi preconditioned solver.

times, which are similar to the speedups observed in the QEq and EE models. On the other hand, **ILUDD** produces speedups ranging from 3.3 to 3.7 fold for ACKS2, despite being less parallelizable than SAI. In fact, **ILUDD** reduces the Jacobi solver iteration numbers by about a factor of 5 as can be seen in Table 2.5 compared to the roughly 2.2x improvement observed with SAI. This suggests that the quality of the SAI preconditioner can potentially be improved further by selecting a better sparsity pattern – we currently do not make any distinction between the $\mathbf{H}_{\mathrm{QEq}}$ and $\mathbf{X}$ subblocks in the ACKS2 matrices while selecting the sparsity pattern.

In contrast to the **ILUDD** and **SAI** techniques, the **FG-ILUDD** preconditioner leads to speedups of 1.6 to 1.9 times mostly due to the high preconditioner construction cost and lack of pivoting to handle zero diagonal elements in the charge matrix (the zero values were substituted with unit values for preconditioner computation, thereby introducing errors). One idea that was explored to improve the performance of **FG-ILUDD** was to significantly increase the convergence of incomplete factorization by raising the number of sweeps performed. However, this approach proved to be far too costly, as any gains from decreasing the solver iterations were more than outweighed by the increase in preconditioner computation time.

As previously highlighted in Table A.2, the very poor condition numbers of the ACKS2 charge matrices pose a significant challenge as the total number of solver iterations is still very large despite the limited success of the **ILUDD** and **SAI** preconditioners. This results in long solution times during simulations, leaving better preconditioners than the ones explored here to reduce the overall solve time desirable.

2.5 Conclusions

We explored several shared memory preconditioning methods to improve the performance of iterative linear solvers in the computationally expensive charge distribution models. By applying sparsification techniques, carefully tuning the relevant preconditioner parameters

and adopting an efficient dynamic refactorization scheme, we observed that incomplete factorization-based (such as **ICDD**, **ILUDD**, and **FG-ILUDD**) and SAI-based methods produce good quality preconditioners with relatively low costs. Performant implementation of these techniques on shared memory architectures results in solvers that significantly outperform the **Jacobi** baseline in time-to-solution across the majority of the charge models and molecular systems studied. As such, these techniques have been incorporated into PuReMD through an easy-to-use interface. Our future work on this topic will focus on developing GPU and distributed memory versions of the solvers explored in this study.

CHAPTER 3

PERFORMANCE OPTIMIZATION OF REACTIVE MOLECULAR DYNAMICS SIMULATIONS WITH DYNAMIC CHARGE DISTRIBUTION MODELS ON DISTRIBUTED MEMORY PLATFORMS

The following chapter presents previously published work on optimization of the distributed-memory MPI implementation of ReaxFF in PuReMD [43]. This work is reproduced with the permission of ACM.

Following the successes from the charge solver optimizations in a shared-memory context, efforts were made to apply similar preconditioning techniques together with additional communication reduction algorithms within a distributed-memory MPI-only version of PuReMD. During this work, additional observations were made which allowed other portions of the code to be optimized including reducing the number of bond order calculations through appropriate pruning of interactions.

3.1 Introduction

Molecular dynamics (MD) simulations play increasingly important roles in diverse fields, ranging from biophysics to chemistry to materials science. Classical MD techniques rely on static bonds and fixed partial charges associated with atoms, limiting their applicability to non-reactive systems. To study phenomena involving chemical reactions, quantum mechanical (QM) methods have typically been the method of choice. QM simulations must account for electronic degrees of freedom present in the system. As such, they are typically limited to sub-nanometer length and picosecond time scales. However, long-time reactive simulations are critical for several scientific problems such as catalysis, battery interfaces, biological simulations involving water, and emerging areas like surface oxidation and chemical vapor deposition (CVD) growth. Progress on these fronts is limited because long continuous time simulations of large-scale systems are very difficult, if not impossible, to perform using purely quantum mechanical (QM) or hybrid quantum mechanical/molecular dynamics (QM/MD)

simulations. The Reactive Force Field (ReaxFF) method [1], a bond order potential that bridges quantum-scale and classical MD approaches by explicitly modeling bond activity and redistribution of charges, is in principle ideally suited for this purpose. However, the computationally complex force field formulation hinders ReaxFF's scalability to large number of processing cores. In this paper, we present algorithmic and numerical techniques to address scaling bottlenecks and enable high performance ReaxFF simulations at scale.

ReaxFF is a relatively recent model (developed in early 2000s [1]) and is similar to the classical MD model in the sense that it models atomic nuclei together with their electrons as a point particle. Unlike classical MD models, ReaxFF mimics bond formation and breakage observed in QM methods by replacing the static harmonic bond models with the bond order concept, which is a quantity indicating bond strength between a pair of atoms based on the types of the atoms and the distance between them. Consequently, ReaxFF can overcome many of the limitations inherent to conventional MD. While the bond order concept dates back to 1980s and has been exploited in other force fields before (such as COMB [44] and AIREBO [3]), the distinguishing aspect of ReaxFF is the flexibility and transferability of its force field that allows ReaxFF to be applied to diverse systems of interest [1, 4, 5, 6].

ReaxFF is currently supported by major open source (PuReMD [7], LAMMPS [8], RXMD [9]) and commercial (ADF, Material Studio) software with an estimated userbase of over 1,000 groups. In this paper, we focus on the PuReMD software, as PuReMD and its LAMMPS integrations, *i.e.*, the User-ReaxC and User-ReaxC/OMP packages [8], represent the most widely used implementations of ReaxFF. PuReMD uses novel algorithms and data structures to achieve high performance while retaining a small memory footprint. An optimized neighbor generation scheme, elimination of the bond order derivatives list in bonded interactions, lookup tables to accelerate non-bonded interaction computations, and efficient iterative solvers for charge distribution are the major algorithmic innovations in PuReMD [10, 11]. PuReMD has been shown to outperform the LAMMPS/Reax package by 3-5× on various systems while using only a fraction of the memory space [11]. However, solution of

large sparse linear systems required for distribution of partial charges and computations related to dynamic bonding constitute significant bottlenecks against scalability of PuReMD. More precisely, the sparse linear solves and bond order computations at halo (ghost) regions may start accounting for a significant portion of the execution time in large runs due to communication overheads and redundant computations.

In this paper, we discuss the use of a communication-hiding conjugate gradient solver to prevent the onset of collective communication overheads at scale (Section 3.3.1), present a preconditioning technique that significantly accelerates the sparse linear solves in charge distribution (Section 3.3.2), and describe a novel technique to avoid redundant bond order computations at halo regions (Section 3.3.3). We evaluate the performance impact of these techniques through extensive tests and demonstrate that they significantly improve the execution time and scalability of large ReaxFF simulations (Section 3.4). While the techniques presented are discussed in the context of the ReaxFF model, they can directly be used in other bond order potentials. The accelerated charge model solvers can also be useful for classical force fields with polarizable charge models.

3.2 Background

The ReaxFF approach allows reactive phenomena to be modeled with atomistic resolution in a molecular dynamics framework. Consequently, ReaxFF can overcome many of the limitations inherent to conventional molecular simulation methods, while retaining, to a great extent, the desired scalability. Fig. 3.1 depicts the various ReaxFF interactions and summarizes the work flow of a simulation which includes computation of various atomic interaction functions (bonds, lone pair, over-/under-coordination, valance angles, torsions, van der Waals and Coulomb) and summing up various contributions to obtain the net force on each atom for a given time step. In terms of its implementation in PuReMD, major components of ReaxFF can be summed up as *neighbor generation*, *initialization of interaction lists*, *bonded interactions*, *charge distribution* and *non-bonded interactions*. Before presenting

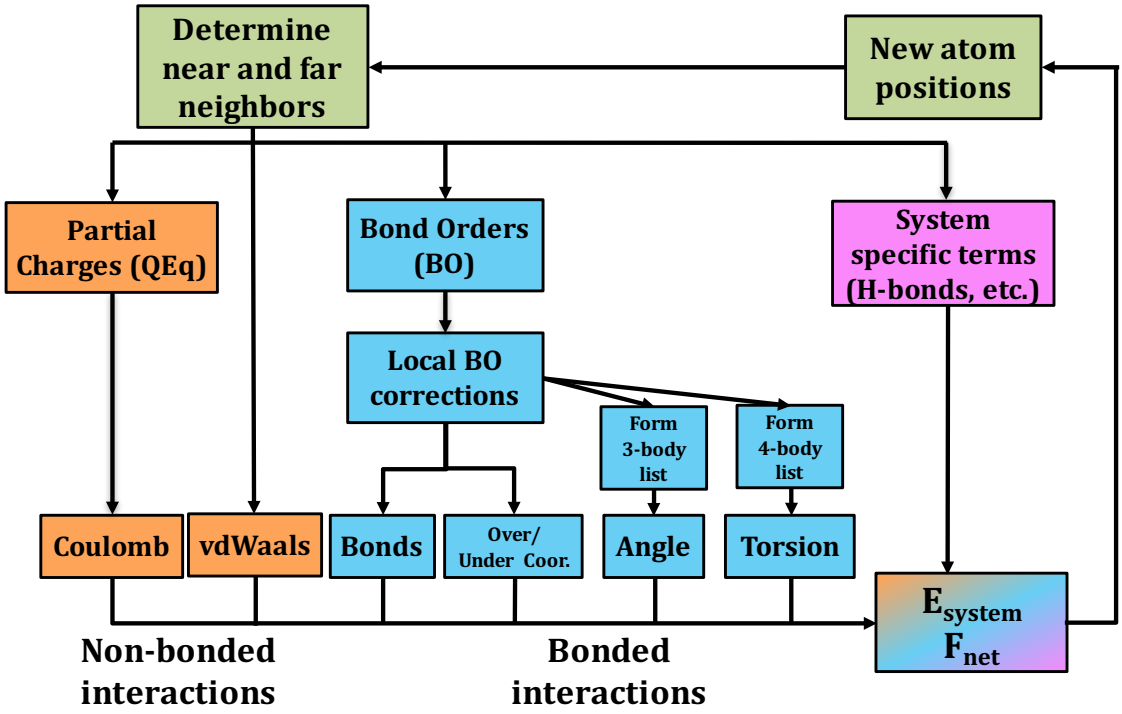


Figure 3.1 **ReaxFF's computational workflow includes bonded, non-bonded, and system-specific interactions, each with different physical formulations and cut-offs.**

our performance improvement techniques, we summarize each of these major components and the general parallelization scheme in PuReMD.

3.2.1 Parallelization

PuReMD uses the well-known domain decomposition technique where the input domain is divided into equal-sized subdomains according to a prescribed 3D Cartesian configuration of processes. Each process is then responsible for calculating the net force on all atoms in its subdomain and updating their positions based on the velocity Verlet integration scheme [45]. After position updates, atoms crossing process boundaries, if any, are transferred, and atom information at boundaries are exchanged locally (halo exchange) between processes that are neighbors in the 3D grid. Since interaction functionals in ReaxFF are very expensive computationally, interactions falling at process boundaries are computed on only one of the

processes (as explained in [11]) to prevent redundant computations. This necessitates the exchange of force information at the end of a timestep for those atoms whose coordinates were communicated at the start of the timestep.

3.2.2 Neighbor Generation

In PuReMD, neighbor lists are generated using the linked cell algorithm [45] which bins atoms into small cells based on their coordinates and generates the neighbors for each atom by only scanning atoms in cells within applicable interaction cutoffs (typically 10-12 Å). The neighbor list is stored by default as a half list, *i.e.*, for neighboring atoms i and j, only a single record is kept. A compact adjacency list format (similar to the compressed row format in sparse matrices) is used for storing the neighbor list. Neighbor generation is accelerated using the Verlet list method, which adds a buffer region on top of the largest interaction cutoff and updates the neighbors of each atom by scanning the Verlet list instead of the neighboring cells. This way, Verlet lists need to be formed from scratch (which we call *reneighboring* and is performed using the neighboring cell information) only occasionally. Reneighborings in ReaxFF can be delayed for a few hundred timesteps (typically 250 to 500) as each timestep in ReaxFF is only fractions of a femtosecond, *i.e.*, an order of magnitude smaller than the typical timestep length in classical MD.

3.2.3 Initialization of Interaction Lists

While neighbor information is needed only for non-bonded interactions (with the cutoff radii r_{nonb} typically being 10-12 Å) in most classical MD models, in ReaxFF neighbor information is needed additionally for formation of the Hamiltonian matrix for the dynamic charge model (which uses a distance cutoff identical to non-bonded Coulomb interactions), for bonded interactions (with cutoff radii r_{bond} being 4-5 Å), and for hydrogen bond interactions (with cutoff radii r_{hbond} being 7.5 Å). Consequently, PuReMD creates four interaction lists by scanning through the Verlet list, updating the atomic pair distances found therein and checking

them against various cutoffs involved. As expected, the process of creating the interaction lists is memory-bound in terms of performance because calculations associated with creation of these lists are trivial. However, the bonds list is an exception here as it requires expensive bond order calculations between pairs of atoms that are within the prescribed r_{bond} cutoff. The memory-bound nature of this kernel and expensive bond order calculations make this routine (which we denote shortly by *init*) one of the expensive parts in PuReMD.

3.2.4 Bonded Interactions

Since bonds are dynamic in bond order formalism, 3-body and 4-body interactions (which involve three and four atoms, respectively, as their names indicate) also need to be discovered on-the-fly. Accurately modeling chemical reactions and avoiding discontinuities on the potential energy surface in the presence of dynamic bonds requires almost all bonded interactions (such as bond energy, 3-body valence angle energy, and 4-body torsion energy) to have significantly more complex mathematical formulations than those found in classical MD models [1, 4]. In addition, in a reactive environment, atoms often do not achieve their optimal coordination numbers; to compensate for this, ReaxFF requires additional modeling abstractions such as lone pair, over/under-coordination, and 3-body and 4-body conjugation potentials, which introduce significant computational cost to evaluation of bonded interactions. Consequently, bonded interaction calculation costs which are insignificant in classical MD models constitute a significant part of the total execution time in ReaxFF.

3.2.5 Charge Distribution

An important requirement for correctly modeling reactions is the charge distribution procedure, which tries to approximate the partial charges on atoms using suitable models such as the Electronegativity Equilibration Method (EEM) [21] and the Charge Equilibration Method (QEq) [22]. While the chemical intuition behind these two methods is quite different, they produce identical-looking charge distributions in practice. Since QEq produces a

symmetric positive definite Hamiltonian which is easier to solve using distributed memory solvers, PuReMD uses the QEq method. In QEq, charges are determined by minimizing the electrostatic energy E_{ele}. Let $\mathbf{R} = (\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_n)$ denote the atomic positions in a system with n atoms, where $\mathbf{r}_i \in \mathbb{R}^3$. Atomic charges $\mathbf{q} = (q_1, q_2, \ldots, q_n)$, $q_i \in \mathbb{R}$, are thus defined by solving the following optimization problem:

$$\begin{aligned} \underset{\mathbf{q}}{\text{argmin}} \quad & E_{\text{ele}}(\mathbf{q}) = \sum_i \chi_i q_i + \frac{1}{2} \sum_{i,j} H_{ij} q_i q_j \\ \text{subject to} \quad & q_{\text{net}} = \sum_i q_i \\ \text{where} \quad & H_{ij} = \delta_{ij} \cdot J_i + (1 - \delta_{ij}) \cdot F_{ij} \\ & F_{ij} = \begin{cases} \frac{1}{\sqrt[3]{r_{ij}^3 + \gamma_{ij}^{-3}}}, & r_{ij} \leq r_{\text{nonb}} \\ 0, & \text{otherwise.} \end{cases} \end{aligned} \tag{3.1}$$

In Eq. (3.1), χ_i and J_i denote the atomic electronegativity and idempotential; δ_{ij} denotes the Kronecker delta operator; $r_{ij} = ||\mathbf{r}_j - \mathbf{r}_i||_2$ signifies the distance between the atomic pair i and j; and $\gamma_{ij} = \sqrt{\gamma_i \cdot \gamma_j}$ denotes a pairwise tuned parameter for elements i and j for avoiding unbounded electrostatic energy at short distances. Applying the method of Lagrange multipliers to Eq. (3.1), the sets of linear equations below are obtained [25, 10]:

$$\begin{aligned} \sum_{i=1}^{n} H_{ki} s_i &= -\chi_k, \quad k = 1, \ldots, n \\ \sum_{i=1}^{n} H_{ki} t_i &= -1, \quad k = 1, \ldots, n. \end{aligned} \tag{3.2}$$

The dimension of the linear systems in Eq. (3.2) is equal to the number of atoms in the simulation (and can be on the order of several millions to billions). Due to the computational expense of including long-range interactions during charge distribution, QEq uses a truncated electrostatic kernel and therefore $\mathbf{H}$ is a sparse matrix. Consequently, these systems must be solved approximately using iterative methods [10]. Using solutions to Eq. (3.2), partial atomic charges q_i can be determined:

$$q_i = s_i - \frac{\sum_{j=1}^{n} s_j}{\sum_{j=1}^{n} t_j} \cdot t_i.$$

3.2.6 Nonbonded Interactions

Nonbonded interactions include van der Waals and Coulomb forces, which have functional forms of the Morse and electrostatic potentials, respectively. As these interactions are pairwise interactions, they can be calculated directly by going over the Verlet list. When advantageous performance-wise, non-bonded interactions can be approximated with very high accuracy using cubic spline interpolations over lookup tables.

3.3 Methods

Charge distribution is essentially a precursor to Coulomb interactions. However, the iterative method used therein is significantly more expensive than the Coulomb interaction itself. Even worse, this iterative method requires a large number of communication operations (both local & global). We note that with the exception of position updates and force exchanges needed as part of parallelization, among all kernels described above, only the charge distribution solver requires communications. Again, among all kernels, only initialization of bonded interactions list incurs redundant calculations which may hamper scalability in large simulations. In this section, we present algorithms and numerical techniques to alleviate these inefficiencies.

3.3.1 Reduction of Global Communication Overheads in Charge Solvers

3.3.1.1 Baseline Preconditioned Conjugate Gradient

Previous versions of PuReMD used the conjugate gradient (CG) algorithm for solving for the charges in the QEq procedure [11]. CG is a member of the Krylov subspace methods; these methods seek a solution x to the sparse system $Ax = b$, with A being symmetric positive definite, which falls within the Krylov subspace $\kappa(A, b) = \{b, Ab, A^2b, \ldots\}$. Each iteration of CG increases the dimension of the subspace, and the procedure terminates when an acceptable solution is found. In the case of CG, storing all the vectors which define κ is not necessary due to orthogonality and A-orthogonality of the conjugate vectors. Computing

additional conjugate vectors with the prescribed A-orthogonality obviously requires sparse matrix vector multiplications (SpMVs), which in the parallel context of PuReMD entails two local communications over the 3D process torus – a forward communication to distribute the input vectors and a backward communication to accumulate partial results into the output vector (note that in PuReMD the Hamiltonian is stored as a half matrix to leverage symmetries). Also, the orthonormalizations require calculation of inner products and vector norms. In a parallel setting, these operations entail two *all-reduce* operations which are quite expensive global communications in large scale runs.

In PuReMD, the standard CG solver is improved by: i) producing good initial guesses, ii) solving both systems in Eq. (3.2) through simultaneous iteration, and iii) utilizing a Jacobi (diagonal) preconditioner. The initial guesses in PuReMD are based on the observation that atomic positions (and hence charges) change only slightly from timestep to timestep; as such PuReMD uses cubic and quadratic extrapolations to solutions of Eq. (3.2) from previous timesteps. The Jacobi preconditioning idea stems from the fact that the Hamiltonian **H** carries a heavy diagonal. Both techniques are inexpensive, yet quite effective; together they substantially improve the convergence rate of CG for QEq, but as we demonstrate in Section 3.4, this basic solver still requires tens of iterations per timestep, hampering the overall scalability.

3.3.1.2 Preconditioned Pipelined Conjugate Gradient

The scalability of Krylov subspace methods like CG is hampered by global communications during the calculation of inner products and vector norms. To combat this scalabilty issue, the pipelined CG (PIPECG) algorithm [46] aims to achieve lower communication latency by reducing the number of these communications to only one non-blocking global reduction per iteration. PIPECG essentially rearranges the algebraic formulation of CG in order to achieve this goal of one communication per step. As a trade-off for this, PIPECG ends up performing more computation (dense vector-dense vectors operations) per step and also ex-

hibits slightly worse convergence properties than CG (i.e., stagnation at tolerances beginning at around 10^{-11}). Algorithm 3.1 highlights the advantages and disadvantages of PIPECG by showing that the single global communication (lines 20 and 21) can be overlapped with the preconditioner application and SpMV on subsequent lines, effectively hiding some of the communication latency – specifically, an `MPI_IAllreduce` call can be started at these lines and waited on until the completion of the SpMV. Additionally, the increased number of vector operations are shown in lines 17 through 19. Because of these advantages, we implement PIPECG and analyze its impact on performance in the following section.

Algorithm 3.1 Preconditioned Pipelined Conjugate Gradient.

1: **function** PIPECG(H, x_0, b, M, τ, m_{iters})
2: $u \leftarrow Hx_0$ ▷ SpMV
3: $r \leftarrow b - u$
4: $u \leftarrow Mr$ ▷ Apply Preconditioner
5: $w \leftarrow Hu$ ▷ SpMV
6: $\delta \leftarrow w^T u$, $\gamma_{\text{new}} \leftarrow r^T u$
7: $\kappa \leftarrow \sqrt{u^T u}$, $b_{\text{norm}} \leftarrow \sqrt{b^T b}$ ▷ Global Redux
8: $m \leftarrow Mw$ ▷ Apply Preconditioner
9: $n \leftarrow Hm$ ▷ SpMV
10: $x \leftarrow x_0$, $i \leftarrow 0$
11: **while** $\frac{\kappa}{b_{\text{norm}}} \geq \tau$ And $i \leq m_{\text{iters}}$ **do**
12: **if** $i > 0$ **then**
13: $\beta \leftarrow \frac{\gamma_{\text{new}}}{\gamma_{\text{old}}}$, $\alpha \leftarrow \frac{\gamma_{\text{new}}}{\delta - \beta/\alpha \cdot \gamma_{\text{new}}}$
14: **else**
15: $\beta \leftarrow 0$, $\alpha \leftarrow \frac{\gamma_{\text{new}}}{\delta}$
16: **end if**
17: $z \leftarrow n + \beta z$, $q \leftarrow m + \beta q$, $p \leftarrow u + \beta p$
18: $d \leftarrow w + \beta d$, $x \leftarrow x + \alpha p$, $u \leftarrow u - \alpha q$
19: $w \leftarrow w - \alpha z$, $r \leftarrow r - \alpha d$
20: $\gamma_{\text{old}} \leftarrow \gamma_{\text{new}}$, $\delta \leftarrow w^T u$
21: $\gamma_{\text{new}} \leftarrow r^T u$, $\kappa \leftarrow \sqrt{u^T u}$ ▷ Global Redux
22: $m \leftarrow Mw$ ▷ Apply Preconditioner
23: $n \leftarrow Hm$ ▷ SpMV
24: $i \leftarrow i + 1$
25: **end while**
26: **return** x
27: **end function**

3.3.2 Acceleration of QEq through preconditioning

Despite reducing global communication overheads, the PIPECG solver for QEq that is described above still has excessive computation and communication costs compared to the rest of the operations that must be performed in a simulation step. Therefore, to accelerate the convergence of the PIPECG algorithm, we present a novel distributed memory preconditioning scheme based on the sparse approximate inverse (SAI) technique.

3.3.2.1 SAI Preconditioning

Given a linear system $\mathbf{Hx} = \mathbf{b}$, sparse approximate inverse (SAI) preconditioning aims to find a matrix $\mathbf{M}$ that serves as a good approximation to $\mathbf{H}^{-1}$ by selectively computing the entries of $\mathbf{H}^{-1}$. Using $\mathbf{M}$ as a preconditioner (the transformed system is $\mathbf{MHx} = \mathbf{Mb}$ in case of left preconditioning and $\mathbf{HMu} = \mathbf{b}$ where $\mathbf{x} = \mathbf{Mu}$ in case of right preconditioning), ideally the preconditioned system is expected to converge faster than the original linear system. To satisfy that, the cost of computing and applying the preconditioner should be marginal as construction of $\mathbf{M}$ is non-trivial and SAI preconditioning introduces one extra SpMV per solver iteration. Considering that the cost of SAI preconditioner construction and application are directly proportional to the number of non-zeros in $\mathbf{M}$, $\mathbf{M}$ should be chosen to be even sparser than the linear system itself $\mathbf{H}$ (note that $\mathbf{H}^{-1}$ is actually a dense matrix).

For the QEq problem, we choose the Frobenius norm minimization variant of left SAI preconditioning [47], which tries to find an $\mathbf{M}$ such that $||\mathbf{I} - \mathbf{MH}||_F$ is minimized where $||.||_F$ denotes the Frobenius norm of a matrix. In PuReMD, $\mathbf{H}$ is stored in the compressed sparse row (CSR) format, which actually makes constructing $\mathbf{M}$ as a right preconditioner more straightforward and affordable. Note that since $\mathbf{H}$ is symmetric, the left SAI preconditioner can then easily be inferred because left and right preconditioners of SAI are transposes of each other [41]. Let $\mathbf{e}_j$ be the j-th column of the $n \times n$ identity matrix and let $\mathbf{m}_j$ be the

j-th column of $\mathbf{M}$. Then we have

$$\min_{\mathbf{M}\in\mathbb{R}^{n\times n}} ||\mathbf{I}-\mathbf{HM}||_F^2 \;= \sum_{j=1}^{n} \min_{\mathbf{m_j}\in\mathbb{R}^{n\times 1}} ||\mathbf{e}_j - \mathbf{Hm}_j||_2^2 .$$

Finding $\mathbf{m}_j$ for each $||\mathbf{e_j}-\mathbf{Hm_j}||_2^2$ corresponds to solving a set of independent least squares problems, and as such $\mathbf{M}$ can be constructed column by column through a series of QR decompositions. These decompositions are performed using optimized LAPACK libraries (e.g., the Intel Math Kernel Library). Note that solving each least squares problem can be quite expensive considering the dimensions of $\mathbf{H}$. Leveraging the fact that $\mathbf{M}$ will be much sparser than $\mathbf{H}$, we reduce the computational costs of the least squares problems significantly by building dense vectors $\hat{\mathbf{m}}_j$ and $\hat{\mathbf{e}}_j$, and dense matrix $\hat{\mathbf{H}}_j$. This densification process essentially amounts to eliminating rows and columns that are entirely composed of zeros, as those rows and columns do not contribute to $\mathbf{Hm_j}$ (see Sect. 2.1 of [41] for further details).

While the densification step significantly reduces the computation cost of $\mathbf{M}$, it alone is not sufficient to make our SAI preconditioner useful in practice. *The key insight that makes SAI useful for ReaxFF in practice is the same as the one we use for solver initial guesses: because an atomic system evolves slowly, the preconditioner* $\mathbf{M}$ *computed at a certain timestep can be reused effectively for several subsequent steps.* In fact, as we demonstrate in Section 3.4, the SAI preconditioner can be useful for hundreds of steps. As such in the accelerated QEq solver, we reconstruct $\mathbf{M}$ only occasionally and amortize the SAI preconditioner computation costs over several steps.

Determining the sparsity pattern of $\mathbf{M}$, however, remains a challenge. The inverse of a sparse matrix is generally a dense matrix; for the inverses of our QEq matrices ($\mathbf{H}^{-1}$), we have observed that the magnitude of many entries is small. While it is desirable to select a sparsity pattern that captures the positions of the numerically large entries of $\mathbf{H}^{-1}$, these positions cannot be known *a priori*. Several works have suggested approaches for choosing a sparsity pattern [40], but these are neither sparser than $\mathbf{H}$ nor practical to compute in our case. During experiments with several molecular systems, we observed a strong correlation between

the positions of the numerically large entries of $\mathbf{H}$ and $\mathbf{H}^{-1}$. Therefore, we hypothesized that positions of numerically large entries in $\mathbf{H}$ are a good candidate for the sparsity pattern of $\mathbf{M}$. Subsequent experiments confirmed that including only τ percent of the numerically large entries of $\mathbf{H}$ indeed yields promising results. We remark, however, that finding the ideal τ value is non-trivial. On one hand, larger τ values (15%-20%) substantially improve the CG convergence but computation and application of $\mathbf{M}$ becomes expensive. On the other hand, smaller τ values make computation and application of $\mathbf{M}$ more affordable at the expense of smaller improvements in solver convergence. The longevity of $\mathbf{M}$ (discussed in the previous paragraph), the number of processes used, and the parallel efficiency are certainly other important considerations.

We observe that in general τ being 10% to 15% (depending on the molecular system) leads to decent improvements in total QEq solve time. While we empirically determine the ideal value of τ for experiments presented in this paper, note that MD simulations typically last for millions to billions of steps. *As such, an auto-tuning mechanism that empirically determines the ideal τ value and preconditioner reconstruction frequency on-the-fly for a given atomic system and architecture, while performing the MD simulation itself, is certainly feasible and is planned as future work.*

3.3.2.2 Parallelization of the SAI Preconditioner

After describing how we "engineer" a custom SAI preconditioner that is practical for dynamic charge models, we next detail its parallelization. While this topic is arguably well documented in the literature, our contribution here is the development of an SAI implementation that leverages problem-specific characteristics of ReaxFF for high efficiency and scalability, so that the resulting solver can perform significantly better than PuReMD's existing QEq solver that already delivers decent results with its simple Jacobi preconditioning scheme.

Our discussion focuses on parallel construction of the SAI preconditioner $\mathbf{M}$ as this step

is highly non-trivial; application of $\mathbf{M}$, which needs to be performed at each iteration of PIPECG, is trivial as it simply requires a parallel SpMV. Construction of $\mathbf{M}$ includes two main subtasks: i) pattern selection, and ii) setup and solution of the least squares problems. As both subtasks are intimately tied to the structure of the Hamiltonian $\mathbf{H}$, we begin with its description.

3.3.2.3 Structure of the Hamiltonian

In the Hamiltonian $\mathbf{H}$, a matrix entry (i, j) represents the charge affinity relation between atoms i and j. While the original charge solver in PuReMD adopts a half matrix for reducing storage and SpMV computation costs, in the SAI preconditioned solver we store the full $\mathbf{H}$ matrix in distributed memory (hence redundantly storing the symmetric entries twice), as this simplifies the setup of the least squares problems (subtask 2). With the full storage scheme, the $\mathbf{H}$ matrix is effectively 1D block partitioned because each process knows all neighbors (*i.e.*, all non-zeros in a row) of its local atoms (*i.e.*, all rows it owns).

Locally each process stores its submatrix in a structure that facilitates the construction of $\mathbf{M}$, as well as parallel SpMVs (needed in CG/PIPECG). As depicted in Fig. 3.2, lower indices are reserved for local atoms. In PuReMD, local communications between neighboring processes follows a three-stage communication scheme that respects the 3D torus topology: first, atom information is exchanged in $-x$ and $+x$ directions; then in $-y$ and $+y$ directions, and finally in $-z$ and $+z$ directions [11]. Atoms received after each stage are indexed contiguously in the atom list as well as the local Hamiltonian, and messages to be transmitted in the subsequent step are augmented with the received atom information as necessary. This communication scheme yields the indexing and organization shown in Fig. 3.2.

3.3.2.4 SAI Preconditioner Sparsity Pattern Selection

Given p processes and the filtering parameter τ, for each process local Hamiltonians $\mathbf{H}^{\mathbf{s}}_{\mathbf{loc}}$, with $1 \leq s \leq p$, the top τ percent numerically largest non-zeros of $\mathbf{H}$ are selected for the

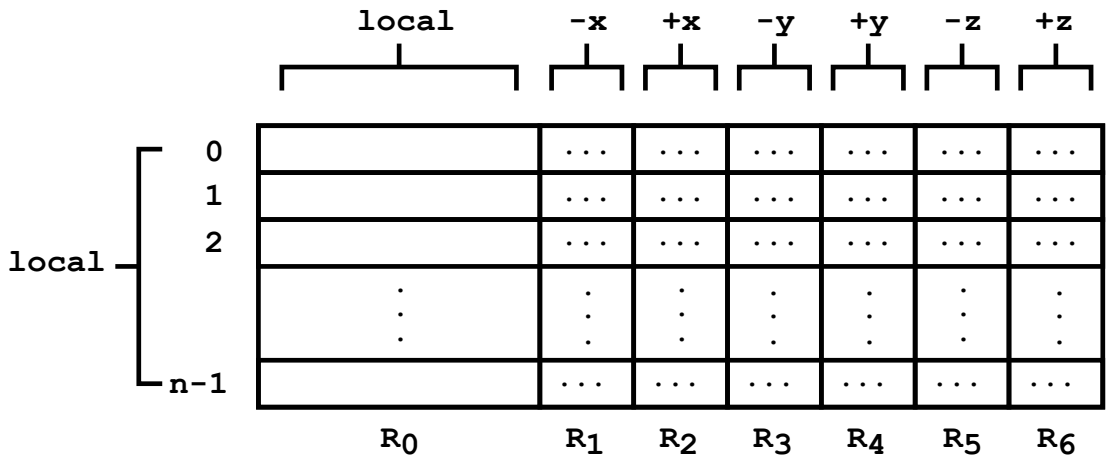


Figure 3.2 **Structure of the local charge matrix in distributed-memory simulations.** In the figure, region R_0 corresponds to the interactions between local atoms, R_1 corresponds to the interactions between local atoms and the atoms imported from $-x$ direction, etc.

sparsity pattern of $\mathbf{M}$. This problem can equivalently be stated as finding the k-th largest non-zero, namely α, in the union of all local Hamiltonians such that when entries less than α are filtered out, only τ percentile of the non-zeros remain. However, trying to find the exact α does not lend itself to an efficient parallel algorithm, but trying to be exact is not required as τ is an empirically determined threshold. As such, we approximate τ using a sampling-based technique as shown in Algorithm 3.2 (see line 2). Sampled numbers from each process are collected at the root (line 5), and these numbers are processed using a Quickselect algorithm to efficiently find α (line 7). After broadcasting α, each process can obtain their local sparsity patterns $\mathbf{H^s_{sp}}$ by filtering out entries in their local $\mathbf{H^s_{loc}}$ matrices (line 10).

3.3.2.5 SAI Preconditioner Computation

As described in Section 3.3.2.1, constructing the SAI preconditioner entails solving n least squares problems where n is the total number of atoms. These problems are independent and thus can be solved in parallel by performing QR decompositions. However, the precursor to the QR decompositions is building the least squares problems which we explain through the

Algorithm 3.2 Distributed SAI(τ) Preconditioner Sparsity Pattern Selection.

```
1: function SetupSAI(H, p, τ)
2:     P_local ← Sample(H, samplerate)
3:     s ← Reduce(length of P_local)
4:     if rank = root then
5:         P_global ← Gather(P_local)
6:         k ← ⌊s·τ / 100⌋
7:         α ← Quickselect(P_global, k)
8:     end if
9:     Bcast(α)                                  ▷ Broadcast threshold value
10:    H_sp ← FilterMatrix(H, α)
11:    return H_sp
12: end function
```

example in Fig. 3.3. After determining α, the least squares problem for atom i on process s is built based on the sparsity pattern of the i-th column of $\mathbf{H^s_{sp}}$. In this example, we form the least squares problem for atom 0, and suppose that the non-zeros in column 0 of $\mathbf{H^s_{sp}}$ are in rows 0 and 3 (non-zeros in rows 4 and 7 are below α and are dropped). Consequently, columns 0 and 3 of $\mathbf{H}$ are included, but the real challenge lies in finding the subset of rows of H that are to be included in the densified least squares problem $\hat{H}_i$. Those rows are the union of j's such that at least one of the selected columns of $\hat{H}_i$ has a non-zero element at index j. Referring to the example in Fig. 3.3, the rows incorporated into the densified matrix will be $\{0, 3, 4, 7\} \cup \{3, 5\} = \{0, 3, 4, 5, 7\}$, which are the rows with non-zero entries found in columns 0 and 3, respectively. This process is straight-forward if all atoms were to be local, but when there is a non-local atom j in the sparsity pattern of the i-th column of $\mathbf{H^s_{sp}}$, all neighbors of j are needed; this information must be imported from another process. Fortunately, when local Hamiltonians are stored as full matrices, all neighbors of atom j can be obtained from the process that owns atom j. We describe the SAI preconditioner setup operation in Algorithm 3.3. To efficiently import all the neighbors of a non-local atom, a modified version of three-stage messaging is employed in the CompSAI function. All local atoms save the number of their neighbors, which is equal to the number of non-zero entries in the corresponding row of the local matrix, to a list (line 2-3). Then, every process distributes

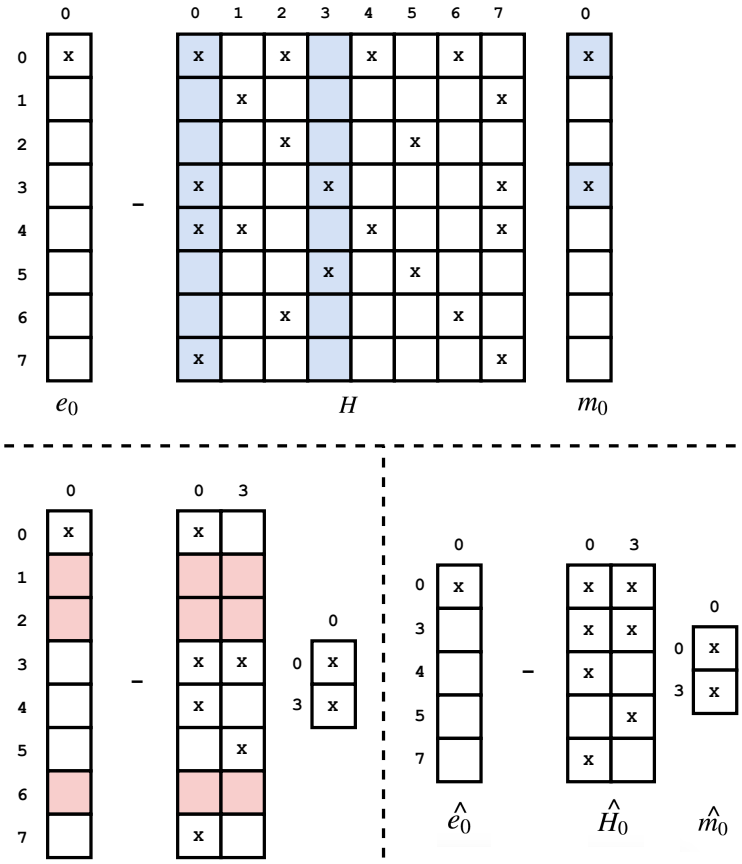


Figure 3.3 **The global view of the process of constructing and solving one of the least-squares problems for the distributed SAI preconditioning approach with an example QEq matrix.** In the figure, formation of $\hat{e}_0$, $\hat{H}_0$, and $\hat{m}_0$ are shown for an example QEq matrix $H \in \mathbb{R}^{8\times 8}$ and m_0 through column selection (blue) and row eliminations (red). In the above matrices, empty spaces and x's denote zero and non-zero entries, respectively.

their list using staged communication (line 8) so that each atom can get to learn how many non-zero entries have to be imported for each of its non-local neighbors (line 10-14). Next, the processes allocate space for those non-zero entries (line 15) and pack the non-zero entries for the local atoms that are needed by other processors (line 16-20). Then, again using the same communication scheme, each processor sends its packed entries (line 21). Finally, processors start constructing dense matrices and solving QR decompositions for each of their local atoms (line 22-25).

Algorithm 3.3 Distributed SAI(τ) Preconditioner Computation.

1: **function** CompSAI(H, H_{sp}, n, N)
2: **for** $i \leftarrow 0$ to $n-1$ **do**
3: $row\_nnz(i) \leftarrow H(i).end - H(i).start$
4: **end for**
5: **for** $i \leftarrow n$ to $N-1$ **do**
6: $row\_nnz(i) \leftarrow 0$
7: **end for**
8: $Dist(row\_nnz)$ ▷ Send non-zeros per row to neighboring processors
9: $nnz_{recv} \leftarrow 0$
10: **for** $i \leftarrow n$ to $N-1$ **do**
11: **if** $row\_nnz(i) \neq 0$ **then** ▷ If non-local atom i is needed by a processor
12: $nnz_{recv} \leftarrow nnz_{recv} + row\_nnz(i)$
13: **end if**
14: **end for**
15: $AllocateSpace(rows_{recv}, nnz_{recv})$
16: **for** $i \leftarrow 0$ to $n-1$ **do**
17: **if** atom i is needed by a neighbor **then**
18: $rows_{send}.append(H(i))$ ▷ Copy row_i non-zeros into comm. buffers
19: **end if**
20: **end for**
21: $Dist(rows_{send})$ ▷ Send non-zeros to neighboring processors
22: **for** $i \leftarrow 0$ to $n-1$ **do**
23: $\hat{e}_i, \hat{H}_i, \hat{m}_i \leftarrow Build\_Dense\_Matrix(H, H_{sp}, rows_{recv}, i)$
24: $H_{sai}(i) \leftarrow QR(\hat{e}_i, \hat{H}_i, \hat{m}_i)$
25: **end for**
26: **return** H_{sai}
27: **end function**

3.3.3 Optimization at the Ghost Regions

In ReaxFF, the precursor to bonded interactions is the calculation of the uncorrected bond orders (BOp), which is performed within the r_{bond} radii of each atom during initialization of interaction lists. Once the bond order list is initialized, all bonded interaction functions are calculated without any redundancies. For instance, a bond at the boundary between two processes is calculated by the owner of the atom with the smaller *global id*, or a 3-body interaction shared between 3 processes is calculated by the owner of the middle atom. In PuReMD, a process initializes all bonds that fall within its boundaries including the ghost regions to be prepared to handle all kinds of bonded interactions that it may end up being

responsible for. Note that in the strong scaling regime, the redundant computations associated with doing so can be significant. For instance, when the dimensions of the subdomain of a process is equal to r_{nonb}, the three dimensional import region (which must have a thickness of r_{nonb} at least in each direction) would have a volume of roughly 26 times the volume of the process's local subdomain. However, inspecting how bonded interactions are shared between processes in PuReMD reveals that one must only extend up to 3 hops into the ghost from any local atom (as expected, this happens for 4-body bonded interactions) where a hop is defined to be a bond order with strength above a certain threshold. This suggests that there can be a significant number of redundant bond order calculations in the ghost region.

To avoid redundant BOp calculations in an effort to improve the strong scaling performance, i) we adopted a BFS-style branching with multiple sources, and ii) we implemented a scheme to automatically tighten the r_{bond} cutoff for a given input system. In our first optimization, *i.e.*, the BFS-style branching scheme, all local atoms are initially added to a queue with a hop distance of 0. Then, an atom is popped from the queue at each BFS step and its neighbors within a hop distance are inserted into the queue. The method stops when all the atoms with less than 4 hop distance are inserted into the queue. Notice that this approach can be used only when full neighbor list format is employed; otherwise, we would have to calculate the hop distances of atoms in the underlying graph of a digraph, which is identical to turning a half neighbor list into a full one. Moreover, we take advantage of symmetry to optimize the computations even further as full neighbor list is the only viable option. To realize this, we ensured that the interaction between atom i and atom j is computed by the atom that is popped from the queue first.

Our second optimization, *i.e.*, automatic tightening of r_{bond}, relies on the observation that, given two atom types t_i and t_j, BOp is a monotonically decreasing function of the distance between atoms. However, r_{bond} is usually conservatively given as 4 Å (or even 5 Å) as part of simulation parameters. Instead of relying on this parameter, we scan all possible atom type pairs at several distances to precisely determine the distance after which the BOp

value for all type pairs present in the given system are guaranteed to fall below bond order acceptance threshold.

3.4 Numerical Results

3.4.1 Benchmarking Systems and Hardware

Results from numerical experiments presented in the following subsections were obtained on Cori at the National Energy Research Scientific Computing Center (NERSC). Each Haswell node in this Cray XC40 system has 32 cores, on two sixteen-core Intel Xeon E5-2698v3 Haswell 2.3 GHz processors, and has 128 GB DDR4 2133 MHz ECC memory. Each core possesses a 64 KB L_1 cache (32 KB instruction, 32 KB data), a 256 KB L_2 cache, and the capability of running one or two user threads (i.e., hyperthreading). All cores on a single processor share a 40 MB L_3 cache. At the time of the experiments, Cori ran the SuSE Linux Enterprise Server version 12.3 for x86_64 architectures, linux kernel version 4.4.162-94.72-default, and glibc version 2.22-62.16.2.

The preconditioned solvers detailed in the above sections were implemented in the PuReMD ReaxFF software [10, 11]. The software was built using the Intel Compiler Collection version 18.0.1 with the `-O3 -march=native` flags and the Cray MPICH library version 7.7.3. For relevant experiments, we restricted our simulations to one process per core (no hyperthreading was utilized).

For benchmarking purposes, two molecular systems from application scenarios of reactive and polarizable force fields were selected. These systems, which were comprised of bulk water (H_2O) and amorphous silica (SiO_2), range in size from thousands to millions of atoms depending on the experiment.

To quantify the impact of optimizations presented, we created four different versions of the PuReMD code: i) original PuReMD code as published on its website [7] (*i.e.*, CG+Jacobi [Half]), ii) PuReMD code modified to work with full neighbor lists and full Hamiltonians (*i.e.*, CG+Jacobi [Full], aims to quantify the impact of the switch to full list and matrices), iii)

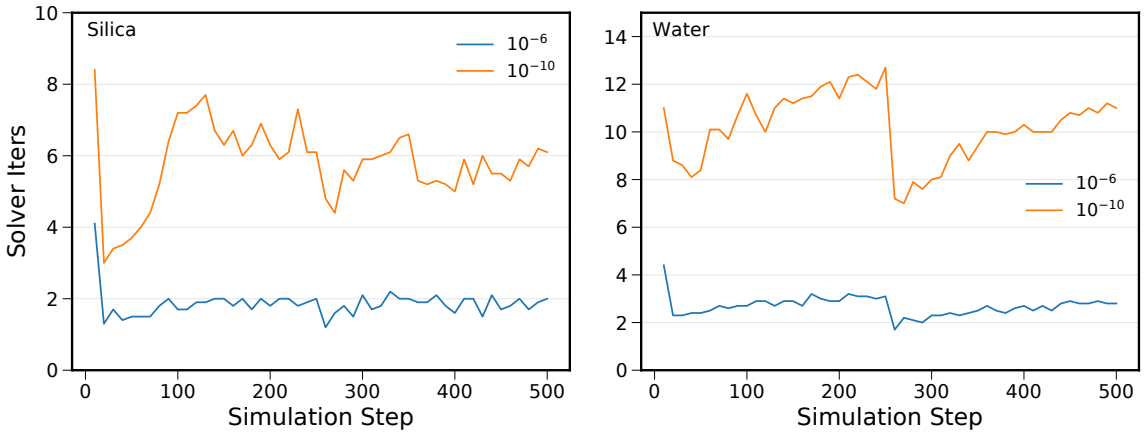


Figure 3.4 **Longevity of the CG+SAI(0.15) distributed preconditioned solver for systems using the QEq model.** In the figure, results are shown for the silica system with 864K atoms (left column), and bulk water system with 837K atoms (right column) using the QEq model at different solver tolerances. Preconditioners were utilized for 250 steps before being re-computed.

PuReMD with SAI preconditioning and ghost region optimizations (*i.e.*, CG+SAI(0.15) that denotes a τ value of 15%), and our final version which includes all presented optimizations (*i.e.*, PIPECG+SAI(0.15)).

3.4.2 Impact of Charge Solver Improvements

3.4.2.1 SAI Longevity

We start by examining for how long an SAI preconditioner can be effective, as the longevity of the preconditioner is crucial for determining how much preconditioner computation costs can be amortized, as well as for estimating the average number of solver iterations. As shown in Fig. 3.4, the SAI preconditioners can be effective for tens to hundreds of steps, especially for the silica system which is a solid material where atoms cannot move freely like they do in water. SAI preconditioner loses its effectiveness after some point, and the necessity of preconditioner reconstruction can be seen clearly. Overall, silica system requires fewer iterations for convergence than the water system at the same tolerances, and the gap becomes more apparent as the convergence tolerance is decreased to 10^{-10}.

Based on results shown in Fig. 3.4 and further empirical tests (not shown), we set the SAI reconstruction rates to 250 steps for evaluations performed in this paper. All simulations have been executed for 500 steps and averages of relevant quantities are reported, unless stated otherwise.

3.4.2.2 SAI Preconditioning Convergence Rates

In Table 3.1, we present the mean solver iterations for the Jacobi and SAI preconditioned QEq solvers at 10^{-6} and 10^{-10} tolerances for the 864K silica and 837K water systems using 512 processes. As expected, neither the switch to full matrix with CG, nor replacing the CG solver with PIPECG affects convergence rates. SAI preconditioning significantly improves the convergence rate of QEq compared to Jacobi preconditioning, yielding about 4 to 5 times better convergence rates for silica and water at 10^{-6} tolerance. The margin between SAI and Jacobi preconditioning gets larger as we go to 10^{-10} tolerance threshold both for silica and water systems. Other tests (not shown here) indicate that this trend continues as we move to even smaller tolerances, *e.g.*, 10^{-13}.

3.4.2.3 Strong Scaling Tests

While convergence results in Table 3.1 are highly encouraging regarding the speedups that can be obtained through SAI preconditioning, they omit the execution time overheads associated with SAI in actual solves which include preconditioner construction and application costs. While the preconditioner construction is amortized over several steps, preconditioner application – which essentially is a parallel SpMV – must be performed at each iteration of the QEq solver and requires an extra set of local communications; therefore it can be expensive. In Fig. 3.5, we present the results of a strong scaling study for silica and water systems. In comparing CG+Jacobi[Half] and CG+Jacobi[Full], we observe that the switch to full neighbor lists and full charge matrices does not have a significant negative effect. In fact, at a large number of cores, we observe a positive impact from CG+Jacobi[Full], because it only

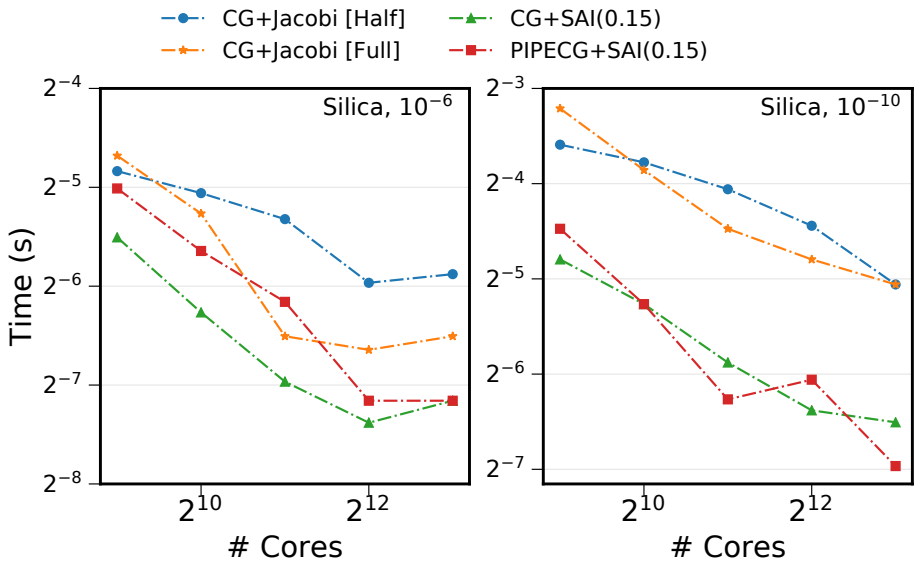
Table 3.1 **Mean solver iterations for the Jacobi and SAI distributed preconditioned QEq solvers.**

Dataset	Tol.	Solver	Iters.
Silica	10^{-6}	CG+Jacobi [Half]	11.8
		CG+Jacobi [Full]	11.8
		CG+SAI(0.15)	1.9
		PIPECG+SAI(0.15)	1.9
	10^{-10}	CG+Jacobi [Half]	39.2
		CG+Jacobi [Full]	39.2
		CG+SAI(0.15)	5.8
		PIPECG+SAI(0.15)	5.8
Water	10^{-6}	CG+Jacobi [Half]	9.5
		CG+Jacobi [Full]	9.5
		CG+SAI(0.15)	2.7
		PIPECG+SAI(0.15)	2.7
	10^{-10}	CG+Jacobi [Half]	38.4
		CG+Jacobi [Full]	38.4
		CG+SAI(0.15)	10.2
		PIPECG+SAI(0.15)	10.2

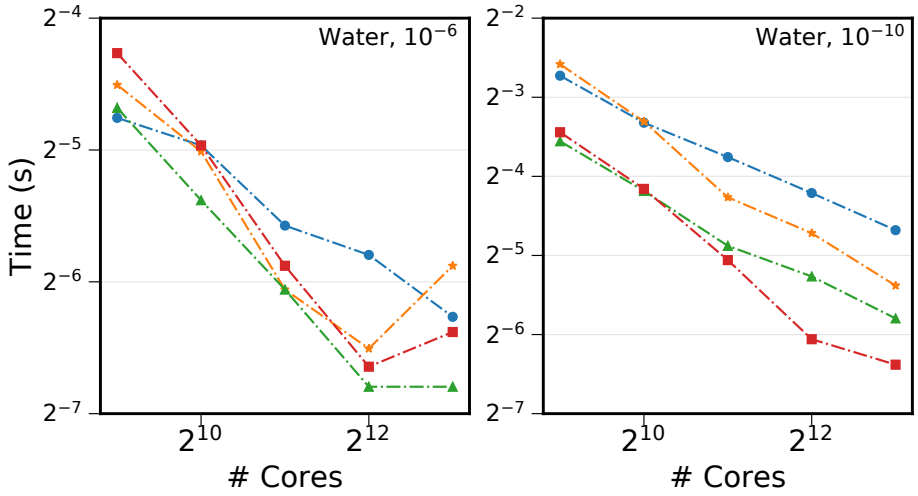
needs to do a forward communication before the SpMV in CG iterations. In comparison, CG+Jacobi[Half] needs to do a forward communication before SpMV and backward communication after SpMV. In any case, neither CG+Jacobi[Half] nor CG+Jacobi[Full] exhibit good scaling beyond 2048 cores.

By switching to SAI preconditioning, CG+SAI(0.15) is able to achieve significant speedups over the Jacobi variants by virtue of reduced CG iterations. As expected, the fully optimized PIPECG+SAI(0.15) version exhibits the best performance in large core counts, while it attains similar (or sometimes slightly worse) performance to its CG variants at smaller runs. As shown in Fig. 3.6, we observe 0.7x to 2.4x speedup at 10^{-6} tolerances, and 1.6x to 4.6x speedup at the 10^{-10} tolerances from PIPECG+SAI(0.15) over the original QEq solver in PuReMD. Again looking at Fig. 3.6, it can be said that PIPECG+SAI(0.15) almost always yields better strong scaling efficiencies than the original QEq solver.

Note that the simulation volume (hence computational load) assigned to each process



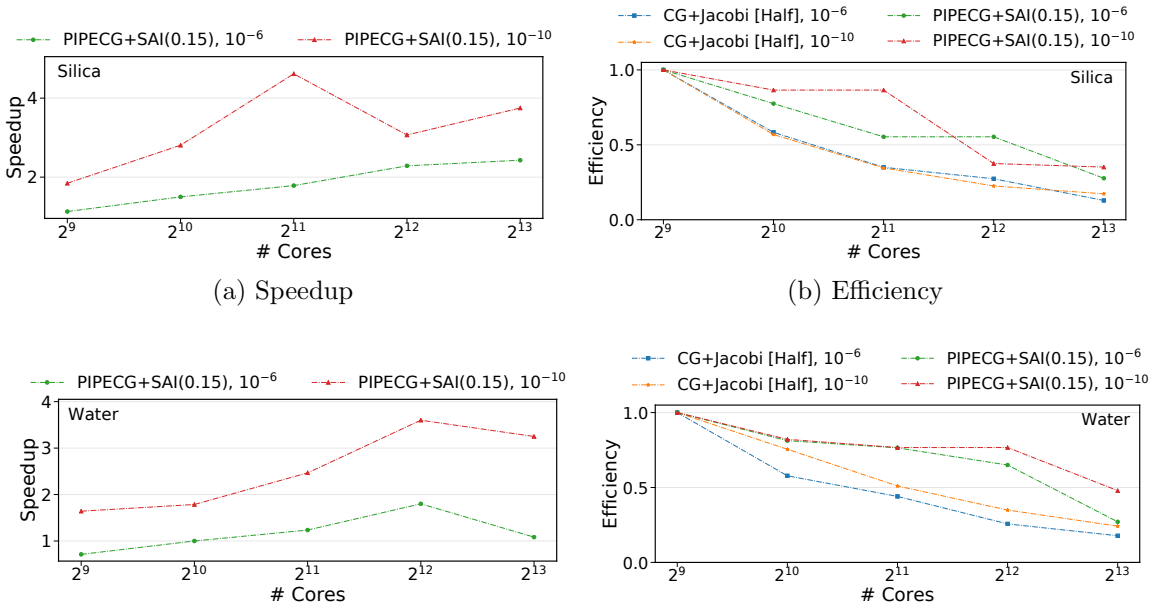
(a) **Silica 864000**



(b) **Water 837120**

Figure 3.5 **Strong scaling plots for mean QEq time of the distributed preconditioned solvers for the silica and water systems.** Figures from left to right within a row show results at convergence tolerance levels of 10^{-6} and 10^{-10}.

decreases linearly with the increase in the core count. However, the volume of the ghost region (hence local communication) decreases at a slower pace. Additionally, the overhead due to global communication operations of CG (*i.e.*, all-reduces) grows with increasing core counts. Consequently, the ratio of communication overheads to computational load on each process is maximized at high core counts, and unsurprisingly this is where the impact of SAI



(a) Speedup

(b) Efficiency

(c) Speedup

(d) Efficiency

Figure 3.6 **Speedups and parallel efficiencies for the distributed strong scaling study.** In the figures, results are shown for silica (top row) and water (bottom row). Speedups are relative to the **CG+Jacobi [Half]** baseline at the same number of cores.

and PIPECG are observed most significantly.

3.4.2.4 Weak Scaling

To better analyze the impact of core count over execution time and scaling, we also conducted weak scaling experiments. As indicated earlier, in the current PuReMD implementation, the dimensions of the simulation box that can be assigned to each process must be equal to or greater than the neighbor generation cutoff (which is equal to r_{nonb} cutoff plus the Verlet buffer size). Therefore for our weak scaling experiments, we choose bulk water and silica systems with cubic shapes that roughly have the minimum possible box dimensions per process, *i.e.*, ≈13 Å, and scale it up to 1,728 cores. As given in Fig. 3.7, PIPECG+SAI(0.15) yields speedups of between 0.9x to 2.5x for a 10^{-6} tolerance, and between 1.4x to 5.1x for a 10^{-10} tolerance over the original QEq solver. As a result of our highly optimized SAI preconditioner, SAI related overheads are minimal, allowing the reduction in iteration counts

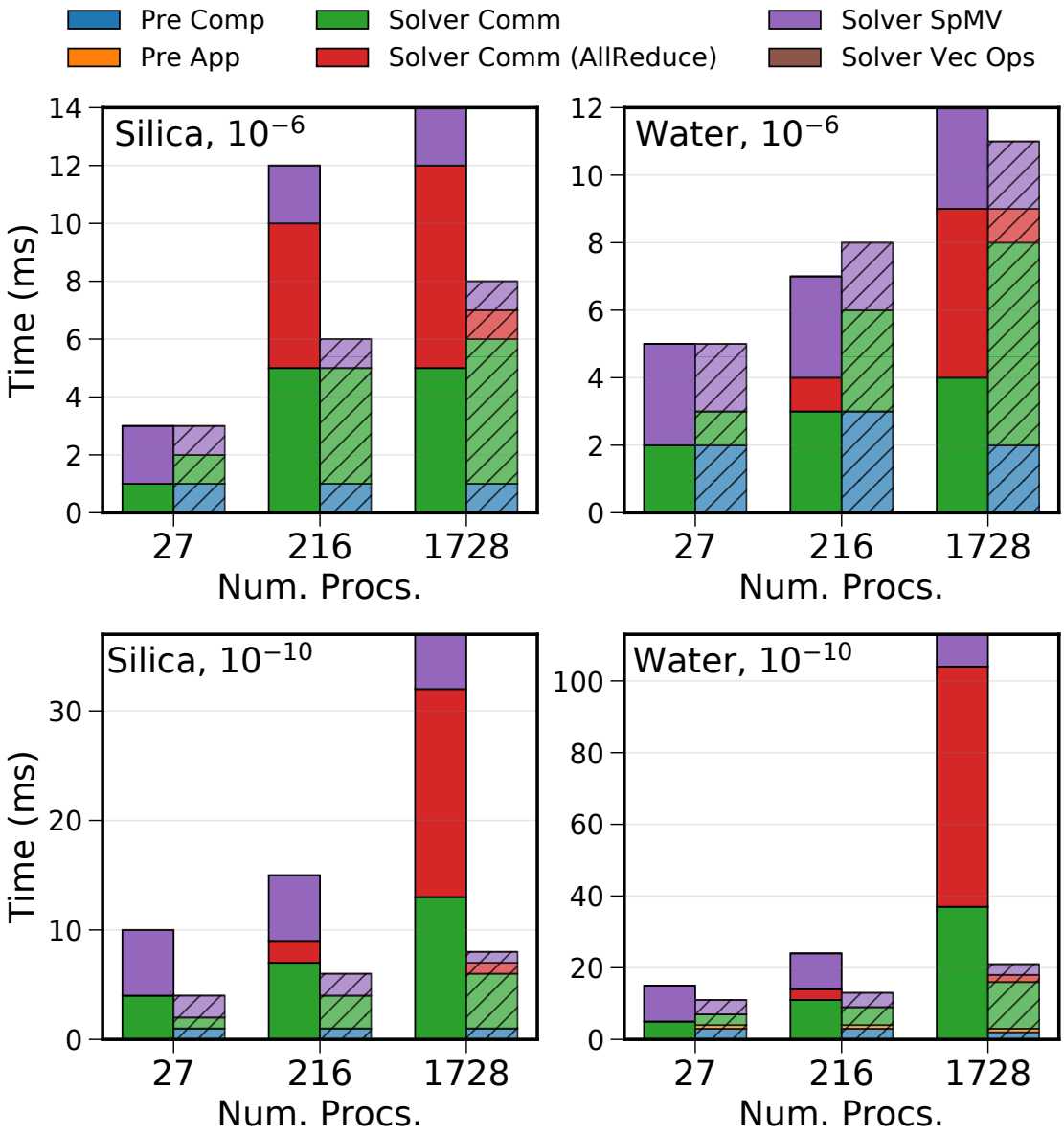


Figure 3.7 **Weak scaling plots for the distributed solver sub-kernels for QEq.** In the figures, results are shown for the silica and bulk water systems at solver tolerances of 10^{-6} and 10^{-10}. Unhatched bars (left within a grouping) are from **CG+Jacobi [Half]** while hatched bars (right within a grouping) are from **PIPECG+SAI(0.15)**.

to translate to QEq speedups. Another source of performance gain with PIPECG+SAI(0.15) is the overlapping of global communications which constitutes the most significant part of the original QEq solver on 1,728 cores.

3.4.3 Impact of Ghost Region Optimizations

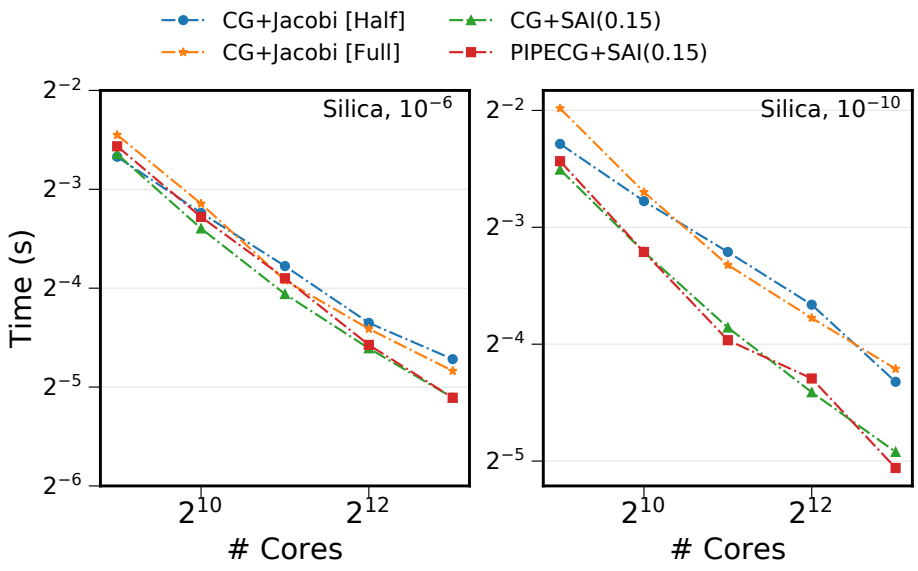
In Table 3.2, we quantify the impact of the ghost region optimizations. Since the implementation of the original interaction list initializations in PuReMD does not allow us to time the exact duration spent in bond order initializations, we rather compare the number of BOp evaluations in both schemes. As shown in this table, the presented optimizations significantly reduce the number of BOp evaluations by a factor of about 5 for water and about 2 for silica. This is expected as silica has longer maximum bond distance (3.5 Å) and therefore 3 hops into the ghost region already covers a significant volume. But, the maximum bond distance for water is shorter, and as such redundant calculations in a larger portion of ghost region can be avoided.

Table 3.2 **Average number of bond order calculations per simulation step.** Simulations were performed with the silica and bulk water systems (6000 and 6540 atoms, respectively) using 27 cores.

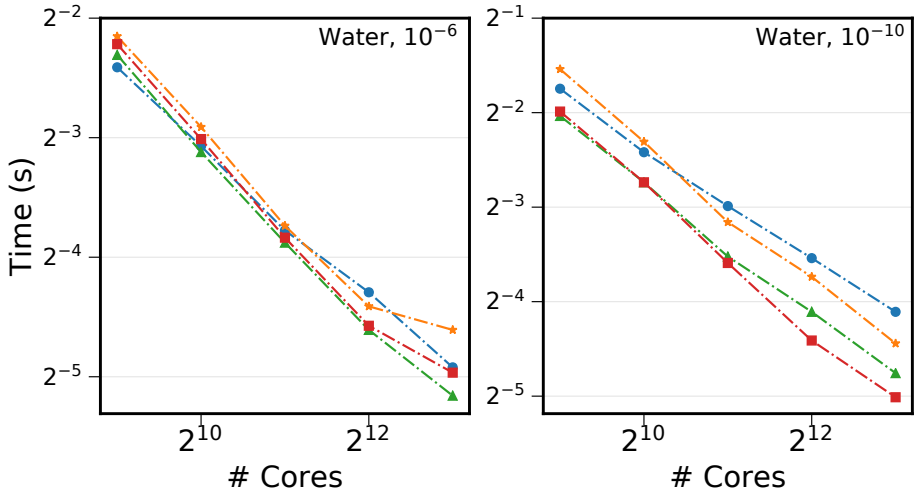
Dataset	Original Bond Init	Optimized Bond Init
Silica	2.6E5	1.1E5
Water	6.3E5	1.3E5

3.4.4 Overall Simulation Performance

In Figs. 3.8 and 3.9, we present the strong scaling and weak scaling results for the entire PuReMD simulation, which are indeed quite similar to those presented above for QEq. Since QEq solves and initialization of interaction lists constitute a significant amount of the total running time, improvements obtained for each of these kernels carry their benefits to the overall simulation times, especially when solver tolerance is reduced to 10^{-10}. In our experiments, we observed overall speedups of 1.1x to 1.9x for strong scaling experiments, and 0.9x to 1.8x for weak scaling experiments.



(a) **Silica 864000**



(b) **Water 837120**

Figure 3.8 **Strong scaling plots for total simulation time of distributed preconditioned solvers for the benchmark systems.** Figures from left to right within a row show results at convergence tolerance levels of 10^{-6} and 10^{-10}.

3.5 Conclusions

Reactive MD models fill an important void in the molecular dynamics landscape, but the scalability of existing software is limited due to the onset of communication overheads and redundant calculations, specifically due to charge distribution solvers and computations related to dynamic bond order lists. In this paper, we presented a number of novel techniques

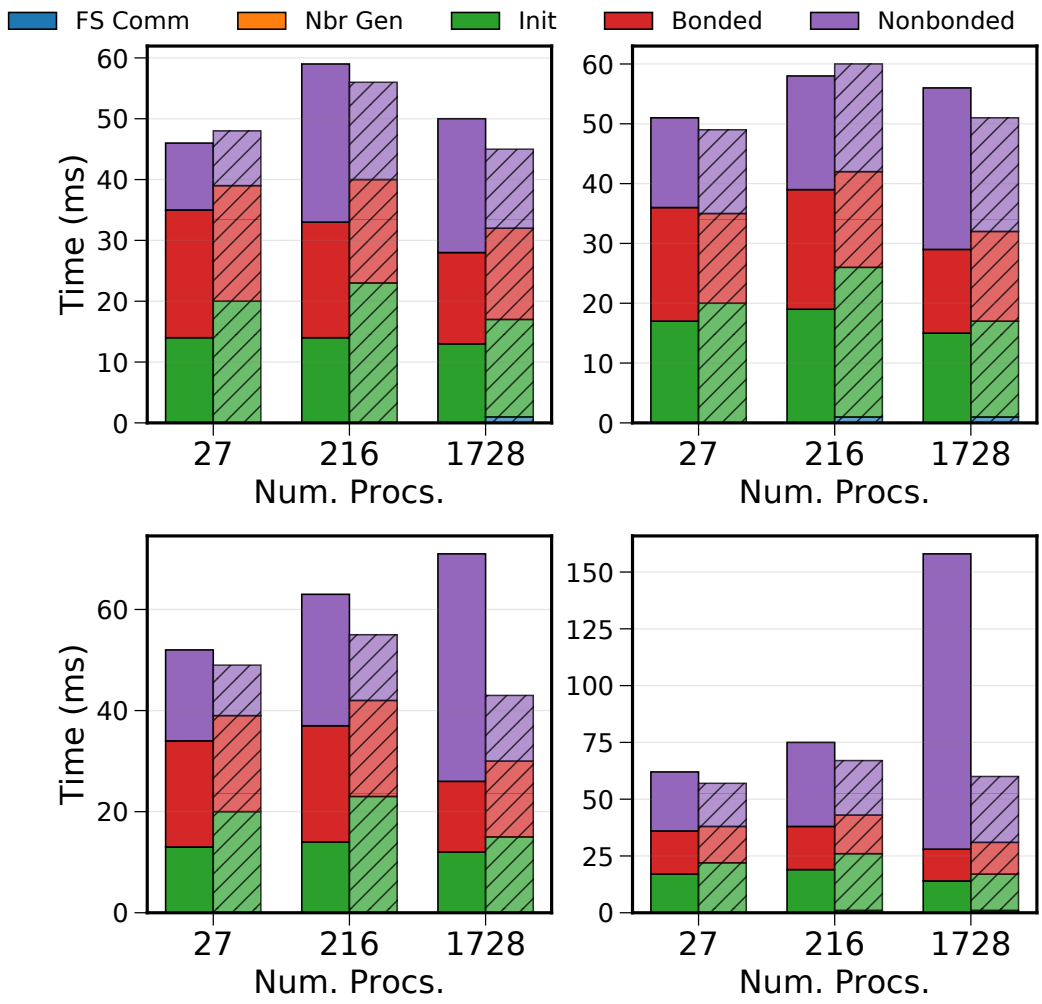


Figure 3.9 **Weak scaling plots for main simulation sub-kernels in distributed memory using the benchmark systems.** Figures depict results for silica (left column) and bulk water (right colum) systems at solver tolerances of 10^{-6} (top row) and 10^{-10} (bottom row). Unhatched bars (left within a grouping) are from **CG+Jacobi [Half]** while hatched bars (right within a grouping) are from **PIPECG+SAI(0.15)**.

to address these bottlenecks and studied their performance impact in large-scale simulations. Our results show that the presented techniques can significantly improve the overall performance and scaling of reactive MD simulations. A number of potential performance improvement opportunities have been observed as discussed above, and will be the subject of our future work in this area. While the techniques presented in this paper are discussed in the context of the ReaxFF model, they can directly be used in other bond order potentials. The accelerated charge model solvers can also be useful for classical force fields with

polarizable charge models.

CHAPTER 4

OPTIMIZATION OF THE REAX FORCE FIELD FOR THE LITHIUM-OXYGEN SYSTEM USING A HIGH FIDELITY CHARGE MODEL

This chapter presents previously published work on improving ReaxFF parameter optimization with the recently published ACKS2 charge model using the OGOLEM software integrated with a modified shared-memory version of PuReMD [48]. This work is reproduced with the permission of AIP Publishing.

While ReaxFF has been shown to be highly accurate potential with acceptable computational cost for a variety of application areas, the development of new force fields remains a challenging task which is often limited to a handful of researchers with deep domain knowledge and experience. Moreover, the high dimensionality and difficult features of the ReaxFF parameter space pose significant challenges to applying global optimization techniques, such as evolutionary genetic algorithms. To address and further diagnose these issues, a parameter optimization study was conducted with Li-O systems for ReaxFF coupled with the QEq and ACKS2 charge models. In support of this, the OGOLEM genetic algorithm framework was coupled with a modified shared-memory version of PuReMD.

4.1 Introduction

Accurately simulating chemical bond breaking is a non-trivial task. In particular, the widely used methods based on density functional theory (DFT) [49] struggle with describing bond dissociation processes. In materials science, fracture can simply be defined as a long series of bond breaking events, naturally making it a significantly more difficult task than the description of an individual bond breaking event [50].

Given the hardships faced at the quantum chemistry level, it is no surprise that fracture simulations have proven challenging for atomistic models, too. Fracture is especially pivotal in ceramic materials, which exhibit catastrophic brittle failures as opposed to the more

ductile failure modes of metals. Their failure typically starts from pre-existing cracks, in which stress is concentrated at the sharp tip of the cracks. Griffith theory [51] further connects the energy to break the bonds ahead of the crack tip with the strain energy due to the tensile stress to determine whether it is energetically favorable for a crack to propagate. This highlights the need for correct description of bond breaking in predicting fracture of ceramic materials.

Fracture simulations are typically beyond the length and time scales currently accessible to quantum-based methods. Reactive molecular dynamics (MD) simulations based on empirical force fields (FFs) allow simulation of mechanical deformation and crack propagation in materials at nanometer scales [52, 53, 54, 55, 56]. However, FF-based MD simulations tend to predict features of ductile fracture, such as void formation [57, 58], extensive sliding-induced necking [59], and crack blunting [56], even in ceramics such as Si, Ge, SiO_2, Li_2O, and $Li_x S$ that are known to exhibit brittle fracture [60, 61, 59, 62]. Although small simulation volumes, lack of preexisting defects or notches, excess surface diffusion in nanostructures, and fast strain rates can all contribute to over-ductility in MD simulations [63], the most important factor is the underlying force field. For example, Kang and Cai compared five different force fields for simulating Si and Ge nanowires under tension. Although all of them predicted reasonable elastic modulus, the predicted tensile strengths and fracture behaviors varied widely [59]. This points to an inaccurate description of bond breaking within the force field itself as a source of the error.

In this study, we use the reactive FF abbreviated as ReaxFF [1, 4] due to its applicability to a wide range of systems. In ReaxFF, reactions are simulated by dynamically computing the bond order between pairs and determining partial charges based on atomic positions. Previous work has shown that ReaxFF is well suited to study mechanical failure in a variety of systems [64, 65, 66]. However, major challenges still remain when it comes to fracture simulations.

As mentioned above, accurate description of bond breaking plays a crucial role in being

able to correctly capture the fracture behavior (ductile vs. brittle). Of the several partial energies employed in ReaxFF, bond energy and van der Waals and Coulomb interactions play the main role in bond breaking. However, the charge prediction scheme utilized in existing ReaxFF is not sufficiently accurate.

Traditionally, ReaxFF adopts the electronegativity equalization method (EEM) [26, 21] (which is closely related to the charge equilibration (QEq) method [22]) for dynamically determining charges in a simulation. Both of these methods obtain the atomic charges by minimizing the electrostatic energy at each step as a function of atomic positions. Such system-wide optimization of charges can result in instantaneous long-range charge transfer during bond breaking. For example, when a diatomic molecule dissociates and the constituent atoms move far apart, they should become neutral species rather than being charged. However, the QEq and EEM methods allow these two atoms to exchange electrons and carry fractional charges even when they are well separated. Since Coulomb interaction decays slowly, residual charges in a fracture simulation remaining after bond breaking can still lead to strong electrostatic forces between opposing surfaces which cause overestimation of the toughness and lead to unphysical ductility.

Another shortcoming in existing force fields is the training set used in parameter fitting. Typically, training set data are computed by DFT calculations for structures near the equilibrium bond distance. Since bond breaking is crucial for fracture, highly accurate bond breaking data should be included in the training set. However, as mentioned earlier, some electronic structure theory methods, such as DFT, are unable to correctly describe bond breaking processes. In those cases, highly correlated *ab initio* wave function methods, especially those using multiple suitably chosen reference determinants, such as the multireference configuration interaction (MRCI) approach developed in Refs. [67] and [68] exploited in this study, may provide the desired solution. As illustrated in Ref. [4], data from these higher-level quantum chemistry calculations can be used to train a Reax force field to correctly describe bond breaking events.

In this paper, we explore a new Reax force field concept and its ability to describe crack propagation in lithium oxide materials. Our force field development method combines advancements in the training set generation phase and the fitting protocol itself, in addition to utilizing a high-fidelity charge model to mitigate the accuracy issues with the conventional ReaxFF charge models. We created a training data set suitable for the study of crack propagation, and fitted ReaxFF parameters by adopting the atom-condensed Kohn-Sham DFT approximated to second order (ACKS2), [24] which is derived from the Kohn-Sham formulation of DFT, [69] as the charge model. The use of ACKS2 instead of QEq/EEM for charge distribution together with the training data set used lead to a significantly different ReaxFF bond description (where Coulomb interactions are heavily emphasized as they should be in an ionic system) compared to an existing ReaxFF parameter set [70]. Therefore, contributions from other energy terms had to be re-balanced by refitting those parameters as well. Since ReaxFF was developed with the capability to describe many varying chemistries, the complexity of the force field makes parameter optimization a challenging task. Oftentimes tens to hundreds of parameters must be simultaneously optimized when fitting to a large training dataset. For this purpose, a genetic algorithm (GA) based parameter optimization software was used, while the choice of parameters and their ranges were chosen carefully to avoid an over-fitted force field with incorrect physics.

We demonstrate that this new Reax force field is able to correctly describe bond breaking and charge transfer in the Li_2O molecule. This translates to improved charge prediction during bond formation and breaking in solid Li_2O systems, leading to an improved description of fracture in amorphous Li_2O. Together with improvements in formation enthalpies of both Li_2O and Li_2O_2, the resulting force field represents a major step forward in the simulation of the lithium oxidation process, and paves the way for improved atomistic simulations of lithium-air batteries as well as dendrite formation, solid-electrolyte interphase formation, and passivation layers in lithium-ion batteries. [71]

4.2 Methodology

Our newly fitted force field for the Li-O system advances in three distinct improvements: i) selection of a suitable training data set, ii) adoption of the ACKS2 model for charge prediction, iii) the fitting procedure itself. We present the details for these improvements next.

4.2.1 Training Data Set

In training the new force field, a mix of crystalline and molecular structures were used. Relative energies (with respect to a closely related reference structure) rather than absolute energies are used in the training data set. This allows us to combine different quantum chemistry methods for generating the training data to balance accuracy and efficiency. For example, DFT was used to obtain the equations of state for crystals and surface energies for slab models, while the higher-level *ab initio* MRCI methodology was employed to determine accurate partial charges and, after adding the multireference Davidson correction through the MRCI+Q approach, accurate energetics characterizing the asymmetric and symmetric bond dissocation in a model molecular system containing Li and O atoms.

The training set consisted of 276 items: energies of crystalline Li_2O, Li_2O_2, and Li metal under varying strain rates, energies of Li−Li molecules (taken from Ref. [72]), and energies and charges of Li−O−Li molecules under dissociation. Since DFT calculations are suitable to give the relative energies for systems near their equilibrium structures, energies for crystals were calculated using the DFT method in VASP [73] with PBE-GGA [74] and PAW pseudopotentials [75]. In order to correctly capture the thermodynamic driving forces for oxidation, the formation energies of crystal structures were used to train the force field:

$$E^F[\mathrm{Li}_x\mathrm{O}_y] = E[\mathrm{Li}_x\mathrm{O}_y] - (x-y)E_0[\mathrm{Li}] - \frac{y}{2}E_0[\mathrm{Li}_2\mathrm{O}_2]. \tag{4.1}$$

The required MRCI+Q calculations of the potential energy curves characterizing asymmetric and symmetric bond dissociations in the linear Li−O−Li triatomic and the corre-

sponding MRCI computations of atomic charges were performed using the MOLPRO 2010.1 program. [76] In the case of asymmetric single bond breaking, we stretched one of the Li−O bonds while keeping the other fixed at the equilibrium distance. In the case of symmetric double dissocation, we stretched both Li−O bonds symmetrically. The information about the grid of nuclear geometries used in these calculations can be found in the Appendix B. In order to perform the MRCI computations, we employed the highly efficient internally contracted MRCI algorithm developed in Refs. [67] and [68]. The underlying orbitals and reference functions were obtained with the complete active-space self-consistent field (CASSCF) approach [77, 78] using the active space of 8 valence electrons distributed among 12 valence orbitals correlating with the 2s and 2p shells of the lithium and oxygen atoms. In the subsequent internally contracted MRCI calculations, we explicitly correlated the 8 valence electrons of Li−O−Li by including all asymmetric and symmetric excitations out of the multi-determinantal reference space. The final energetics entering our training dataset included the quasi-degenerate Davidson correction through the aforementioned MRCI+Q approach. Due to the fact that the Mulliken population analysis is not reliable for lithium-containing compounds, [79] the MRCI atomic charges that form part of our training dataset were obtained using the natural bond orbital analysis of Weinhold and co-workers, [80, 81, 82] as implemented in MOLPRO 2010.1. [83] All of our MRCI and MRCI+Q calculations for the Li−O−Li system used the aug-cc-pVDZ basis set. [84, 85, 86] The complete set of results of our MRCI and MRCI+Q computations can be found in the Appendix B.

4.2.2 High Fidelity Charge Model

The residual charge problem is a common issue beyond the Li-O systems studied herein. Some *ad hoc* fixes have been proposed in the literature. Patel and Brooks add artificial constraints on atomic charges in CHARMM [87]. A different approach called atom-atom charge transfer (AACT), proposed by Chelli *et al.*, [88] introduced a dummy variable, called a split charge, to share charges across a bond. The split charge variable determines the

amount of charge transferred from one atom to the other. Later, Nistor *et al.* [23] proposed the split-charge equilibration (SQE) method which combines the EEM and AACT methods. SQE and its improved version can accurately capture some charge transfer phenomena such as static electricity transfer [89].

In this study, we adopt the more general ACKS2 method, which is a relatively recent model [24]. It is derived by expanding the Legendre transform of the Kohn-Sham kinetic energy to second order in the atomic populations and introducing a new set of dual atomic variables, the relative atomic Kohn-Sham potentials. The ACKS2 model [24] can be regarded as an extension of QEq/EEM methods. Essentially, ACKS2 was developed with the aim of correcting issues with accurately modeling dipole polarizability and charges during bond formation/dissociation via additional empirically fitted parameters.

From a computational point of view, a block matrix representation of the linear system arising in the ACKS2 model is as follows:

$$\begin{bmatrix} \mathbf{H}_{\mathrm{QEq}} & \mathbf{I}_n & \mathbf{1}_n & \mathbf{0}_n \\ \mathbf{I}_n & \mathbf{X} & \mathbf{0}_n^T & \mathbf{1}_n \\ \mathbf{1}_n^T & \mathbf{0}_n^T & 0 & 0 \\ \mathbf{0}_n^T & \mathbf{1}_n^T & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{q} \\ \mathbf{U} \\ \mu_{\mathrm{mol}} \\ \lambda_U \end{bmatrix} = \begin{bmatrix} -\boldsymbol{\chi} \\ \mathbf{q}_{\mathrm{ref}} \\ q_{\mathrm{net}} \\ 0 \end{bmatrix}. \tag{4.2}$$

In Eq. (4.2), n is the number of atoms in the system, $\mathbf{H}_{\mathrm{QEq}}$ is the $n \times n$ Hamiltonian matrix for the QEq model, $\mathbf{X}$ is an $n \times n$ matrix that contains terms for the linear response kernel of the Kohn-Sham potential, $\mathbf{I}_n$ is the n-dimensional identity matrix, $\mathbf{0}_n$ and $\mathbf{1}_n$ are n-dimensional column vectors whose individual elements equal 0 and 1, respectively, $\mathbf{q}$ is the n-dimensional column vector whose elements are the atomic charges, $\mathbf{U}$ is an n-dimensional column vector containing the coefficients defining the Kohn-Sham potential, μ_{mol} and λ_U are Lagrange multipliers from the underpinning optimization problem, $\boldsymbol{\chi}$ is an n-dimensional column vector containing the electronegativity of each atom, $\mathbf{q}_{\mathrm{ref}}$ is the set of atomic reference charges (taken as all zero for this work), and q_{net} represents the net charge of the system to be simulated.

The entries of $\mathbf{H}_{\text{QEq}}$ are defined as $H^{ij}_{\text{QEq}} = \delta_{ij}\eta_i + (1-\delta_{ij}) \cdot T_{ij}$, where δ_{ij} denotes the Kronecker delta operator; η_i denotes the atomic idempotential; and T_{ij} is defined as

$$T_{ij} = \begin{cases} \frac{1}{\sqrt[3]{r_{ij}^3 + \gamma_{ij}^{-3}}}, & \text{if } r_{ij} \leq r_{\text{nonb}} \\ 0, & \text{otherwise.} \end{cases}$$

In the above equation, r_{ij} signifies the distance between atoms i and j and $\gamma_{ij} = \sqrt{\gamma_i \cdot \gamma_j}$ denotes a pairwise shielding term tuned for element types of atoms i and j to avoid unbounded electrostatic energy at short distances. Using a tapering function, the term T_{ij} decays smoothly to zero for all pairs beyond the non-bonded interaction cutoff r_{nonb} (typically 10 Å). The diagonal entries of the linear response kernel are $X^{ii} = \sum_{j=1}^{i-1} -X^{ij}$, while the off-diagonal entries are defined as

$$X^{ij} = \begin{cases} \Lambda \cdot \left(\frac{r_{ij}}{\sigma_{ij}}\right)^3 \cdot \left(1 - \frac{r_{ij}}{\sigma_{ij}}\right)^6, & r_{ij} \leq r_{\text{min}}, X^{ij} > 0 \\ 0, & \text{otherwise.} \end{cases}$$

In the previous equation, $\sigma_{ij} = \frac{\sigma_i + \sigma_j}{2}$ and $r_{\text{min}} = \min\{r_{\text{nonb}}, \sigma_{ij}\}$. The ACKS2-specific parameters σ_i and Λ are fitted as part of the GA optimization process. σ_i thresholds entries based on element types and Λ is a global bond softness parameter.

The resulting ACKS2 matrix $\mathbf{H}_{\text{ACKS2}} \in \mathbb{R}^{(2n+2)\times(2n+2)}$ is symmetric, indefinite, and sparse. Atomic charges $\mathbf{q} = (q_1, q_2, \ldots, q_n)$, $q_i \in \mathbb{R}$ are thus obtained by solving the linear system in Eq. (4.2). Due to the size of the linear systems involved, we use the accelerated sparse solvers for ACKS2 [12, 43] in the PuReMD package [10, 11].

It has been shown that ACKS2 gives a better description of atomic charge, as well as a more accurate energy prediction [4]. Therefore, we anticipate that ACKS2 will describe the bond breaking process in Li-O with higher fidelity, and hence lead to a more accurate characterization of its brittle fracture property.

4.2.3 Optimization of the Force Field Parameters

Published parameter sets for the ReaxFF are the result of a combination of chemical intuition guided by a physical interpretation of the parameters, together with several generations of parameter fitting. Hence when an extension of the method to a new system is desired, a careful balance must be struck: adjusting parameters to better describe the new system while maintaining the core chemistry of ReaxFF. It is also desirable to maintain information from past training that is encoded in the parameters, without needing to explicitly include the entire training history of ReaxFF in a new training set (due to the computational costs of the parameter fitting phase).

A two-element system in ReaxFF (such as Li-O) typically requires simultaneous optimization of tens of parameters, or over 100 parameters for a full re-fit. In the case of the Li-O system, the bond-order parameters have already been fit to capture bond formation in this system reasonably well [4, 70, 90]. We therefore keep bond-order related parameters fixed in our fitting, with the notable exception of the lithium bond-order over-coordination correction. In the Li_2O crystal, the 2016 QEq FF finds a large number of lithium-lithium bonds in addition to the desired lithium-oxygen bonds. This is because the Li-Li distance in Li_2O is smaller than that in the lithium metal, so the bond-order terms appropriate for the metallic bonds are also activated in Li_2O. This results in a large over-coordination correction to the energy, but the forces due to the Li-Li bonds may still have undesired effects in fitting all quantum-mechanical training data and in subsequent MD simulations. We therefore turned on over-coordination correction at the bond-order level for lithium. This ensured that lithium only formed bonds with oxygen in the Li_2O solid.

While these parameters were kept fixed, adopting the ACKS2 model does require re-optimization of a significant number of force field parameters. Most obvious are the ACKS2-specific parameters: the ACKS2 softness parameter as well as the EEM shielding and atomic softness parameters for oxygen and lithium. Upon fitting these parameters to the MRCI charges and MRCI+Q energies, we observe that the use of ACKS2 instead of QEq for charge

assignment leads to significant differences in the Coulomb force. Consequently, the strengths of the other partial energy terms and how much they contribute to the total energy need to be balanced. We therefore perform a "high-level" re-fit of the lithium and oxygen bonding and angle terms, too. We find that re-fitting a total of 39 parameters (the ACKS2 softness parameter, 14 atomic parameters, 12 bonding parameters, and 12 angle parameters) provides sufficient flexibility to achieve a good fit while preserving the basic chemical properties of the underlying ReaxFF structure. Additional details are provided in Section B.1 and Table B.1.

For such high dimensional ReaxFF parameter optimization problems, recent works have utilized GAs [91, 92], which essentially are heuristic optimization techniques that take their namesake from the biological concept of evolution. In GAs, optimization proceeds using mutation and crossover operations (at a predetermined rate) until some termination criteria are reached (e.g., a solution with small enough error is found or a maximum number of generations is reached). For evaluation of the solutions, a fitness function is used:

$$F_i = \sum_{j=1}^{T} \left(\frac{x_{\text{ref}}^{(j)} - x_{\text{actual}}^{(i,j)}}{\sigma^{(j)}} \right)^2 . \tag{4.3}$$

In Eq. (4.3), F_i is the fitness score for organism i, j denotes an index over the items in the training dataset (T total items), $x_{\text{ref}}^{(j)}$ denotes the reference values for the j-th training item with weight $\sigma^{(j)}$, and $x_{\text{actual}}^{(i,j)}$ denotes the computed value for training item j using the force field parameters associated with organism i.

In this study, we utilized the OGOLEM software [93], a global optimization GA framework aimed at computational chemistry problems, including the fitting of MD force field parameters. In conjunction with OGOLEM, the PuReMD ReaxFF software was used for evaluation of the parameter sets and the MD simulations [11, 10, 94]. Novel to this work has been the integration of ACKS2 model into PuReMD for optimization with OGOLEM.

Several identical fitting runs were performed in parallel using OGOLEM; a model was deemed to have converged when fitness as a function of GA generation leveled off (see Fig. 4.1). Among the converged models, the force field that performed well on the training

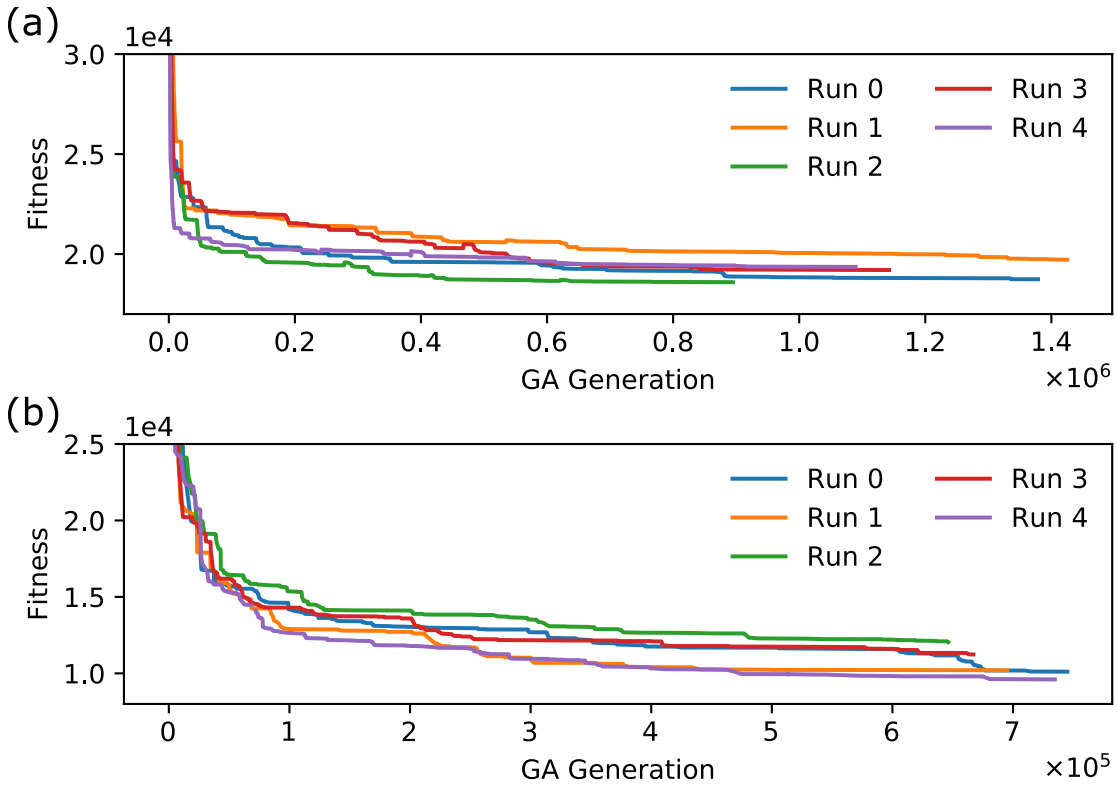


Figure 4.1 **Fitness of the best performing ReaxFF parameter set during optimization.** Results obtained from using OGOLEM are presented by run number for using the (a) QEq and (b) ACKS2 charge models.

data according to the fitness score and showed good stability in brief energy minimization runs was selected for use in the fracture simulations.

4.3 Results and Discussion

In this section, we present the fitting results using the ACKS2 in Reax force field (denoted as ACKS2 FF) and the MD simulation results for Li_2O fracture. There already is a parameter set for ReaxFF from 2016 to study Li-O systems [70], which uses the EEM/QEq model for charge equilibration. We use this parameter set, denoted as "2016 QEq FF" in what follows, as our comparison point. The results from these two force fields are compared.

4.3.1 Fitness to the Training Data Set

The optimized ReaxFF with the ACKS2 model significantly improves the accuracy of the atomic charge assignment when compared against the MRCI training data. Fig. 4.2 shows the predicted atomic charges in the Li−O−Li molecule during stretching. The ACKS2 FF shows an important qualitative improvement over the 2016 QEq FF, as the latter leaves a large residual charge on the mobile Li even after bond-breaking. ACKS2 FF, on the other hand, correctly predicts it to be neutral at a sufficiently large separation. In the bottom figure of Fig. 4.2, when both Li atoms are slowly pulled away from the central O atom, charges on Li atoms decrease as expected. The MRCI method can correctly predict the neutrality of Li atoms when they are more than 4 Å away from O. But DFT still predicts Li and O are +0.2 and −0.4 charged after their distance is beyond 8 Å. This result shows that it is necessary to use more accurate quantum-mechanical methods to predict charges during the bond breaking process. The 2016 QEq FF also significantly under-predicted the Li−O−Li molecule dissociation energy compared to the MRCI+Q results, as shown in Fig. 4.3. This has been corrected in our new force field. We achieve much improved agreement particularly for the "asymmetric stretching" mode in which one Li−O bond is kept fixed while the other Li atom is removed.

Significant improvement is also found in the description of the Li_2O and Li_2O_2 crystal energies, as shown in Fig. 4.3. The 2016 QEq FF significantly over-predicted the formation energy of Li_2O (as given by Eq. (4.1)) in addition to the incorrect prediction of the overall shape of the energy vs. strain curve. Li_2O_2 is even more poorly described, with the 2016 QEq FF failing to even recognize Li_2O_2 as a local minimum in energy. Table 4.1 summarizes the basic materials properties predicted by the ACKS2 and the 2016 QEq FFs in comparison with experiments and DFT data for the crystal phases. ACKS2 FF indeed shows improvement on lattice parameters, formation energies, and bulk modulus, especially for the oxide phases, as these properties are embedded in the energies in the training data. The cohesive energies were not included in the training set for the ACKS2 FF but were fitted by 2016 QEq FF, which

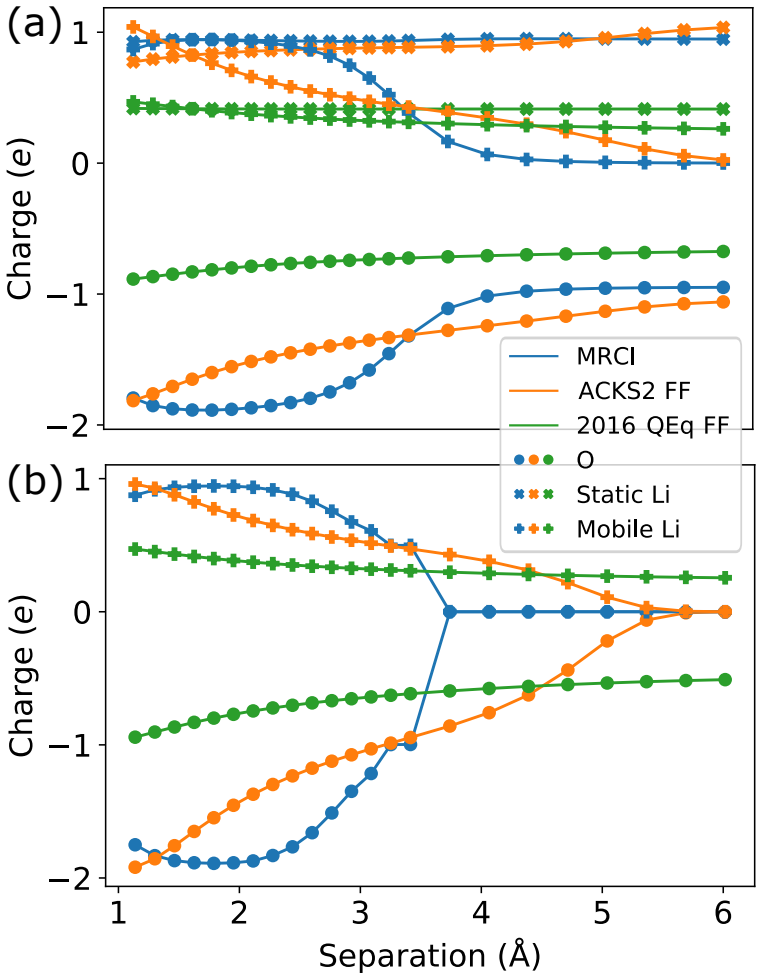


Figure 4.2 **Charge during dissociation of the Li−O−Li molecule for asymmetric symmetric bond stretching.** Results are plotted for the MRCI training data, the fitted ACKS2 FF, and the 2016 QEq FF for (a) asymmetric stretching (one Li−O bond remains static) and (b) symmetric stretching (O remains static).

shows slightly better performance. The elastic constants were not included in the training data. ACKS2 FF is softer for Li than 2016 QEq FF. The 2016 QEq predicted two shear instabilities ($C_{44} < 0$ and $C_{11} - C_{12} < 0$) and the ACKS2 FF showed one shear instability ($C_{11} - C_{12} < 0$). This can be potentially problematic, as they suggest that deformation may lead to phase instability, which is not the case experimentally. The fracture energy was not included in the training set either. For fracturing along $\langle 111 \rangle$ direction, the force fields are very close to one another and significantly over-predict the DFT value. By contrast, both force fields significantly under-predict the DFT fracture energy along the $\langle 110 \rangle$ direction, but

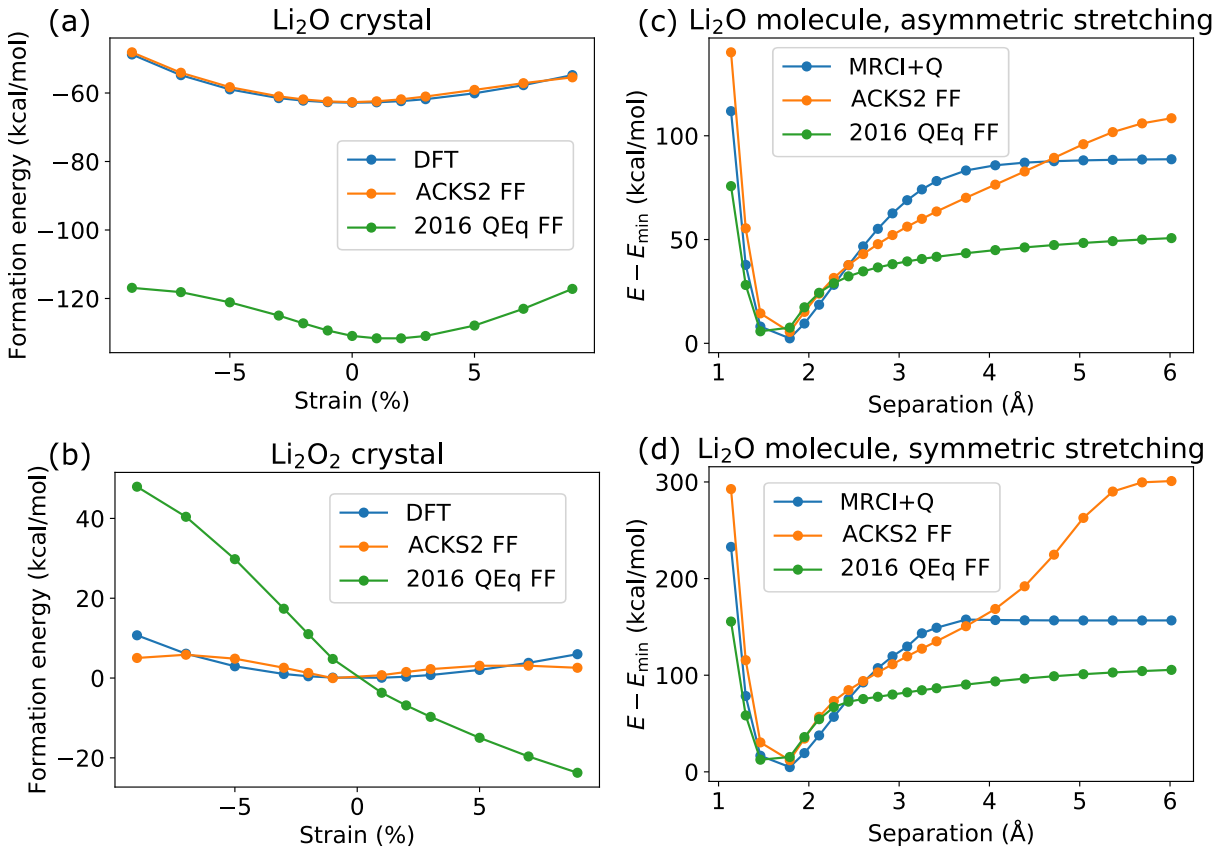


Figure 4.3 **Energy vs. hydrostatic strain of Li_2O and Li_2O_2 crystals along with energy vs. separation of the Li_2O molecule for bond stretching.** Energy vs. hydrostatic strain of (a) Li_2O and (b) Li_2O_2 crystals, plotted for the DFT training data, the fitted ACKS2 FF, and the 2016 QEq FF. (a) shows the Li_2O formation energy (given by Eq. (4.1)) and (b) shows the Li_2O_2 energy referenced to the DFT-optimized structure. Energy vs. separation of the Li_2O molecule for (c) asymmetric stretching (one Li−O bond fixed) and (d) symmetric stretching. Plotted for the MRCI+Q training data, the ACKS2 FF, and the 2016 QEq FF.

the ACKS2 FF shows marked improvement over the 2016 QEq FF. These results suggest that it is difficult for ReaxFF to predict the large variation in surface energies among crystal facets. In the future, a larger number of amorphous structures, surfaces, and deformed lattices could be included in the training set, potentially improving the performance of the FF on phase stability and mechanical properties. For example, both the ACKS2 FF and the 2016 QEq FF predict that the Li_2O crystal is higher in energy than the amorphous phase, which is not a correct description at room temperature. However, the ACKS2 FF obtained in this work already represents a major improvement over the 2016 QEq FF in this

regard, placing the Li_2O crystal at 16 kcal/mol above the amorphous phase, as opposed to 42 kcal/mol predicted by the 2016 QEq FF. We anticipate that with the future consideration of larger number of amorphous structures in the ACKS2 FF fitting, we should be able to lower the energy of the crystalline phase even further, making the simulations more realistic.

Table 4.1 **Materials properties as predicted by DFT, the fitted ACKS2 FF, and the 2016 QEq FF.** Cohesive energy, elastic constants, and fracture energy, which is defined as twice the surface energy, were not included in the training data.

		Li	Li_2O_2	Li_2O
Fitted Stable Cryst. Phase		Bcc, Im3m	$P6_3/mmc$	$Fm\bar{3}m$
Lattice Param. (Å)	Experiment	$a = 3.49$ [95]	$a = 3.17, c = 7.72$ [96]	$a = 4.57$ [97]
	DFT	$a = 3.43$	$a = 3.18, c = 7.70$	$a = 4.66$
	ACKS2 FF	$a = 3.44$	$a = 3.15$ (a/c fixed)	$a = 4.63$
	2016 QEq FF	$a = 3.44$	not stable	$a = 4.65$
Form. Energy (kJ/mol) (Eq. (4.1))	DFT	-	-	263
	ACKS2 FF	-	-	263
	2016 QEq FF	-	-	547
Cohesive Energy (kJ/mol)	Experiment	158 [95]	-	
	DFT	153	1515	1173
	ACKS2 FF	99.7	2583	1654
	2016 QEq FF	157.7	832	1121
Bulk Modulus (GPa)	Experiment	11.6 [95]	-	88.0 [97]
	DFT	12.3	78.7	82.5
	ACKS2 FF	2.55	159	132
	2016 QEq FF	7.30	-	217
Elastic Consts. C_{11}, C_{12}, C_{44} (GPa)	Experiment	14, 11, 9 [95]	-	217, 25, 68 [97]
	DFT	$14, 13, 16$	$153, 51, 37$	$203, 20.3, 53.7$
	ACKS2 FF	$2, 2, 2$	$211, 67, -94$	$143, 185, 33$
	2016 QEq FF	$16, 8, 8$	-	$250, 250, -164$
Fracture Energy (J/m^2)	Surface	$\langle 111 \rangle$	$\langle 110 \rangle$	$\langle 111 \rangle$
	DFT	1.08	1.68	1.07
	ACKS2 FF	3.09	39.3	2.26
	2016 QEq FF	7.78	-2.24	2.31
	Surface	$\langle 110 \rangle$	$\langle 100 \rangle$	$\langle 110 \rangle$
	DFT	1.00	3.55	7.53
	ACKS2 FF	0.70	19.0	3.63
	2016 QEq FF	3.44	-8.23	2.50

4.3.2 Evaluation using Bulk Lithium Oxide

To test the newly fitted ACKS2 FF, a 324-atom cubic supercell of crystalline Li_2O under periodic boundary conditions was simulated within an NPT ensemble at zero pressure and 300K for 40 ps. Tests in the NVE ensemble showed that 0.25 fs time step was required to avoid energy drift, so this time step value was selected and is used consistently throughout the paper. In both force fields, the structure became amorphous, but the volume increased by 6% using the ACKS2 FF while decreasing by 25% using the 2016 QEq FF. Fig. 4.4 shows the evolution of the volume and the resulting radial distribution functions (RDFs), while more detailed RDF evaluations are shown in Fig. B.4. The nearest-neighbor Li-O peak in the ACKS2 FF split into two peaks at 1.8 Å and 2.4 Å, respectively, causing anisotropic mechanical behavior. The 2016 QEq FF showed similar Li-O peak splitting, with a small Li-Li peak appearing close to 1.3 Å, which is much shorter than the Li-Li bond distance in crystal Li (3.0 Å) and Li_2O (2.3 Å). Based on these results, we conclude that the ACKS2 FF equilibrium structure is more realistic and stable than that of 2016 QEq FF.

Subsequent to NPT relaxation of the Li_2O boxes, slabs were then constructed from a $8 \times 1 \times 6$ tiling of the equilibrated boxes, with a 90 Å vacuum space included in the x direction. The slabs were then further equilibrated within the NPT ensemble under 3D periodic boundary conditions at 300K (Fig. 4.5). Equilibration of the ACKS2 FF slab does not have much effect on the amorphous Li_2O other than smoothing out the random distribution of atoms over a large number of particles. The RDF stabilizes after about 10 ps in ACKS2, showing very little subsequent change out to 40 ps. The same process occurs in the first 8 ps of the 2016 QEq FF slab, but then starting around 10 ps the slab shrinks considerably and develops long-range order, as shown by the emergence of distinct peaks in the RDF, reaching a new equilibrium by 40 ps. The order is not commensurate with real crystalline Li_2O and represents an unphysical crystalline phase. This provides further confirmation that the ACKS2 FF predicts a more stable solid Li_2O than the 2016 QEq FF.

In order to test if the improved success of our ACKS2 FF was due to the charge assignment

method and not just the training procedure, a nearly-identical parameter-fitting run was attempted for a ReaxFF force field using QEq for charge assignment. The only difference was that the ACKS2-specific parameters were not allowed to change during optimization, since they are not relevant to a QEq force field. While a comparable fitness value on the training items was achieved as shown in Fig. 4.1, the resulting force fields are unstable when simulating solid Li_2O (see Section B.2). This suggests that QEq-based ReaxFF is incapable of describing the underlying charges properly in the training set, and the genetic algorithm may be forced to "over-fit" the training data with an unphysical parameter combination which leads to an unusable force field. This highlights the need for ACKS2 for the correct description of charges in these materials and the difficulty of using fully automated fitting procedure, as the physics insights are still important.

4.3.3 Evaluation using Deformation and Fracture

In a molecular dynamics simulation, brittle fracture is characterized by the propagation of a sharp crack tip across a sample and a relatively flat fracture surface. To explore brittle fracture, MD simulations were performed on slabs of Li_2O. Atoms were removed from each slab to create a sharp notch in the xy plane, which corresponded to the [100] crystal direction in the original structures. Every picosecond, 0.5% strain was applied normal to the crack plane while the atom positions were evolved under NVT at 300K. This simulates a plane-strain condition. Initially, crystalline slabs were used as the starting point for the deformation simulations. However, the large difference in built-in stresses due to the instability of the crystal structure made it impossible to draw meaningful comparison between the results. Therefore, NPT-equilibrated slabs described in Section 4.3.2 by ACKS2 FF and 2016 QEq FF methods (both with and without long-range order) were selected to test crack propagation. All of these initially have low stress in all three directions, suggesting equilibrium state, and thus represent a fair comparison. Parameters of the slabs at the beginning of the fracture simulations are shown in Table 4.2. Duplicates of each fracture simulation were performed

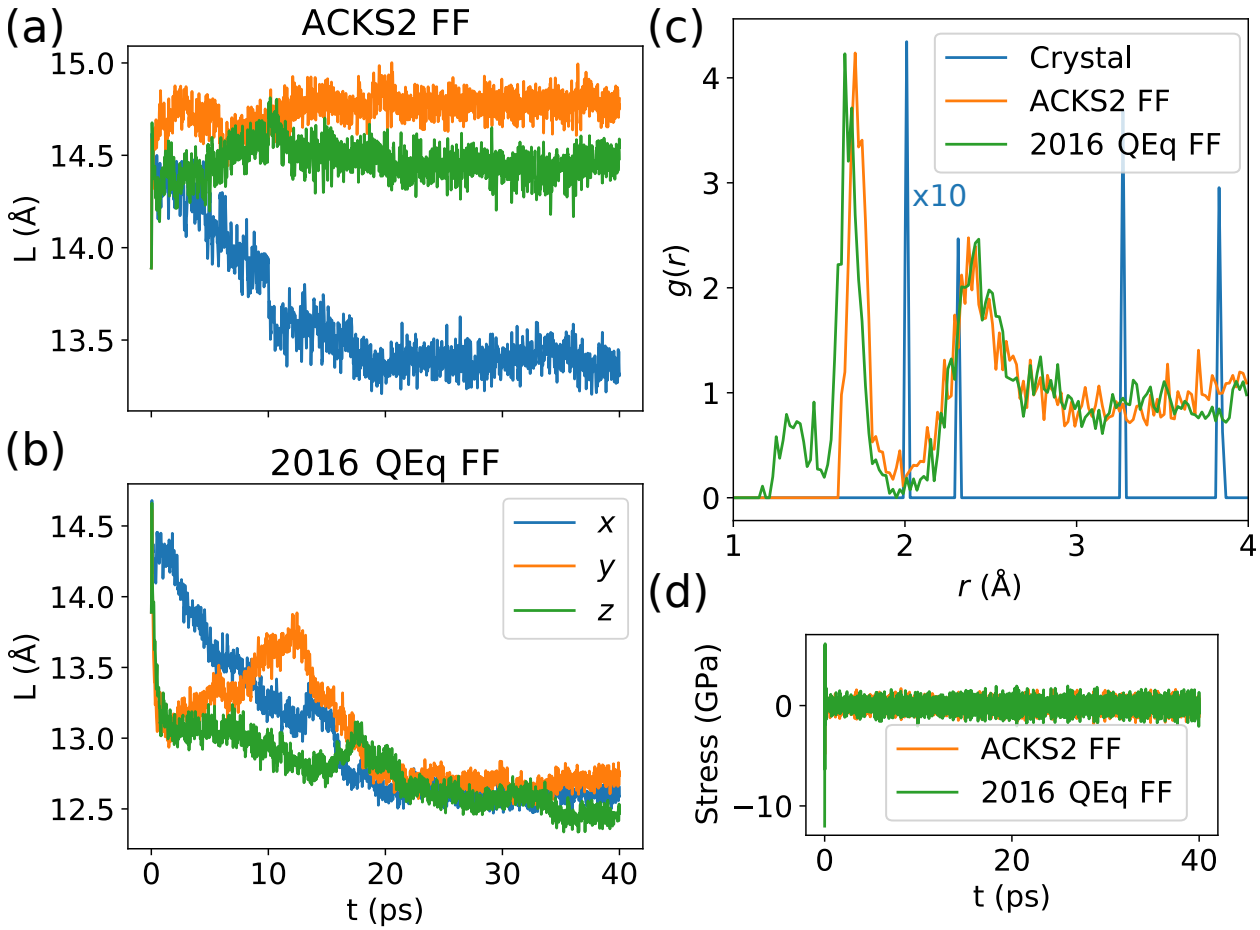


Figure 4.4 **NPT simulation and material properties of crystalline Li_2O with the ACKS2 FF and the 2016 QEq FF.** Simulations were performed for 40 picoseconds at 300K using a 324-atom cube of crystalline Li_2O with the ACKS2 FF and the 2016 QEq FF. Panels (a) and (b) show side lengths of the simulation box (under periodic boundary conditions). Resulting RDFs are shown in (c), along with the crystal RDF (scaled by 0.1 for ease of comparison). The ACKS2 FF equilibration is more realistic, developing a single nearest-neighbor peak closer to the crystal Li-O bond distance, whereas the 2016 QEq FF peak is split and at smaller separation. Panel (d) shows the average stress over the course of the simulation.

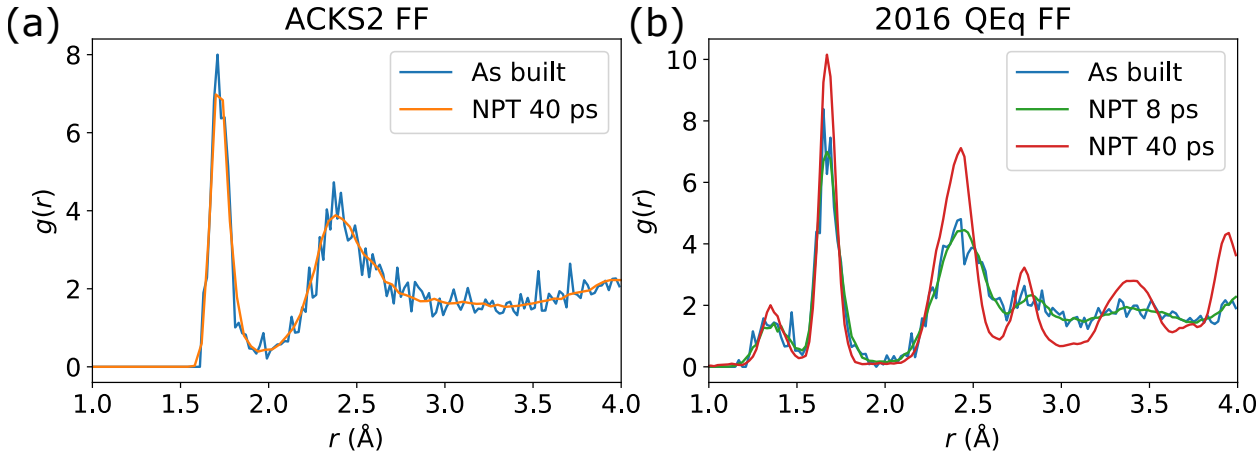


Figure 4.5 **Simulation results and material properties for Li_2O slabs before and after NPT equilibration with ACKS2 and 2016QEq FF.** Results are shown using (a) ACKS2 and (b) 2016QEq FF. Noise is reduced in both simulations due to the large number of particles, but the ACKS2 FF shape remains the same. For the first 8 ps the 2016 QEq FF RDF remains similar, but then it develops long-range order evidenced by the emergence of distinct peaks.

to test the impact of the stochastic MD simulation. The different random starting velocities did not impact the conclusions (see Fig. B.3).

Table 4.2 **Properties of the notched slabs used as a starting point for the fracture simulations.** a is the notch length, b is the slab width, y is the thickness in the periodic direction along the crack, z is the thickness in the periodic direction normal to the crack, h is the space between the crack edge and the periodic boundary, and N is the number of atoms in the slab.

	ACKS2 FF	2016 QEq FF (amorphous)	2016 QEq FF (ordered)
a (Å)	49.7	46.0	46.2
b (Å)	108.3	101.2	97.5
y (Å)	14.8	12.7	12.0
z (Å)	86.6	74.6	71.7
h (Å)	32.5	27.9	27.5
N	14700	14624	14654

Stress-strain curves and visualizations of the slab undergoing crack propagation are shown in Fig. 4.6. Stress was calculated by taking the average pressure over each picosecond of constant strain. Pressure was converted to true stress by multiplying by V/V_{occ} where V is the total volume of the simulation cell and V_{occ} is volume occupied by the slab at each time

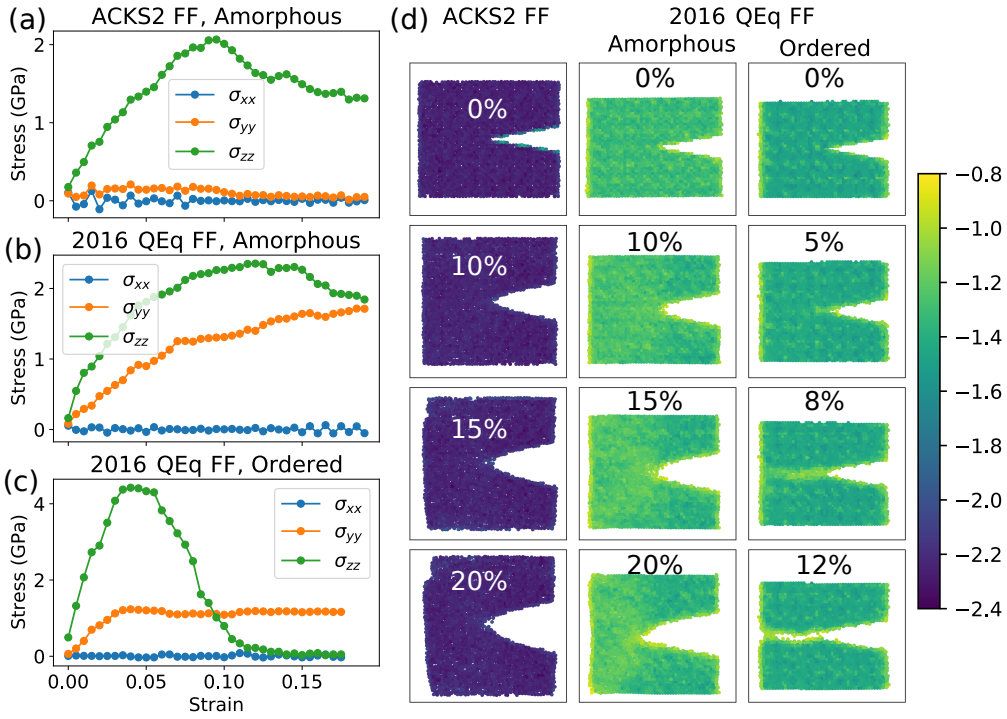


Figure 4.6 **Simulation results for fracture in Li_2O slabs.** After NPT equilibration, a notch is cut in the z-normal plane in each slab, and strain is applied along z constant engineering strain rate of 5×10^{-3}ps^{-1} up to 20% strain to simulate fracture. Panels (a) to (c) show the resulting stress-strain curves using the ACKS2 FF equilibrated slab and two 2016 QEq FF equilibrated slabs: one amorphous, one ordered. The amorphous ACKS2 FF slab does not show complete fracture, but it does show a well-defined peak around 9% strain and stress relaxation as the crack propagates. The 2016 QEq FF amorphous slab shows more ductile behavior, with a broad peak around 13% strain and more gradual stress relaxation. The 2016 QEq FF ordered slab shows classic fracture behavior, with a sharp peak around 5% strain, with the stress falling to zero by about 12% strain. This cracking behavior is likely assisted by the increased density and long-ranged order. Panel (d) shows the shape of the slabs during deformation through scatter plots of the oxygen charges in slabs under various levels of strain. The ACKS2 FF slab retains a sharper crack tip than the amorphous 2016 QEq FF slab, and the oxygen charge remains close to the formal charge of -2 throughout the slab. Both 2016 QEq FF slabs show oxygen charge significantly smaller than the formal charge, with noticeable charge depletion around the crack tip and fracture surfaces.

step. The stress in the x direction remains zero due to the free surface. The stress in the y direction varies due to the different Poisson's ratios. According to the elastic constants for Li_2O shown in Table 4.1, the 2016 QEq FF over-predicts the Poisson's ratio while the ACKS2 FF under-predicts the Poisson's ratio compared to the DFT results. As expected based on its performance on the training data, the newly optimized ACKS2 FF parameter set shows

improvement in the description of fracture in amorphous Li_2O. The ACKS2 FF stress-strain curves show crack propagation events in which the stress along z direction drops sharply at 10% of strain, while the 2016 QEq FF shows more smooth, ductile behavior. However, neither force field produces true brittle fracture in amorphous slabs, although the stress intensity factor is less than 1 $\text{MPa} \cdot \text{m}^{1/2}$ (calculated based on the peak stress [98] and the geometry of the notched slab), due to the small crack length used in MD simulations. Interestingly, the 2016 QEq FF slab with long-range order appears to show classic brittle fracture at 5% strain. However there is local pre-fracture melting near the crack tip, which is not likely to occur in room temperature brittle fracture. So the simulated fracture is likely due to unphysical mechanisms. This suggests that, while the force field used is important, the overall fracture propagation is a much more complicated process that may be related to some non-fitted features in the force field. The current force fields have deficiencies in describing some elastic and fracture properties of the lithium oxides. The phase change during relaxation and deformation is related to the over-stabilization of the amorphous structure and the mechanical instability shown in Table 4.1. For future improvement, the training set should include crystalline structures with a wider variety of elastic deformations in addition to unstable amorphous structures. However, since the energy differences between elastically deformed structures are relatively small ($\pm$5 kcal/mol) compared to the formation energy of lithium oxides ($\sim$ 260 kcal/mol), the weights for fitting have to be carefully picked.

The improvement in fracture behavior on amorphous slabs may once again be attributed to an improvement in the charge assignment due to ACKS2. The oxygen charges are shown in Fig. 4.6 (d). The magnitude of both oxygen and lithium charges are significantly larger in the ACKS2 FF and closer to the formal charges of Li^+ and O^{2-}. The difference between atomic charges in surface and in bulk is improved in the ACKS2 FF. There is only a modest reduction in magnitude of the atomic charge at the surfaces in ACKS2 FF. These ACKS2 FF results are consistent with Bader charge analysis [99] of DFT results on Li_2O surfaces, which show that atom charges at the most favorable (111) surface are essentially unchanged

from their bulk values.

By contrast, the (already small) oxygen charge in the QEq FF is significantly depleted at surfaces of the crack, to a greater degree than at the slab surface. This is especially pronounced at the crack tip, suggesting that the depletion arises from QEq's incomplete charge transfer upon bond breaking. Further details on the charges may be found in Fig. B.2.

We expect that in the future, if an ACKS2-based force field can be trained to produce a stable ordered structure, it will show similar or better brittle fracture behavior to that exhibited by the ordered 2016 QEq FF slab. This will be particularly valuable if the crystal structure matches the experimental Li_2O structure, as this could allow for realistic large-scale MD simulations of lithium oxidation and cracking.

4.4 Conclusion

Both ionic and covalent bond breaking are individually challenging in a MD context, and simulating fracture in ceramics requires an accurate description of both phenomena simultaneously. Our results represent a significant step forward in the atomistic simulation of bond breaking in crystalline systems with mixed ionic-covalent bonding character. We show that a Reax force field employing the high-fidelity ACKS2 method for charge assignment, trained using a GA with training data from high-level *ab initio* wave function quantum chemistry calculations, is able to describe bond-breaking in Li−O−Li molecules. When comparing fracture on stable amorphous Li_2O, the ACKS2 force field represents a modest improvement in fracture behavior compared to the existing 2016 QEq force field.

Future work may be able to improve fracture behavior in Li_2O by training an ACKS2-empowered ReaxFF with a genetic algorithm, while also including a large sample of amorphous structures in the training set. This may train the force field to predict the correct crystal structure of Li_2O rather than amorphization. The improvement in bond-breaking and charge assignment demonstrated by ACKS2 may then lead to true brittle fracture.

The insights gained in this study, our GA-based fitting method, and our training set de-

sign strategies can easily be extended to other systems. They may also serve as a foundation for further automation of force-field generation and fitting [92]. Use of ACKS2 is also likely to improve the description of other systems, especially ceramics or other materials with a high degree of ionic bonding.

CHAPTER 5

PERFORMANCE OPTIMIZATION OF LARGE-SCALE DISTRIBUTED GPU-ACCELERATED REAXFF SIMULATIONS

5.1 Introduction

The original MPI+CUDA implementation of PuReMD was authored when the Fermi and Keplar microarchitectures from the Tesla line of NVIDIA GPUs were in their heyday [94]. Since this time, subsequent hardware revisions have drastically improved peak computational and memory bandwidth. These advancements have highlighted the need for revision and enhancement of ReaxFF implementations.

5.2 Methods

The following subsections discuss several optimizations employed in the MPI+CUDA codebase for PuReMD. These efforts range from generic best practices software engineering approaches to tuning individual kernels for lower resource usage and high parallelizability and to implementing several of the previously discussed charge solver optimizations.

5.2.1 Improving Performance Portability and Scalability via Software Modernization

As previously mentioned, the original MPI+CUDA implementation of PuReMD was developed on the Fermi and Kepler microarchitectures of NVIDIA Tesla GPUs [94]. While this work made great strides in improving performance, numerous limitations to this implementation existed including hand-optimized kernels solely for Kepler GPUs (due to hardware assumptions); fixed-size memory allocations which severely limited scalability to systems with at most a few thousand atoms; low parallelizablity resulting in poor device utilization due to assigning one CUDA thread to compute all interactions for an atom; excessive data transfers the between host and device memory spaces; and lack of implementation of critical

simulation features (e.g., non-NVE ensembles and additional charge models).

To address these shortcomings, the MPI+CUDA codebase was revised and improved using several approaches. To address issues with microarchitecture-specific optimizations, several kernels and small sub-tasks within kernels were replaced with performant portable alternatives within the NVIDIA CUB library [100]. CUB provides optimized implementations of algorithms and performance primitives at varying scopes – device-wide, block-wide, warp-wide – across various NVIDIA GPU microarchitectures which have been used across several scientific and numeric codes [101, 102]. For the PuReMD codebase, several CUB routines were leveraged including reductions and prefix sums.

In order to resolve issues with fixed-size memory allocations, a robust approach was required as the number of interactions per atoms may vary significantly across simulation steps for different terms in the ReaxFF potential. Previous iterations of PuReMD addressed this issue with ReaxFF by adopting a three-stage approach to managing memory: 1) estimate the number of interactions per atom, 2) check current data structures allocations and reallocate more memory if needed, and 3) perform computations for an interaction. While this approach is simple and has reasonable performance in a traditional CPU code, several downsides become apparent when moving to a programming model where most of the computation resides on the GPU. Namely, estimation of interaction counts requires expensive device-wide reductions at every simulation step. Moreover, following these estimations, data must be transferred back to host memory as the host context drivers memory management for the majority of the device memory space. In order to avoid these issues, an atomic transactional approach for dynamic memory reallocation was developed in Algorithm 5.1.

Algorithm 5.1 relies on writing kernels where the computations may be rolled back if an out-of-memory condition is detected. If this atomicity condition is satisfied, then the benefit of avoiding steps within the conditional clause can be gained on the majority of simulation steps if either the system under simulation does not have rapidly changing memory requirements or sufficiently large over-allocations are performed when reallocation steps

Algorithm 5.1 Atomic Transactional Approach for GPU Memory Management.

```
1: Perform computation and note any out-of-memory conditions (e.g., compute bond list)
2: Copy out-of-memory status from computation to host
3: if Out-of-memory condition detected during computation then
4:     Revert any state altered by previous computation
5:     Estimate new storage requirements for computation
6:     Copy storage requirements to host
7:     Reallocation storage space from host
8:     Redo computation
9: end if
```

occur. In practice, a 20% to 40% increase above the current estimation is often sufficient to significantly reduce the number of reallocation steps. Furthermore, refining the scoping of reallocations in data structures from a global scope to a more localized scope may further reduce the frequency of triggering reallocations – a prime example of this is in the interaction lists where changing from global counts across all atoms to localized counts per atom dramatically drops the number of reallocations.

5.2.2 Increasing Parallelism through Kernel-Level Restructuring

In order to fully utilize the thousands of CUDA cores in modern NVIDIA GPUs, additional programming changes are required to achieve degrees of finer-grain parallelism. A core mechanism where a higher degree of parallelism can be expressed is the interaction list data structure. The interaction list in PuReMD is effectively a list of lists where the inner lists are flattened and stored contiguously as a 1-D array (with inter-list padding for memory alignment), with supporting arrays for storing the starting and ending indices of each inner list within the flatten 1-D array. Conceptually, the outer index used to select a specific inner list corresponds to some entity of interaction such as an atom or a group of atoms interacting in some physical manner. In terms of parallelism, the original MPI+CUDA code utilized 1 CUDA thread to process all the valid interactions for a inner list. As the inner lists can vary significantly in terms of size and as many of the computations performed for each entry in an inner list possess a relatively high arithmetic intensity, the introduction of additional

parallelism in processing the inner lists leads to significant gains. Toward this end, a group of threads (e.g., a warp of 32 threads) was utilized to process the inner lists.

The group of threads approach has several advantages including enabling kernels to use fast collective operations – reductions, scans, broadcasts, atomic operations with leader election, and synchronizations – and performing fast memory accesses through memory coalescing via aligned burst reads and writes to aligned sections of the inner lists, at the cost of increased memory usage due to this padding. Additionally, when combining delayed writes to global memory, use of fast shared memory, and reductions of partial results at the warp-level, memory pressure decreases with respect to the use of atomic operations in kernels for, e.g., partial energy and force calculations.

5.2.3 Exploiting Task-Level Parallelism via CUDA Streams

Moving beyond the idea of exposing more parallelism, another approach which is broadly applicable to the PuReMD MPI+CUDA codebase is to leverage task-level parallelism within the ReaxFF potential and the coupled global charge model. Conceptually, the use of multiple simultaneous CUDA streams provides the CUDA warp scheduler more opportunity to hide memory access latency as well as to increase device utilization by scheduling computation on idling streaming multiprocessors. With this in mind, Fig. 5.1 depicts the task dependency graph in ReaxFF. In this graph, there are two points with a number of paths with independent tasks following Verlet list generation/updating: 1) initialization post Verlet list generation, and 2) energy and force computation for the potential terms in ReaxFF. In the former section, three independent paths are present (bond, H-bond, charge matrix initialization) while in the latter section six independent paths exist (QEq/Coulomb, van der Waals, BO/Bonds, BO/OverUnder, BO/Valence/Torsion, H-Bonds). As such, the updated codebase employed the use of 6 CUDA streams.

In order to facilitate the use of CUDA streams in PuReMD, several changes were required including using atomic operations to ensure true data independence between tasks for in-

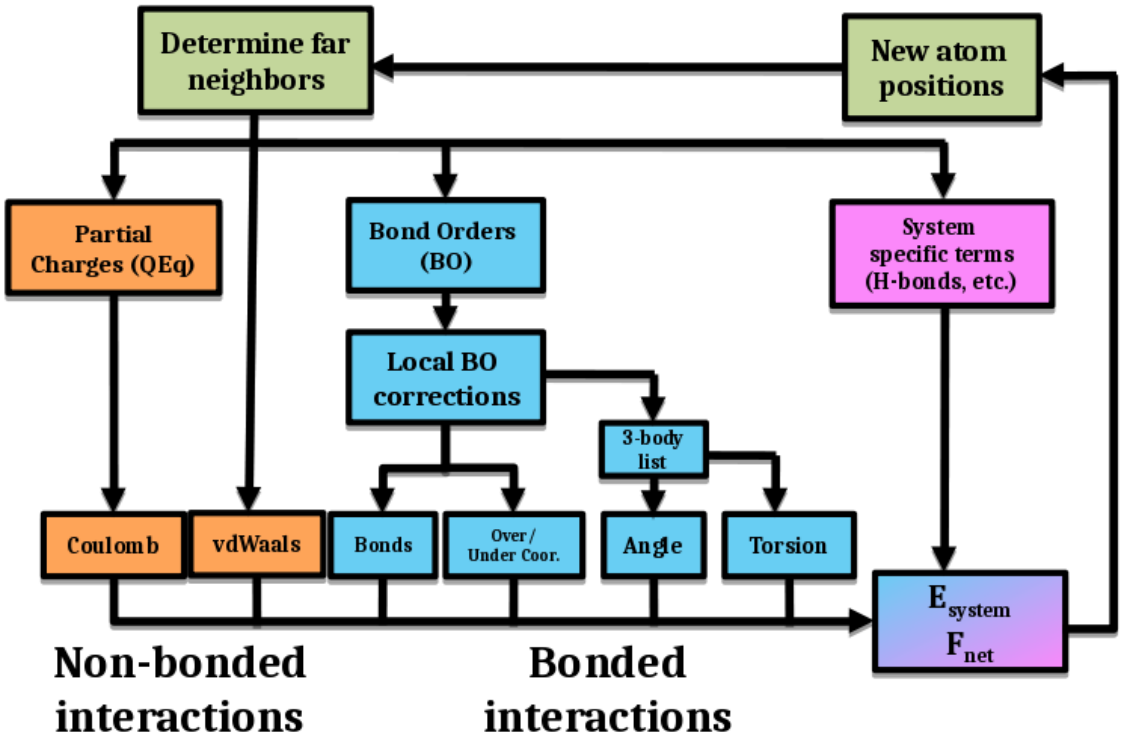


Figure 5.1 **Task dependency graph for the ReaxFF potential coupled with a global charge model (QEq) within a single MD step.**

termediate results, switching from synchronous to asynchronous data transfers between the host and device memory spaces, and injecting CUDA events between asynchronous kernel launches in order to translate dependencies in the task graph into dependencies between kernels executing in different streams. Further to this last point on CUDA events – synchronization of tasks across different streams is accomplished via the queuing of and waiting on CUDA events. As to data movements in the context of multiple CUDA streams, the synchronous CUDA API functions all occur in the default stream and thus cannot execute concurrently with non-default streams; thus, the switch to the asynchronous functions is required and yields additional performance gains due to decreases in synchronization overheads.

5.2.4 Global Charge Solver Optimizations

Another more targeted area for optimization in the MPI+CUDA code is the global charge solver models. As previously discussed concerning optimizations for the shared-memory and distributed-memory MPI-only versions of PuReMD, the sparse linear solvers underpinning the global charge models constitute a significant portion of the total execution time per MD step within ReaxFF+QEq. In order to better understand the need for optimization of the charge solver, Fig. 5.2 presents kernel execution from a profiling run of the MPI+CUDA code for a typical MD step (as delimited by the green vertical bars). As shown, the charge solver starts to run about mid-way through the total step time, and once the charge solver begins execution, the GPU utilization drops due low arithmetic intensity of the kernels in the solver, especially the SpMV kernel. Also, due to host-device data transfers and the MPI communications, several gaps occur in the execution timeline where the streaming multiprocessors on the GPU sit idle. As previously discussed, multiple CUDA streams may be used to hide the data transfer latency, but other optimizations are required to further decrease charge solver runtime. These methods are discussed in detail below.

Approaches for improving the charge solver performance include optimizing significant kernels within the iterative solver algorithm, namely the sparse-matrix-dense-vector multiplications (SpMV's); overlapping the charge solver with other kernels via multiple streams as discussed in the previous section; and decreasing solver runtime through efficient preconditioning. For kernel-level tuning, warps of threads were utilized for the SpMV's along with symmetric charge matrices being stored in the full format. As to the use of CUDA streams, the approach followed the guidance outlined in the previous section, with the additional insight that if one host thread is used to manage the CUDA context then the charge solver should be launched last in order to avoid the blocking other kernel launches in different streams due to waits on memory transfers and MPI communication routines. Lastly, insights from the preconditioning methods explored in the distributed-memory MPI-only section guided implementing SAI-based preconditioning. For this work, computation of the

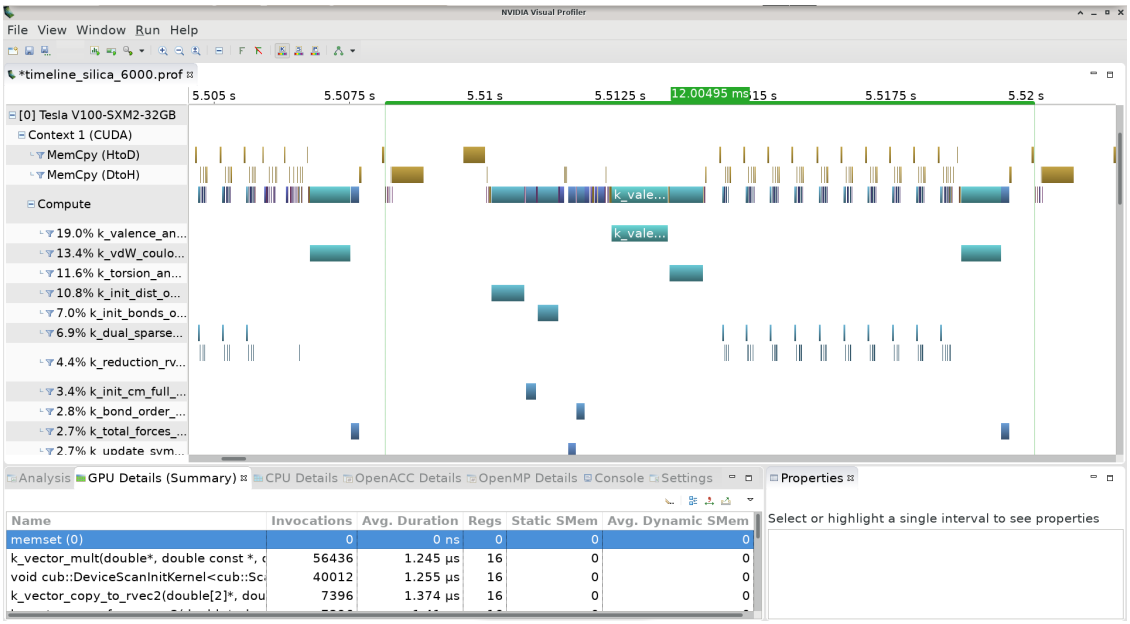


Figure 5.2 **Execution profile of the MPI+CUDA version of PuReMD using a single CUDA stream.** The nvprof profile was collected on a single V100 GPU using a 6000 atom amorphous silica system running for 10K steps with 0.25 fs step sizes and an NVE ensemble. Individual kernels, CUDA API calls, and host functions are visualalized horizontally in the center, while cumulative runtime percentages are summarized in the left-most column.

SAI preconditioner was performed on the host using multiple OpenMP threads due to the current lack of optimized batched least-squares solvers in the CUDA ecosystem – it is left as an area of future work to move this computation to the device.

5.3 Performance Studies

5.3.1 Computing Environment and Benchmark Systems

Results from numerical experiments presented in the following subsections were collected on clusters at the High Performance Computing Center at Michigan State University. Subsets of nodes in these clusters are equipped with NVIDIA Tesla GPUs of varying models and multiplicities: K20's, K80's, V100's, and A100's. For these experiments, we focus on nodes equipped with V100's and A100's. Targeted nodes equipped with V100's contained 8 GPUs and 40 CPU cores – two twenty-core Intel Xeon Gold 6148 Skylake 2.4 GHz processors and 384 GB main memory. Also, nodes equipped with A100's contained 4 GPUs and 64

GPU cores – two thirty-two core Intel Xeon Platinum 8358 Ice Lake 2.6 GHz processors and 256 GB main memory.

At the time of the experiments, the clusters ran version 7.9.2009 of the CentOS distribution of GNU/linux for x86_64 architectures, kernel version 3.10.0-1160.36.2, and glibc version 2.17-325.

For software, the MPI+CUDA version of PuReMD was built using the GNU Compiler Collection version 10.3.0, OpenMPI version 4.1.1, and CUDA version 11.4.2 bundled with CUB version 1.12.1. Compiler optimizations enabled included those implied by the `-O3` flag and those enabled by optimized device code generation for the Volta and Ampere microarchitectures. For numeric libraries, Intel MKL version 2021.2.0 was utilized. For SAI-based precondition computation, the number of OpenMP threads was set to the number of available CPU cores (no hyperthreading enabled).

For benchmarking purposes, two familiar molecular systems were selected: bulk water (H_2O) and amorphous silica (SiO_2). The sizes of these systems varied from hundreds to thousands of atoms depending on the experiment.

5.3.2 Scalability Studies

To better analyze the impact of GPU acceleration over execution time and scaling, we conducted weak and strong scaling experiments for mean simulation time per step for PuReMD and LAMMPS Kokkos, a high performance ReaxFF implementation maintained by Scandia National Laboratories. As indicated earlier, the current PuReMD implementation relies upon a uniform domain decomposition of the simulation space among the computing resources, with MPI processes and GPUs in a one-to-one mapping. Therefore for our weak scaling experiments, we choose water and silica systems with simulation volumes of moderate size which contain several thousand atoms, and scale it up number of GPUs in powers of 2. As given in Fig. 5.3, results for the silica systems on the V100's with PuReMD CG+SAI(0.15) yield speedups of between 1.2x to 1.26x over PuReMD CG+Jacobi, and be-

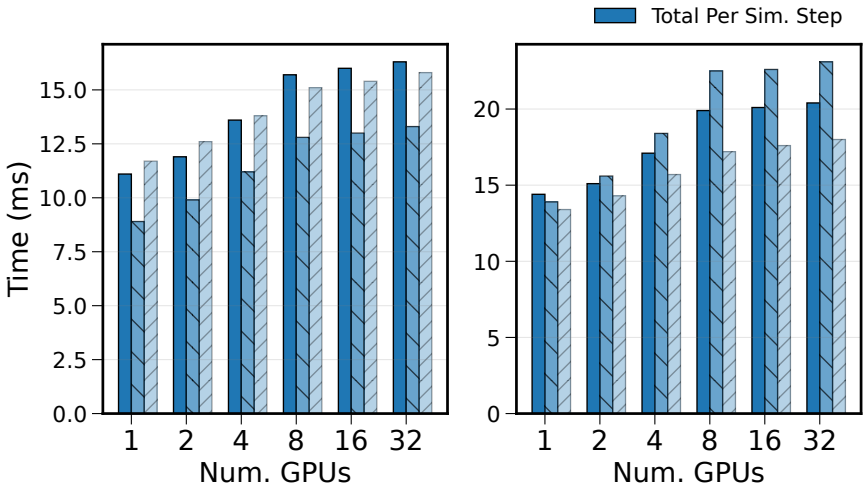
tween 1.12x to 1.31x over LAMMPS Kokkos. For the water systems, speedups for PuReMD CG+SAI(0.15) range from 0.88x to 1.04x over PuReMD CG+Jacobi and from 0.76x to 0.97x over LAMMPS Kokkos. Similar to the V100's, results for the silica systems on the A100's with PuReMD CG+SAI(0.15) yield speedups of between 1.14x to 1.37x over PuReMD CG+Jacobi, and between 1.17x to 1.54x over LAMMPS Kokkos. For the water systems, PuReMD CG+SAI(0.15) fairs better with speedups ranging from 1.02x to 1.08x over PuReMD CG+Jacobi and from 1.05x to 1.24x over LAMMPS Kokkos. Worth noting, gains from SAI preconditioning due to the reduction in iteration counts are somewhat eroded by the relatively large precondition computation cost due to host-side computation and subsequent transfer to device memory. Particularly, as the number of GPUs increases, the number of MPI threads per MPI process decreases, hence the preconditioner computation cost increases.

In Fig. 5.4, we present the strong scaling results mean per-step simulation time. Since QEq solves constitute a significant amount of the total running time, improvements obtained for from SAI preconditioning carry their benefits to the overall simulation times. In our experiments, we observed overall speedups of 0.81x to 1.74x and 0.89x to 1.77x for PuReMD CG+SAI(0.15) over LAMMPS Kokkos on the V100's with the silica and water systems respectively, and 0.79x to 1.17x and 0.82x to 1.57x for silica and water on the A100's. Interestingly, the LAMMPS Kokkos code fairs best with lower GPU counts where the number of atoms per GPU is greatest. Thus, additional tuning on PuReMD may be required, especially if performance can be gained from avoiding memory transfers between host and device memory spaces during the charge solve kernels.

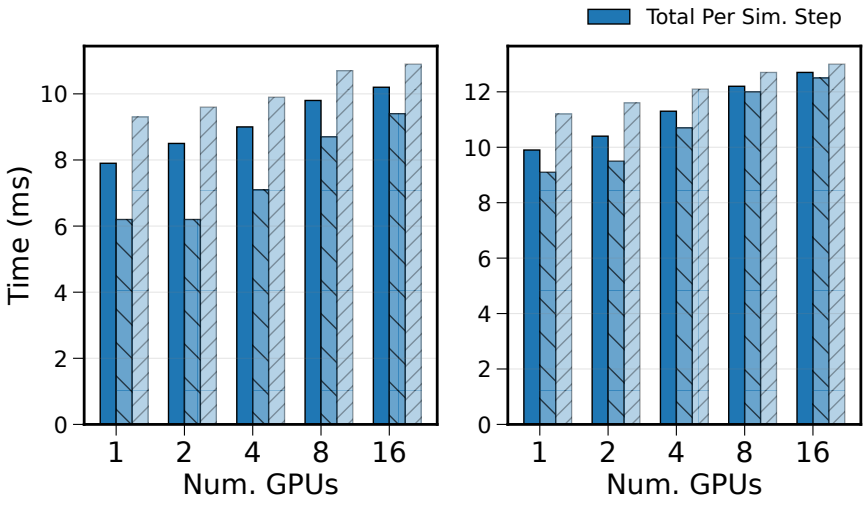
5.4 Conclusions

We explored several optimizations to improve the performance of GPU-accelerated distributed-memory implementation of ReaxFF+QEq in PuReMD. By applying techniques including leveraging performance portable primitives in code libraries, exploiting higher degrees of

parallelizablity through warp-tuned kernels, hiding memory access latency and increasing device utilization through CUDA streams, and improving charge solver performance through tuning and improved preconditioning, the updated MPI+CUDA version of PuReMD stands to enable efficient large-scale scientific studies on modern supercomputers. To confirm these assertions, we performed scaling experiments against the LAMMPS Kokkos codebase, an efficient ReaxFF implementation developed by Scandia National Laboratories. In several cases, PuReMD performance is near parity or above the LAMMPS codebase, with greater performance tending to be achieved at larger scales, thus showing great promise for the future use of PuReMD.

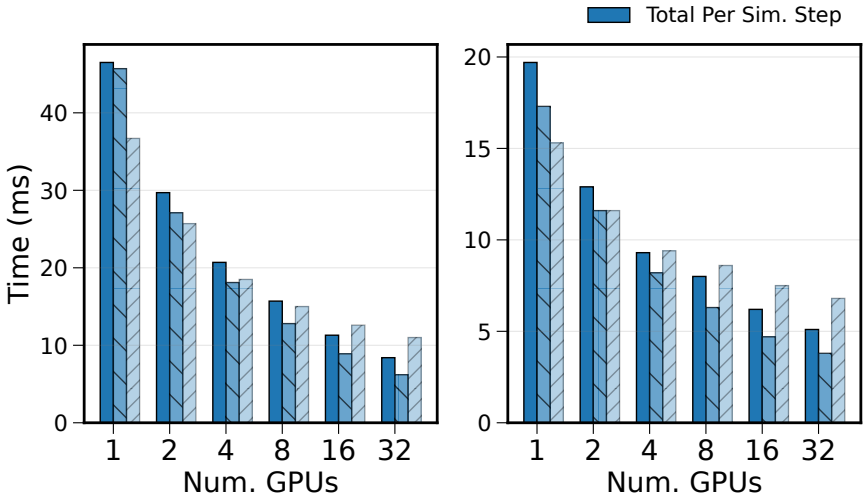


(a) Volta V100's

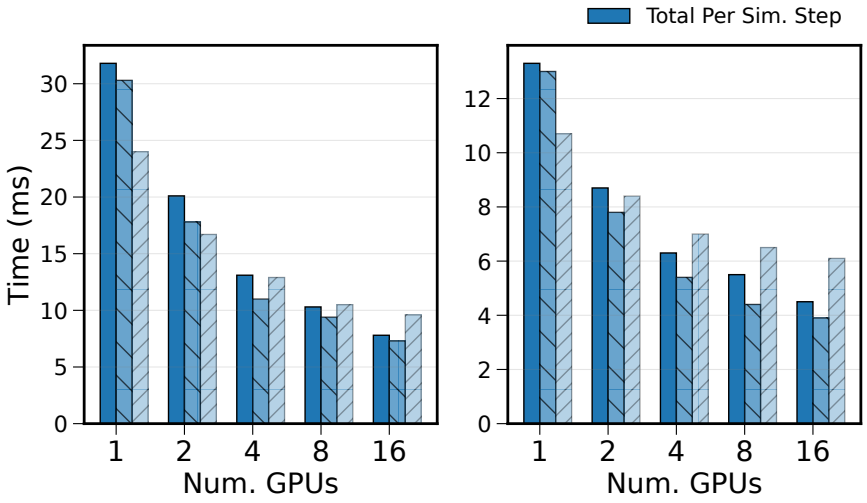


(b) Ampere A100's

Figure 5.3 **Weak scaling study for the MPI+CUDA PuReMD implementation and LAMMPS Kokkos.** Simulations coupled with the QEq charge model were performed on servers with (a) 8 Volta V100 and (b) 4 Ampere A100 GPUs at the MSU HPCC. Within each group of plots above, the left plots correspond to amorphous silica systems (6000 atoms per GPU) and the right plots correspond to bulk water systems (6540 atoms per GPU). PuReMD results correspond to the left two bars in a group (CG+Jacobi, CG+SAI(0.15) solvers, respectively) while LAMMPS Kokkos (CG+Jacobi) results correspond to the transparent, hatched bars. Reported times are mean total times per MD step. For these results, simulations were performed using an NVE ensemble for 10K steps with 0.25 fs per step. Reconstruction of the Verlet neighbor list was done every 25 steps using a 2.0 Å Verlet list buffer for the codes using CG+Jacobi solvers, while for the CG+SAI(0.15) solver reconstruction occurred every 250 steps using a 3.0 Å buffer. All codes used a 10^{-6} solver tolerance.



(a) Volta V100's



(b) Ampere A100's

Figure 5.4 **Strong scaling study for the MPI+CUDA PuReMD implementation and LAMMPS Kokkos.** Simulations were performed on servers with (a) 8 Volta V100 and (b) 4 Ampere A100 GPUs. Within each group of plots above, the left plots correspond to amorphous silica systems (48000 atoms) and the right plots correspond to bulk water (22500 atoms). PuReMD results correspond to the left two bars in a group (CG+Jacobi, CG+SAI(0.15) solvers, respectively) while LAMMPS Kokkos (CG+Jacobi) results correspond to the rightmost bars. Reported times are mean total times per MD step. Additional simulation parameters were identical to those used in the weak scaling study.

APPENDICES

APPENDIX A

SUPPLEMENTAL DATA FOR GLOBAL CHARGE MODEL OPTIMIZATIONS IN SHARED-MEMORY

Table A.1 **Force field parameter values utlized for simulations with the molecular systems in Table 2.1.**

System	Element	γ_i	χ_i	η_i	σ_i	Λ
Bilayer	C	0.7631	5.9993	6.0000	-	-
&	H	0.8203	3.7248	9.6093	-	-
Silica	N	1.0000	6.8287	7.2217	-	-
	O	1.0898	8.5000	8.3122	-	-
	P	1.0000	1.8292	7.2520	-	-
	Si	0.8925	4.6988	6.0000	-	-
PETN	C	0.8712	5.7254	6.9235	-	-
	H	0.8910	3.8446	1.0698	-	-
	N	0.8922	6.7424	6.2435	-	-
	O	0.8712	8.5000	7.1412	-	-
Water	H	0.8203	3.7248	9.6093	3.4114	-
	O	1.0898	8.5000	8.3122	0.9745	-
	-	-	-	-	-	548.6451

Table A.2 **Charge matrix condition numbers for the charge distribution models.** Condition numbers derived from the eigen-spectrum computed by the `LAPACKE_dgesvd` function with Intel Math Kernel Library (MKL) version 2018.1.163.

System	Method	$\dim(\mathbf{H})$	$\kappa(\mathbf{H}) = \frac{\sigma_{\max}}{\sigma_{\min}}$
Bilayer	QEq	56800	2.569E2
	EE	56801	3.166E2
PETN	QEq	48256	1.347E2
	EE	48257	2.117E2
Silica	QEq	72000	1.061E2
	EE	72001	2.315E2
Water	QEq	78480	6.008E1
	EE	78481	1.022E2
	ACKS2	156962	8.231E4

Table A.3 **Tuned preconditioned solver parameters for shared-memory experiments.** For selection, a parameter search was conducted in order to minimize mean solver time for the **ICDD**(t)+**DS**(d), **ILUDD**(t)+**DS**(d), **FG-ICDD**(t, s)+**DS**(d), **FG-ILUDD**(t, s)+**DS**(d), and **SAI**(τ)+**DS**(d) preconditioned solvers. Parameters in the table are ordered from left to right as appearing in the above preconditioner names.

Solver Tolerance			10^{-6}	10^{-10}	10^{-14}
CM	Prec.	System			
QEq	**ICDD**	Bilayer	$(0.0, 0.6)$	$(0.0, 0.6)$	$(0.0, 0.8)$
		PETN	$(0.0, 0.6)$	$(0.0, 0.6)$	$(0.0, 0.6)$
		Silica	$(0.0, 0.6)$	$(0.0, 0.6)$	$(0.0, 0.6)$
		Water	$(0.01, 0.6)$	$(0.0, 0.6)$	$(0.0, 0.6)$
	FG-ICDD	Bilayer	$(0.0, 4, 0.6)$	$(0.0, 3, 0.6)$	$(0.01, 3, 0.6)$
		PETN	$(0.0, 3, 0.6)$	$(0.0, 2, 0.6)$	$(0.01, 2, 0.6)$
		Silica	$(0.0, 2, 0.6)$	$(0.0, 2, 0.6)$	$(0.0, 3, 0.6)$
		Water	$(0.0, 3, 0.6)$	$(0.0, 3, 0.6)$	$(0.0, 4, 0.6)$
	SAI	Bilayer	$(0.05, 0.6)$	$(0.1, 1.0)$	$(0.15, 1.0)$
		PETN	$(0.1, 0.6)$	$(0.15, 1.0)$	$(0.15, 1.0)$
		Silica	$(0.075, 0.8)$	$(0.1, 0.8)$	$(0.1, 0.8)$
		Water	$(0.1, 0.6)$	$(0.15, 0.6)$	$(0.1, 0.6)$
EE	**ILUDD**	Bilayer	$(0.0, 0.6)$	$(0.0, 0.6)$	$(0.0, 0.6)$
		PETN	$(0.0, 0.6)$	$(0.0, 0.6)$	$(0.0, 0.6)$
		Silca	$(0.0, 0.6)$	$(0.0, 0.6)$	$(0.0, 0.6)$
		Water	$(0.0, 0.6)$	$(0.0, 0.6)$	$(0.0, 0.6)$
	FG-ILUDD	Bilayer	$(0.01, 3, 0.6)$	$(0.01, 3, 0.6)$	$(0.01, 4, 0.6)$
		PETN	$(0.1, 2, 0.6)$	$(0.0, 2, 0.6)$	$(0.0, 2, 0.6)$
		Silica	$(0.0, 3, 0.6)$	$(0.0, 3, 0.6)$	$(0.01, 4, 0.6)$
		Water	$(0.1, 2, 0.6)$	$(0.0, 2, 0.6)$	$(0.01, 2, 0.6)$
	SAI	Bilayer	$(0.15, 1.0)$	$(0.1, 1.0)$	$(0.15, 1.0)$
		PETN	$(0.125, 0.8)$	$(0.15, 0.6)$	$(0.125, 1.0)$
		Silica	$(0.1, 0.8)$	$(0.1, 0.8)$	$(0.1, 0.8)$
		Water	$(0.1, 0.6)$	$(0.1, 0.6)$	$(0.1, 0.6)$
ACKS2	**ILUDD**	Water	$(0.0, 0.6)$	$(0.0, 0.6)$	$(0.0, 0.6)$
	FG-ILUDD	Water	$(0.0, 2, 0.6)$	$(0.0, 2, 0.6)$	$(0.0, 2, 0.6)$
	SAI	Water	$(0.075, 0.6)$	$(0.15, 0.6)$	$(0.15, 0.6)$

APPENDIX B

SUPPLEMENTAL DATA FOR OPTIMIZATION OF THE REAX FORCE FIELD FOR THE LITHIUM-OXYGEN SYSTEM USING A HIGH FIDELITY CHARGE MODEL

B.1 Fitted ACKS2-based ReaxFF Parameters

Table B.1 shows the parameters chosen to fit are labeled according to the standard ReaxFF parameter file indexing [103]. The first integer in parentheses indexes the parameter section corresponding to one of the following five categories: General, Atoms, Bonds, Off-diagonal, and Angles. The second integer indexes the block within the section. In this case, we fit lithium and oxygen atomic parameters O-O, Li-O and Li-Li bonds; Li-O off-diagonal parameters; and O-Li-O, O-O-Li, and Li-O-Li angle parameters. The third integer corresponds to the specific parameter within each block (see the ReaxFF Manual for details [103]).

The ACKS2-specific parameters are a global bond softness parameter Λ ((1, 35, 1) in the force field), element-specific EEM shielding parameters γ_i ((2, 2, 6) and (2, 5, 6)), and element-specific thresholding parameters σ_i ((2, 2, 23) and (2, 5, 23)), discussed in Section 4.2.2.

Table B.1 **The ReaxFF parameters adjusted to create the new ACKS2-based force field for Li-O.** Each parameter is constrained to the range defined by its lower bound (L.B.) and upper bound (U.B.) during the optimization procedure. The fourth and fifth columns report the fitted parameter values for the ACKS2-based ReaxFF and the 2016 QEq-based ReaxFF, respectively. Parameters marked with an asterisk* are specific to the ACKS2 model and were not fitted (N.F.) for the ReaxFF QEq 2016 force field. Parameteres not listed in the table are fixed to the values from the 2016 ReaxFF reference.

Parameter Indices	L.B.	U.B.	Fitted ACKS2 Values	ReaxFF QEq 2016 Values

Table B.1 (cont'd)

$(1, 35, 1)^*$	10.00	500.00	234.3037	N.F.
$(2, 2, 5)$	0.01	0.20	0.0502	0.1000
$(2, 2, 6)^*$	0.50	1.50	0.6318	N.F.
$(2, 2, 9)$	8.00	10.00	9.8993	9.7300
$(2, 2, 14)$	1.00	20.00	9.0609	8.5000
$(2, 2, 15)$	1.00	20.00	7.4275	8.3122
$(2, 2, 23)^*$	0.10	10.00	3.4489	N.F.
$(2, 2, 25)$	-5.00	-1.00	-4.0553	-3.5500
$(2, 5, 5)$	0.01	0.20	0.1996	0.1109
$(2, 5, 6)^*$	0.40	1.00	0.7511	N.F.
$(2, 5, 9)$	8.00	10.00	9.9731	9.3147
$(2, 5, 14)$	-12.00	12.00	-6.0479	-6.9345
$(2, 5, 15)$	1.00	20.00	6.2978	15.0000
$(2, 5, 23)^*$	0.50	15.00	9.3096	N.F.
$(2, 5, 25)$	-28.00	-17.00	-23.0012	-25.0000
$(3, 2, 1)$	140.00	150.00	141.9324	142.2858
$(3, 2, 4)$	0.00	1.50	1.2989	0.2506
$(3, 2, 8)$	0.00	1.75	0.4910	0.6051
$(3, 12, 1)$	75.00	85.00	76.8938	76.3632
$(3, 12, 4)$	-1.50	0.00	-1.1429	-0.4309
$(3, 12, 8)$	0.00	1.50	0.0631	0.1271
$(3, 15, 1)$	45.00	75.00	45.0406	52.4319
$(3, 15, 4)$	0.00	1.00	0.0064	0.2219
$(3, 15, 8)$	0.00	3.00	1.5411	0.5355
$(4, 8, 1)$	0.00	2.00	1.4736	0.1112
$(4, 8, 2)$	0.00	2.50	1.4147	1.6982
$(4, 8, 3)$	9.00	13.00	9.0283	11.0473
$(5, 31, 1)$	70.00	90.00	73.8444	83.7146
$(5, 31, 2)$	2.00	12.00	7.3527	8.7579
$(5, 31, 3)$	3.00	5.00	4.2355	4.0000
$(5, 31, 7)$	0.50	2.50	2.1889	1.2463
$(5, 32, 1)$	78.00	89.00	83.7686	81.6233
$(5, 32, 2)$	28.00	32.00	29.7164	30.0000
$(5, 32, 3)$	1.00	3.00	1.4275	2.0000

Table B.1 (cont'd)

$(5, 32, 7)$	0.00	2.00	0.8509	1.0000
$(5, 33, 1)$	85.00	110.00	93.3484	86.6874
$(5, 33, 2)$	15.00	25.00	19.0545	22.4076
$(5, 33, 3)$	3.00	5.00	3.5779	4.0000
$(5, 33, 7)$	0.50	2.50	1.7236	1.3084

B.2 Fitted QEq force field

As discussed in the main text, a ReaxFF force field was fitted using the same parameters and training set as the ACKS2 force field. While it achieved similar fitness values on the training set (see Fig. 4.1), it was unstable and did not capture the correct physics. This is illustrated in Fig. B.1. The failure of this force field is important for two reasons. Firstly, it highlights the fact that good fitness values on the training set alone do not guarantee a good force field; the proof is in the MD simulations. Secondly, it shows that the ACKS2 method is necessary to correctly capture bond-breaking. A QEq force field breaks when forced to fit bond-breaking data.

B.3 The MRCI+Q energies and MRCI atomic charges

Table B.2 shows the MRCI+Q energies and MRCI atomic charges that form part of the training dataset used in force field fitting.

Table B.2 **Nuclear geometries used in the calculations for the Li-O-Li system, along with the MRCI+Q total electronic energies and MRCI atomic charges on Li and O atoms.**

$r_{\mathrm{Li(1)–O}}$	$r_{\mathrm{Li(2)–O}}$	MRCI+Q	$q_{\mathrm{Li(1)}}$	q_{O}	$q_{\mathrm{Li(2)}}$
1.1381	1.6259	-89.865009	0.87	-1.79	0.93
1.3007	1.6259	-89.982903	0.91	-1.85	0.94
1.4633	1.6259	-90.030280	0.93	-1.88	0.94

Table B.2 (cont'd)

1.6259	1.6259	-90.043345	0.94	-1.89	0.94
1.7885	1.6259	-90.039912	0.94	-1.89	0.94
1.9511	1.6259	-90.028990	0.94	-1.88	0.94
2.1137	1.6259	-90.015046	0.93	-1.87	0.94
2.2763	1.6259	-90.000344	0.92	-1.85	0.94
2.4389	1.6259	-89.986037	0.90	-1.83	0.93
2.6015	1.6259	-89.972698	0.86	-1.80	0.93
2.7641	1.6259	-89.960250	0.82	-1.75	0.93
2.9266	1.6259	-89.947858	0.75	-1.68	0.93
3.0892	1.6259	-89.936609	0.65	-1.58	0.93
3.2518	1.6259	-89.926896	0.52	-1.46	0.93
3.4144	1.6259	-89.919775	0.38	-1.32	0.94
3.7396	1.6259	-89.909602	0.17	-1.11	0.94
4.0648	1.6259	-89.904763	0.07	-1.02	0.95
4.3900	1.6259	-89.902426	0.03	-0.98	0.95
4.7152	1.6259	-89.901202	0.01	-0.96	0.95
5.0403	1.6259	-89.900506	0.01	-0.96	0.95
5.3655	1.6259	-89.900080	0.00	-0.95	0.95
5.6907	1.6259	-89.899804	0.00	-0.95	0.95
6.0159	1.6259	-89.899618	0.00	-0.95	0.95
1.1381	1.1381	-89.672358	0.88	-1.75	0.88
1.3007	1.3007	-89.917700	0.92	-1.83	0.92
1.4633	1.4633	-90.016373	0.93	-1.87	0.93
1.6259	1.6259	-90.043345	0.94	-1.89	0.94
1.7885	1.7885	-90.036195	0.95	-1.89	0.95
1.9511	1.9511	-90.013953	0.94	-1.89	0.94
2.1137	2.1137	-89.985959	0.94	-1.87	0.94
2.2763	2.2763	-89.956459	0.92	-1.83	0.92
2.4389	2.4389	-89.928055	0.88	-1.77	0.88
2.6015	2.6015	-89.900678	0.83	-1.66	0.83
2.7641	2.7641	-89.877388	0.76	-1.51	0.76
2.9266	2.9266	-89.856157	0.67	-1.35	0.67
3.0892	3.0892	-89.838679	0.61	-1.22	0.61
3.2518	3.2518	-89.814604	0.50	-1.00	0.50

Table B.2 (cont'd)

3.4144	3.4144	-89.805887	0.50	-1.00	0.50
3.7396	3.7396	-89.790753	0.00	0.00	0.00
4.0648	4.0648	-89.791244	0.00	0.00	0.00
4.3900	4.3900	-89.791538	0.00	0.00	0.00
4.7152	4.7152	-89.791685	0.00	0.00	0.00
5.0403	5.0403	-89.791736	0.00	0.00	0.00
5.3655	5.3655	-89.791730	0.00	0.00	0.00
5.6907	5.6907	-89.791698	0.00	0.00	0.00
6.0159	6.0159	-89.791658	0.00	0.00	0.00

B.4 Further details of the MD simulations

Fig. B.2 shows detailed information on oxygen charge during cracking simulation from Fig. 4.6 (d). Also, Fig. B.3 provides analysis from a second cracking experiment with identical parameters and starting structures to Fig. 4.6 (a)-(c).

Fig. B.4 displays evolution of the RDF of the 324-atom Li_2O cube during NPT simulation in Section 4.3.2.

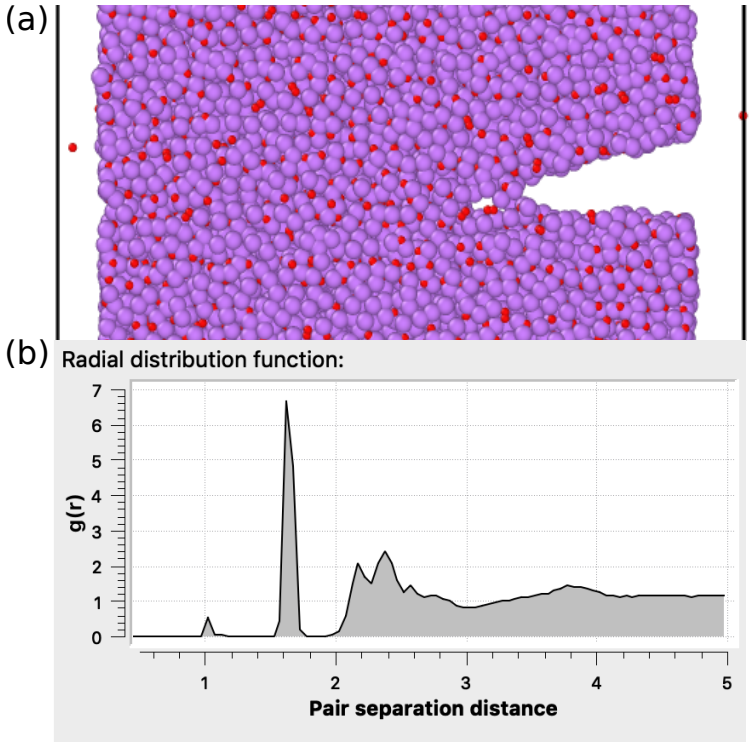


Figure B.1 **Deformation simulation using the fitted QEq force field for Li-O.** In the atomic visualization (a), atoms are clearly being ejected from the slab, despite the fact that this simulation was run at 1K in an attempt to avoid the ejections. In the radial distribution function (b), a small peak around 1 Å appears. This is unphysical and does not occur in the other force fields tested.

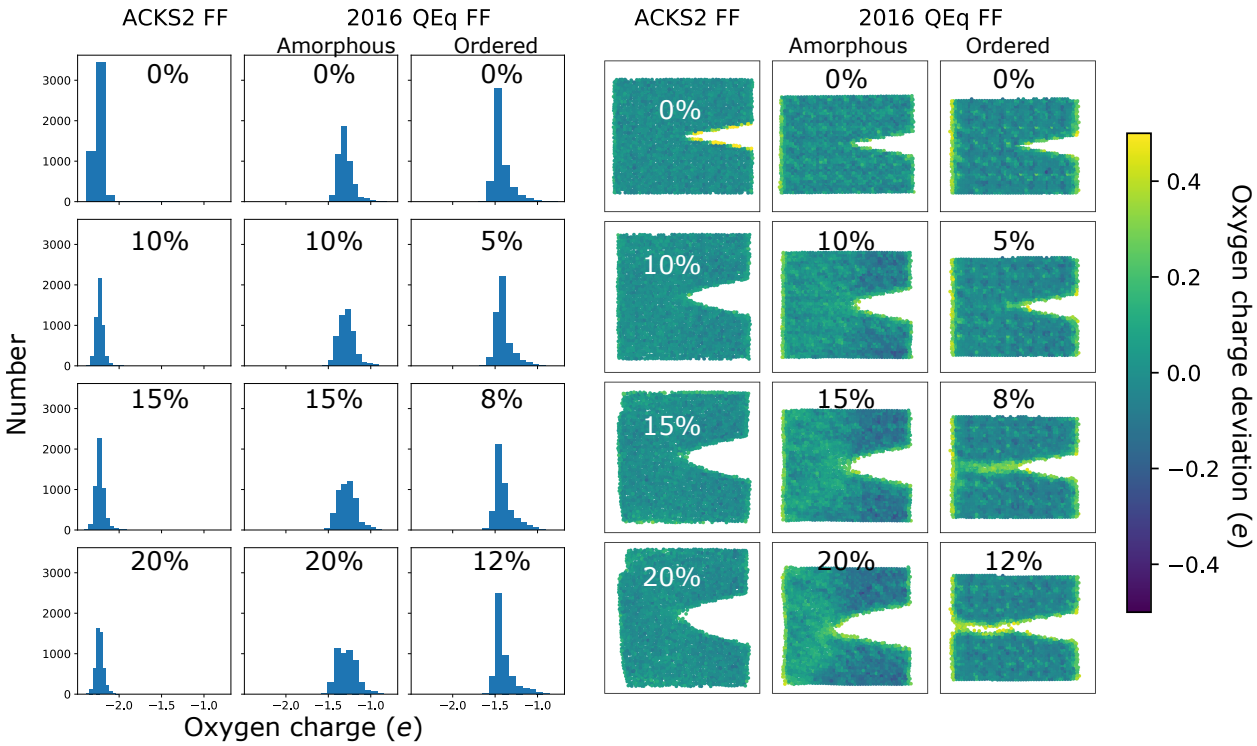


Figure B.2 **Detailed information on oxygen charge during cracking simulation.** Left: histograms of oxygen charge simulations at various points during the deformation simulations. Note that the axis scales are identical among the subplots. Right: deviation of oxygen charge from the average, shown at various points during the deformation simulations. It is clear from this figure that the charge-depleted population is larger in the QEq FF simulations, whereas the distribution is narrower and more symmetric under the ACKS2 FF.

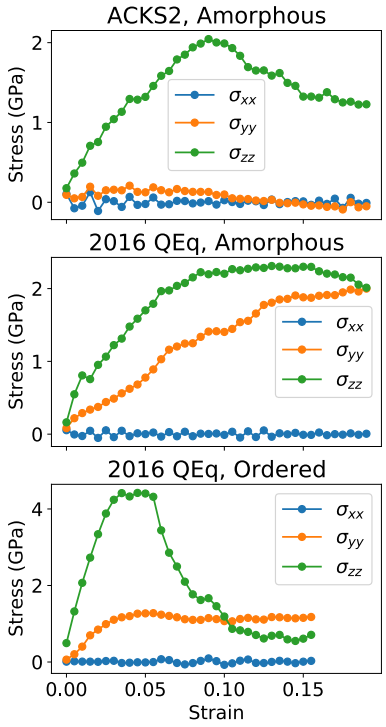


Figure B.3 **Second cracking experiment with identical parameters and starting structures.** The behavior is slightly different but the qualitative trends remain. This provides evidence that the conclusions are not sensitive to the stochastic nature of MD simulation.

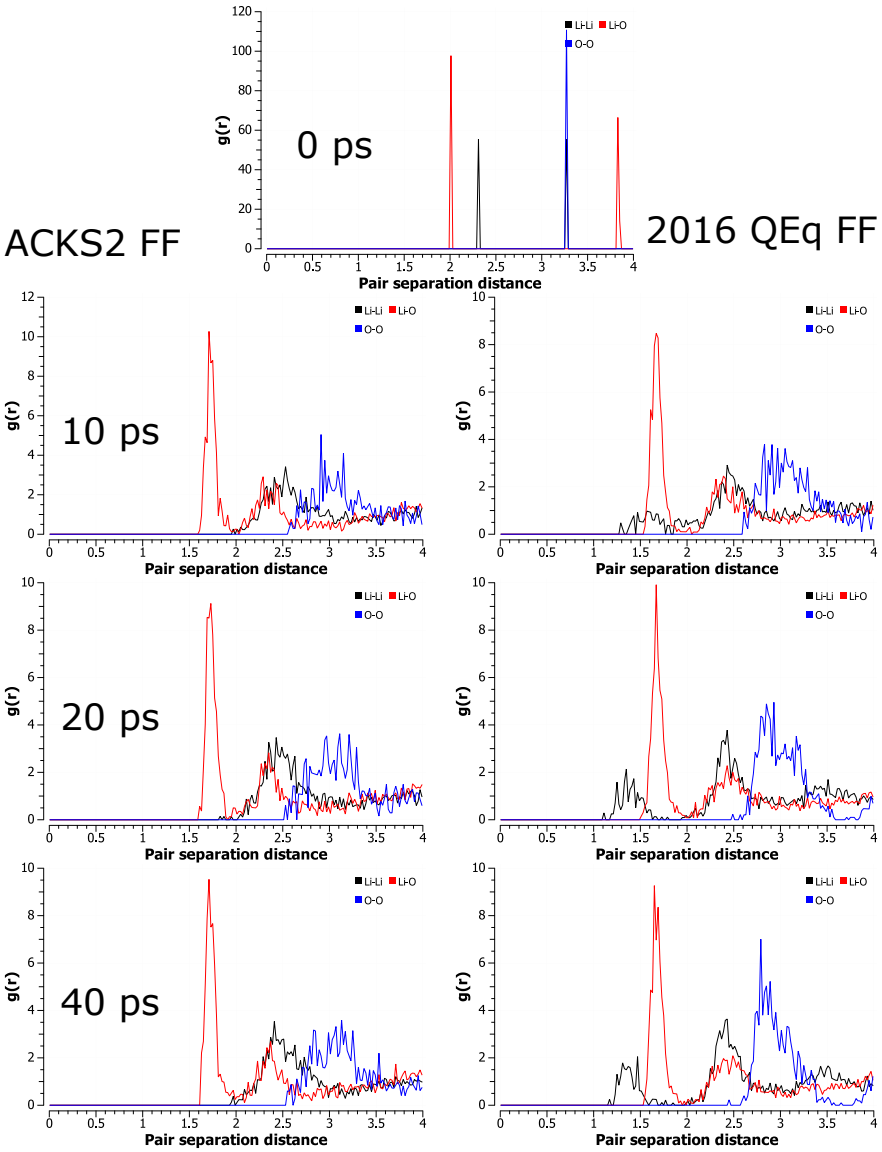


Figure B.4 **Evolution of the RDF during NPT simulation with the ACKS2 FF and the 2016 QEq FF.** Results are shown for the ACKS2 FF (left) and the 2016 QEq FF (right).

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] A. C. T. van Duin, S. Dasgupta, F. Lorant, and W. A. Goddard, "Reaxff: A reactive force field for hydrocarbons," *The Journal of Physical Chemistry A*, vol. 105, no. 41, pp. 9396–9409, 2001.

[2] T.-R. Shan, B. D. Devine, T. W. Kemper, S. B. Sinnott, and S. R. Phillpot, "Charge-optimized many-body potential for the hafnium/hafnium oxide system," *Phys. Rev. B*, vol. 81, p. 125328, Mar 2010.

[3] S. J. Stuart, A. B. Tutein, and J. A. Harrison, "A reactive potential for hydrocarbons with intermolecular interactions," *The Journal of Chemical Physics*, vol. 112, no. 14, pp. 6472–6486, 2000.

[4] T. P. Senftle, S. Hong, M. M. Islam, S. B. Kylasa, Y. Zheng, Y. K. Shin, C. Junkermeier, R. Engel-Herbert, M. J. Janik, H. M. Aktulga, T. Verstraelen, A. Grama, and A. C. T. van Duin, "The reaxff reactive force-field: development, applications and future directions," *npj Computational Materials*, vol. 2, p. 15011, 2016.

[5] H. M. Park, Yumi nd Atkulga, A. Grama, and A. Strachan, "Strain relaxation in si/ge/si nanoscale bars from molecular dynamics simulations," *Journal of Applied Physics*, vol. 106, no. 3, p. 034304, 2009.

[6] J. C. Fogarty, H. M. Aktulga, A. Y. Grama, A. C. T. van Duin, and S. A. Pandit, "A reactive molecular dynamics simulation of the silica-water interface," *The Journal of Chemical Physics*, vol. 132, no. 17, p. 174704, 2010.

[7] A. Grama, H. M. Aktulga, and S. B. Kylasa, "PuReMD, Purdue Reactive Molecular Dynamics package," 2014. Accessed on June 8, 2016.

[8] H. M. Aktulga, C. Knight, P. Coffman, K. A. O'Hearn, T.-R. Shan, and W. Jiang, "Optimizing the performance of reactive molecular dynamics simulations for many-core architectures," *The International Journal of High Performance Computing Applications*, vol. 33, no. 2, pp. 304–321, 2019.

[9] K. ichi Nomura, R. K. Kalia, A. Nakano, and P. Vashishta, "A scalable parallel algorithm for large-scale reactive force-field molecular dynamics simulations," *Computer Physics Communications*, vol. 178, no. 2, pp. 73–87, 2008.

[10] H. M. Aktulga, S. A. Pandit, A. C. T. van Duin, and A. Y. Grama, "Reactive molecular dynamics: Numerical methods and algorithmic techniques," *SIAM Journal on Scientific Computing*, vol. 34, no. 1, pp. C1–C23, 2012.

[11] H. Aktulga, J. Fogarty, S. Pandit, and A. Grama, "Parallel reactive molecular dynamics: Numerical methods and algorithmic techniques," *Parallel Computing*, vol. 38, no. 4, pp. 245–259, 2012.

[12] K. A. O'Hearn, A. Alperen, and H. M. Aktulga, "Fast Solvers for Charge Distribution Models on Shared Memory Platforms," *SIAM Journal on Scientific Computing*, vol. 42, no. 1, pp. C1–C22, 2020.

[13] K. A. O'Hearn and H. M. Aktulga, "Towards fast scalable solvers for charge equilibration in molecular dynamics applications," in *2016 7th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA)*, pp. 9–16, 2016.

[14] A. D. MacKerell, D. Bashford, M. Bellott, R. L. Dunbrack, J. D. Evanseck, M. J. Field, S. Fischer, J. Gao, H. Guo, S. Ha, D. Joseph-McCarthy, L. Kuchnir, K. Kuczera, F. T. K. Lau, C. Mattos, S. Michnick, T. Ngo, D. T. Nguyen, B. Prodhom, W. E. Reiher, B. Roux, M. Schlenkrich, J. C. Smith, R. Stote, J. Straub, M. Watanabe, J. Wiórkiewicz-Kuczera, D. Yin, and M. Karplus, "All-atom empirical potential for molecular modeling and dynamics studies of proteins," *The Journal of Physical Chemistry B*, vol. 102, no. 18, pp. 3586–3616, 1998. PMID: 24889800.

[15] W. D. Cornell, P. Cieplak, C. I. Bayly, I. R. Gould, K. M. Merz, D. M. Ferguson, D. C. Spellmeyer, T. Fox, J. W. Caldwell, and P. A. Kollman, "A second generation force field for the simulation of proteins, nucleic acids, and organic molecules," *Journal of the American Chemical Society*, vol. 117, no. 19, pp. 5179–5197, 1995.

[16] A. K. Rappe, C. J. Casewit, K. S. Colwell, W. A. Goddard, and W. M. Skiff, "Uff, a full periodic table force field for molecular mechanics and molecular dynamics simulations," *Journal of the American Chemical Society*, vol. 114, no. 25, pp. 10024–10035, 1992.

[17] T. A. Halgren and W. Damm, "Polarizable force fields," *Current opinion in structural biology*, vol. 11, no. 2, pp. 236–242, 2001.

[18] S. W. Rick and S. J. Stuart, "Potentials and algorithms for incorporating polarizability in computer simulations," *Reviews in computational chemistry*, vol. 18, pp. 89–146, 2002.

[19] S. Patel and C. L. B. III, "Fluctuating charge force fields: recent developments and applications from small molecules to macromolecular biological systems," *Molecular Simulation*, vol. 32, no. 3-4, pp. 231–249, 2006.

[20] P. Li and K. M. Merz, "Metal ion modeling using classical mechanics," *Chemical Reviews*, vol. 117, no. 3, pp. 1564–1686, 2017. PMID: 28045509.

[21] W. J. Mortier, S. K. Ghosh, and S. Shankar, "Electronegativity-equalization method for the calculation of atomic charges in molecules," *Journal of the American Chemical Society*, vol. 108, no. 15, pp. 4315–4320, 1986.

[22] A. K. Rappe and W. A. Goddard, "Charge equilibration for molecular dynamics simulations," *The Journal of Physical Chemistry*, vol. 95, no. 8, pp. 3358–3363, 1991.

[23] R. A. Nistor, J. G. Polihronov, M. H. Müser, and N. J. Mosey, "A generalization of the charge equilibration method for nonmetallic materials," *The Journal of Chemical Physics*, vol. 125, no. 9, p. 094108, 2006.

[24] T. Verstraelen, P. W. Ayers, V. Van Speybroeck, and M. Waroquier, "Acks2: Atom-condensed kohn-sham dft approximated to second order," *The Journal of Chemical Physics*, vol. 138, no. 7, p. 074108, 2013.

[25] A. Nakano, "Parallel multilevel preconditioned conjugate-gradient approach to variable-charge molecular dynamics," *Computer Physics Communications*, vol. 104, no. 1, pp. 59–69, 1997.

[26] W. J. Mortier, K. Van Genechten, and J. Gasteiger, "Electronegativity equalization: application and parametrization," *Journal of the American Chemical Society*, vol. 107, no. 4, pp. 829–835, 1985.

[27] S. Plimpton, "Fast parallel algorithms for short-range molecular dynamics," *Journal of Computational Physics*, vol. 117, no. 1, pp. 1–19, 1995.

[28] Y. Saad, *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, second ed., 2003.

[29] Y. Saad, "Ilut: A dual threshold incomplete lu factorization," *Numerical Linear Algebra with Applications*, vol. 1, no. 4, pp. 387–402, 1994.

[30] B. Carpentieri and M. Bollhöfer, "Symmetric inverse-based multilevel ilu preconditioning for solving dense complex non-hermitian systems in electromagnetics," *Progress in Electromagnetics Research*, vol. 128, pp. 55–74, 01 2012.

[31] X.-M. Pan and X.-Q. Sheng, "Sparse approximate inverse preconditioner for multiscale dynamic electromagnetic problems," *Radio Science*, vol. 49, no. 11, pp. 1041–1051, 2014.

[32] J. R. Allwright, R. Bordawekar, P. D. Coddington, K. Dincer, and C. L. Martin, "A comparison of parallel graph coloring algorithms," *Technical Report SCCS-666*, vol. Northeast Parallel Architecture Center, Syracuse University, 1995.

[33] E. F. F. Botta and A. van der Ploeg, "Renumbering strategies based on multi-level techniques combined with ILU-decompositions," *Zh. Vychisl. Mat. Mat. Fiz.*, vol. 37, no. 11, pp. 1294–1300, 1997.

[34] P. Hénon and Y. Saad, "A parallel multistage ilu factorization based on a hierarchical graph decomposition," *SIAM Journal on Scientific Computing*, vol. 28, no. 6, pp. 2266–2293, 2006.

[35] ı. V. ÇAtalyüRek, J. Feo, A. H. Gebremedhin, M. Halappanavar, and A. Pothen, "Graph coloring algorithms for multi-core and massively multithreaded architectures," *Parallel Comput.*, vol. 38, pp. 576–594, Oct. 2012.

[36] M. Naumov, P. Castonguay, and J. Cohen, "Parallel graph coloring with applications to the incomplete-lu factorization on the gpu," *NVIDIA Technical Report*, vol. NVR-2015-001, pp. 1–23, 2015.

[37] M. Naumov, "Parallel incomplete-lu and cholesky factorization in the preconditioned iterative methods on the gpu," *NVIDIA Technical Report*, vol. NVR-2012-003, pp. 1–19, 2012.

[38] E. Chow and A. Patel, "Fine-grained parallel incomplete lu factorization," *SIAM Journal on Scientific Computing*, vol. 37, no. 2, pp. C169–C193, 2015.

[39] M. Benzi and M. Tûma, "A comparative study of sparse approximate inverse preconditioners," *Applied Numerical Mathematics*, vol. 30, no. 2, pp. 305 – 340, 1999.

[40] M. Grote and T. Huckle, "Parallel preconditioning with sparse approximate inverses," *SIAM Journal on Scientific Computing*, vol. 18, no. 3, pp. 838–853, 1997.

[41] M. Benzi and M. Tuma, "A sparse approximate inverse preconditioner for nonsymmetric linear systems," *SIAM Journal on Scientific Computing*, vol. 19, no. 3, pp. 968–994, 1998.

[42] Y. Saad and M. Schultz, "Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM Journal on Scientific and Statistical Computing*, vol. 7, no. 3, pp. 856–869, 1986.

[43] K. A. O'Hearn, A. Alperen, and H. M. Aktulga, "Performance optimization of reactive molecular dynamics simulations with dynamic charge distribution models on distributed memory platforms," in *ICS '19: Proceedings of the Association of Computing Machinery International Conference on Supercomputing*, pp. 150–159, 2019.

[44] T. Liang, T.-R. Shan, Y.-T. Cheng, B. D. Devine, M. Noordhoek, Y. Li, Z. Lu, S. R. Phillpot, and S. B. Sinnott, "Classical atomistic simulations of surfaces and heterogeneous interfaces with the charge-optimized many body (comb) potentials," *Materials Science and Engineering: R: Reports*, vol. 74, no. 9, pp. 255–279, 2013.

[45] D. Frenkel and B. Smit, *Chapter 10 - Free Energies of Solids*. San Diego: Academic Press, second edition ed., 2002.

[46] P. Ghysels and W. Vanroose, "Hiding global synchronization latency in the preconditioned conjugate gradient algorithm," *Parallel Computing*, vol. 40, no. 7, pp. 224–238, 2014. 7th Workshop on Parallel Matrix Algorithms and Applications.

[47] M. Benzi and M. Tuma, "A comparative study of sparse approximate inverse preconditioners," *Applied Numerical Mathematics*, vol. 30, pp. 305–340, 04 1998.

[48] K. A. O'Hearn, M. W. Swift, J. Liu, I. Magoulas, P. Piecuch, A. C. T. van Duin, H. M. Aktulga, and Y. Qi, "Optimization of the reax force field for the lithium–oxygen system using a high fidelity charge model," *The Journal of Chemical Physics*, vol. 153, no. 8, p. 084107, 2020.

[49] P. Hohenberg and W. Kohn, "Inhomogeneous electron gas," *Phys. Rev.*, vol. 136, pp. B864–B871, Nov 1964.

[50] S. Huang, S. Zhang, T. Belytschko, S. S. Terdalkar, and T. Zhu, "Mechanics of nanocrack: Fracture, dislocation emission, and amorphization," *Journal of the Mechanics and Physics of Solids*, vol. 57, no. 5, pp. 840–850, 2009.

[51] A. A. Griffith and G. I. Taylor, "Vi. the phenomena of rupture and flow in solids," *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, vol. 221, no. 582-593, pp. 163–198, 1921.

[52] J. Belak, J. N. Glosli, D. B. Boercker, and I. F. Stowers, "Molecular dynamics simulation of mechanical deformation of ultra-thin metal and ceramic films," *MRS Proc.*, vol. 389, p. 181, 1995.

[53] I. Salehinia, J. Wang, D. Bahr, and H. Zbib, "Molecular dynamics simulations of plastic deformation in nb/nbc multilayers," *International Journal of Plasticity*, vol. 59, pp. 119–132, 2014.

[54] D. Wolf, V. Yamakov, S. Phillpot, A. Mukherjee, and H. Gleiter, "Deformation of nanocrystalline materials by molecular-dynamics simulation: relationship to experiments?," *Acta Materialia*, vol. 53, no. 1, pp. 1–40, 2005.

[55] P. S. Branício and J.-P. Rino, "Large deformation and amorphization of ni nanowires under uniaxial strain: A molecular dynamics study," *Phys. Rev. B*, vol. 62, pp. 16950–16955, Dec 2000.

[56] F. Yuan and L. Huang, "Brittle to ductile transition in densified silica glass," *Sci. Rep.*, vol. 4, p. 5035, 2015.

[57] K. Muralidharan, K.-D. Oh, P. A. Deymier, K. Runge, and J. H. Simmons, "Molecular dynamics simulations of atomic-level brittle fracture mechanisms in amorphous silica," *Journal of Materials Science*, vol. 42, pp. 4159–4169, 2007.

[58] M. M. Islam, A. Ostadhossein, O. Borodin, A. T. Yeates, W. W. Tipton, R. G. Hennig, N. Kumar, and A. C. T. van Duin, "Reaxff molecular dynamics simulations on lithiated sulfur cathode materials," *Phys. Chem. Chem. Phys.*, vol. 17, pp. 3383–3393, 2015.

[59] K. Kang and W. Cai, "Brittle and ductile fracture of semiconductor nanowires – molecular dynamics simulations," *Philosophical Magazine*, vol. 87, no. 14-15, pp. 2169–2189, 2007.

[60] L. Van Brutzel, C. L. Rountree, R. K. Kalia, A. Nakano, and P. Vashishta, "Dynamic fracture mechanisms in nanostructured and amorphous silica glasses million-atom molecular dynamics simulations," *MRS Online Proceedings Library*, vol. 703, no. 1, p. 39, 2011.

[61] C. Xu and W. Gao, "Pilling-bedworth ratio for oxidation of alloys," *Materials Research Innovations*, vol. 3, no. 4, pp. 231–235, 2000.

[62] J. Liu, *Atomic Simulation on Chemical-Mechanical Coupled Deformations in Complex Nano Structures*. PhD thesis, Michigan State University, 2019.

[63] F. G. Sen, A. T. Alpas, A. C. T. van Duin, and Y. Qi, "Oxidation-assisted ductility of aluminium nanowires," *Nature Communications*, vol. 5, p. 3959, 2014.

[64] B. Mortazavi, A. Ostadhossein, T. Rabczuk, and A. C. T. van Duin, "Mechanical response of all-mos2 single-layer heterostructures: a reaxff investigation," *Phys. Chem. Chem. Phys.*, vol. 18, pp. 23695–23701, 2016.

[65] G. M. Odegard, B. D. Jensen, S. Gowtham, J. Wu, J. He, and Z. Zhang, "Predicting mechanical response of crosslinked epoxy using reaxff," *Chemical Physics Letters*, vol. 591, pp. 175–178, 2014.

[66] B. D. Jensen, K. E. Wise, and G. M. Odegard, "The effect of time step, thermostat, and strain rate on reaxff simulations of mechanical failure in diamond, graphene, and carbon nanotube," *Journal of Computational Chemistry*, vol. 36, no. 21, pp. 1587–1596, 2015.

[67] H. Werner and P. J. Knowles, "An efficient internally contracted multiconfiguration–reference configuration interaction method," *The Journal of Chemical Physics*, vol. 89, no. 9, pp. 5803–5814, 1988.

[68] P. J. Knowles and H.-J. Werner, "An efficient method for the evaluation of coupling coefficients in configuration interaction calculations," *Chemical Physics Letters*, vol. 145, no. 6, pp. 514–522, 1988.

[69] W. Kohn and L. J. Sham, "Self-consistent equations including exchange and correlation effects," *Phys. Rev.*, vol. 140, pp. A1133–A1138, Nov 1965.

[70] A. Ostadhossein, S.-Y. Kim, E. D. Cubuk, Y. Qi, and A. C. T. van Duin, "Atomic insight into the lithium storage and diffusion mechanism of sio2/al2o3 electrodes of lithium ion batteries: Reaxff reactive force field modeling," *The Journal of Physical Chemistry A*, vol. 120, no. 13, pp. 2114–2127, 2016. PMID: 26978039.

[71] A. Yulaev, V. Oleshko, P. Haney, J. Liu, Y. Qi, A. A. Talin, M. S. Leite, and A. Kolmakov, "From microparticles to nanowires and back: Radical transformations in plated li metal morphology revealed via in situ scanning electron microscopy," *Nano Letters*, vol. 18, no. 3, pp. 1644–1650, 2018. PMID: 29397748.

[72] B. Narayanan, A. C. T. van Duin, B. B. Kappes, I. E. Reimanis, and C. V. Ciobanu, "A reactive force field for lithium–aluminum silicates with applications to eucryptite phases," *Modelling and Simulation in Materials Science and Engineering*, vol. 20, p. 015002, nov 2011.

[73] G. Kresse and J. Furthmüller, "Efficient iterative schemes for ab initio total-energy calculations using a plane-wave basis set," *Phys. Rev. B*, vol. 54, pp. 11169–11186, Oct 1996.

[74] J. P. Perdew, K. Burke, and M. Ernzerhof, "Generalized gradient approximation made simple," *Phys. Rev. Lett.*, vol. 77, pp. 3865–3868, Oct 1996.

[75] P. E. Blöchl, "Projector augmented-wave method," *Phys. Rev. B*, vol. 50, pp. 17953–17979, Dec 1994.

[76] H.-J. Werner, P. J. Knowles, G. Knizia, F. R. Manby, M. Schütz, P. Celani, T. Korona, R. Lindh, A. Mitrushenkov, G. Rauhut, K. R. Shamasundar, T. B. Adler, R. D. Amos, A. Bernhardsson, A. Berning, D. L. Cooper, M. J. O. Deegan, A. J. Dobbyn, F. Eckert, E. Goll, C. Hampel, A. Hesselmann, G. Hetzer, T. Hrenar, G. Jansen, C. Köppl, Y. Liu, A. W. Lloyd, R. A. Mata, A. J. May, S. J. McNicholas, W. Meyer, M. E. Mura, A. Nicklass, D. P. O'Neill, P. Palmieri, K. Pflüger, R. Pitzer, M. Reiher, T. Shiozaki, H. Stoll, A. J. Stone, R. Tarroni, T. Thorsteinsson, and M. Wang. MOLPRO, version 2010.1, a package of *ab initio* programs, see https://www.molpro.net.

[77] K. Ruedenberg, M. W. Schmidt, M. M. Gilbert, and S. Elbert, "Are atoms intrinsic to molecular electronic wavefunctions? i. the fors model," *Chemical Physics*, vol. 71, no. 1, pp. 41–49, 1982.

[78] B. O. Roos, *The Complete Active Space Self-Consistent Field Method and its Applications in Electronic Structure Calculations*, pp. 399–445. John Wiley & Sons, Ltd, 1987.

[79] J. B. Collins and A. Streitwieser Jr, "Integrated spatial electron populations in molecules: Application to simple molecules," *Journal of Computational Chemistry*, vol. 1, no. 1, pp. 81–87, 1980.

[80] A. E. Reed and F. Weinhold, "Natural bond orbital analysis of near-hartree–fock water dimer," *The Journal of Chemical Physics*, vol. 78, no. 6, pp. 4066–4073, 1983.

[81] A. E. Reed, R. B. Weinstock, and F. Weinhold, "Natural population analysis," *The Journal of Chemical Physics*, vol. 83, no. 2, pp. 735–746, 1985.

[82] A. E. Reed and F. Weinhold, "Natural localized molecular orbitals," *The Journal of Chemical Physics*, vol. 83, no. 4, pp. 1736–1740, 1985.

[83] R. A. Mata and H.-J. Werner, "Local correlation methods with a natural localized molecular orbital basis," *Molecular Physics*, vol. 105, no. 19-22, pp. 2753–2761, 2007.

[84] T. H. Dunning, "Gaussian basis sets for use in correlated molecular calculations. i. the atoms boron through neon and hydrogen," *The Journal of Chemical Physics*, vol. 90, no. 2, pp. 1007–1023, 1989.

[85] R. A. Kendall, T. H. Dunning, and R. J. Harrison, "Electron affinities of the first-row atoms revisited. systematic basis sets and wave functions," *The Journal of Chemical Physics*, vol. 96, no. 9, pp. 6796–6806, 1992.

[86] B. P. Prascher, D. E. Woon, K. A. Peterson, T. H. Dunning, and A. K. Wilson, "Gaussian basis sets for use in correlated molecular calculations. vii. valence, core-valence, and scalar relativistic basis sets for li, be, na, and mg," *Theoretical Chemistry Accounts*, vol. 128, no. 1, pp. 69–82, 2011.

[87] S. Patel and C. L. Brooks III, "Charmm fluctuating charge force field for proteins: I parameterization and application to bulk organic liquid simulations," *Journal of Computational Chemistry*, vol. 25, no. 1, pp. 1–16, 2004.

[88] R. Chelli, P. Procacci, R. Righini, and S. Califano, "Electrical response in chemical potential equalization schemes," *The Journal of Chemical Physics*, vol. 111, no. 18, pp. 8569–8575, 1999.

[89] W. B. Dapp and M. H. Müser, "Redox reactions with empirical potentials: Atomistic battery discharge simulations," *The Journal of Chemical Physics*, vol. 139, no. 6, p. 064106, 2013.

[90] M. M. Islam and A. C. T. van Duin, "Reductive decomposition reactions of ethylene carbonate by explicit electron transfer from lithium: An ereaxff molecular dynamics study," *The Journal of Physical Chemistry C*, vol. 120, no. 48, pp. 27128–27134, 2016.

[91] M. Dittner, J. Müller, H. M. Aktulga, and B. Hartke, "Efficient global optimization of reactive force-field parameters," *Journal of Computational Chemistry*, vol. 36, no. 20, pp. 1550–1561, 2015.

[92] G. Shchygol, A. Yakovlev, T. Trnka, A. C. T. van Duin, and T. Verstraelen, "Reaxff parameter optimization with monte-carlo and evolutionary algorithms: Guidelines and insights," *Journal of Chemical Theory and Computation*, vol. 15, no. 12, pp. 6799–6812, 2019. PMID: 31657217.

[93] J. M. Dieterich and B. Hartke, "Ogolem: Global cluster structure optimisation for arbitrary mixtures of flexible molecules. a multiscaling, object-oriented approach," *Molecular Physics*, vol. 108, no. 3-4, pp. 279–291, 2010.

[94] S. Kylasa, H. Aktulga, and A. Grama, "Puremd-gpu: A reactive molecular dynamics simulation package for gpus," *Journal of Computational Physics*, vol. 272, pp. 343–359, 2014.

[95] C. Kittel, *Introduction to solid state physics*. J. Wiley, 8th ed ed., 2004.

[96] M. K. Y. Chan, E. L. Shirley, N. K. Karan, M. Balasubramanian, Y. Ren, J. P. Greeley, and T. T. Fister, "Structure of lithium peroxide," *The Journal of Physical Chemistry Letters*, vol. 2, no. 19, pp. 2483–2486, 2011.

[97] S. Hull, T. Farley, W. Hayes, and M. Hutchings, "The elastic properties of lithium oxide and their variation with temperature," *Journal of Nuclear Materials*, vol. 160, no. 2, pp. 125–134, 1988.

[98] S.-H. Cheng and C. T. Sun, "Size-dependent fracture toughness of nanoscale structures: Crack-tip stress approach in molecular dynamics," *Journal of Nanomechanics and Micromechanics*, vol. 4, no. 4, p. A4014001, 2014.

[99] W. Tang, E. Sanville, and G. Henkelman, "A grid-based bader analysis algorithm without lattice bias," *Journal of Physics: Condensed Matter*, vol. 21, p. 084204, jan 2009.

[100] D. Merrill and NVIDIA CORPORATION. CUB, version 1.12.1, see https://github.com/NVIDIA/cub.

[101] M. M. Hussain and N. Fujimoto, "Gpu-based parallel multi-objective particle swarm optimization for large swarms and high dimensional problems," *Parallel Computing*, vol. 92, p. 102589, 2020.

[102] C. A. Navarro, R. Carrasco, R. J. Barrientos, J. A. Riquelme, and R. Vega, "Gpu tensor cores for fast arithmetic reductions," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 72–84, 2021.

[103] A. C. T. van Duin, *ReaxFF User Manual*, 2002 (Accessed December 19, 2019), see https://www.engr.psu.edu/adri/ReaxffManual.aspx.