# GPU Accelerated Barycentric Treecodes and their Application to Kohn-Sham Density Functional Theory

by

Nathanial J. Vaughn

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Applied and Interdisciplinary Mathematics and Scientific Computing)
in the University of Michigan
2020

Doctoral Committee:

Associate Professor Vikram Gavini, Co-Chair
Professor Robert Krasny, Co-Chair
Professor Selim Esedoglu
Associate Professor Shravan Veerapaneni

Nathanial J. Vaughn

njvaughn@umich.edu

ORCID iD: 0000-0001-8680-5616

## *Dedication*

Dedicated to Emily, thank you for your constant support and encouragement.

***Acknowledgments***

This work was made possible by the support of many people. First, I would like to thank my advisors Dr. Robert Krasny and Dr. Vikram Gavini. Their willingness to embrace interdisciplinary research enabled us to explore this frontier of integral equation methods for electronic structure calculations. The project benefited greatly from having both of their perspectives and expertises. I am grateful for their support and guidance during this project; I will take many lessons from both of them going forward. I also thank my committee members, Dr. Selim Esedoglu and Dr. Shravan Veerapaneni. I appreciate their feedback on the planning of this research project, the dissertation, and their continued availability.

Next I want to recognize the funding sources that have supported me during this research. This work was supported by National Science Foundation grant DMS-1819094, Extreme Science and Engineering Discovery Environment (XSEDE) grants ACI-1548562 and ASC-190062, and the Mcubed program and top-off fellowship from the Michigan Institute for Computational Discovery and Engineering (MICDE) at the University of Michigan.

Next, I thank the staff in the Department of Mathematics for all of their care and assistance. I also thank the support staff of the university's Advanced Research Computing – Technology Services. Countless times throughout the past years you have helped me with computing related questions; I greatly appreciate the timely and high quality assistance you gave me.

Next, I acknowledge my fellow group members in the Computational Materials Physics Group that helped me in many ways; Bikash Kanungo, Sambit Das, Phani Motamarri, Ian Lin, Nelson Rufus, and Paavai Pari. Not only did I enjoy the group camaraderie, I learned a great deal from the discussions about the challenges you faced in your methods and the solutions you developed; they were instrumental in the development of my own solutions facing similar challenges. I thank fellow graduate student Leighton Wilson, both as a collaborator and as a friend. I've enjoyed the long hours collaborating on our code and the long hours grilling in the backyard.

Finally, I thank my family. My parents and sisters have always been supportive and encouraging. Leaving North Carolina was a challenge, but I have enjoyed the countless phone calls throughout the years and appreciate all the moral support. I also thank my extended family that lives in Michigan; they welcomed me into their traditions and helped make Michigan feel like home. I thank my fiancée Emily for her unwavering support and the sacrifices she has made to help me pursue my goals. You (and Pepper) have made this journey very enjoyable.

# Table of Contents

# List of Figures

Figure

# List of Tables

Table

# List of Symbols

$n$ interpolation degree, Eq. (2.2)

$C$ cluster of source particles, Eq. (2.8)

$N_C$ number of particles in cluster $C$, Alg. 2.1

$N_L$ maximum number of particles in a source leaf cluster, Alg. 2.1

$N_B$ maximum number of particles in a target batch, Alg. 2.1

$r_B$ radius of target batch, Eq. (2.23), Fig. 2.3

$r_C$ radius of source cluster, Eq. (2.23), Fig. 2.3

$R$ distance between target batch and source cluster, Eq. (2.23), Fig. 2.3

$\theta$ multipole acceptance criterion, separation parameter, Eq. (2.23)

$h$ size check prefactor, Eq. (2.23)

$\mathcal{H}[\rho]$ Kohn-Sham Hamiltonian, Eq. (3.1)

$\rho(\mathbf{r})$ electron density, Eq. (3.4)

$(\varepsilon_i, \psi_i)$ Kohn-Sham eigenpairs, Eq. (3.1)

$V_{eff}[\rho]$ Kohn-Sham effective potential, Eq. (3.2)

$N_A$ number of atoms, Eq. (3.3)

$\mathbf{R}_j$ atomic position, Eq. (3.3)

$Z_j$ nuclear charges, Eq. (3.3)

$N_e$ number of electrons, Eq. (3.5)

$k_B$ Boltzmann constant, Eq. (3.4)

$T$ temperature, Eq. (3.4)

$\mu_F$ Fermi energy, Eq. (3.5)

$N_w$ number of wavefunctions, Eq. (3.5)

$tol_{scf}$ Self Consistent Field iteration tolerance, Alg. 3.1

$\beta$ Anderson mixing parameter, Alg. 3.1

$\mathcal{G}(\varepsilon)$ Green's function integral operator, Eq. (3.21)

$(\mu_i, \phi_i)$ integral operator eigenpairs, Eq. (3.55)

$tol_m$ mesh refinement parameter, Eq. (3.41)

$N_c$ number of cells in mesh, Eq. (3.23)

$p$ quadrature order, Eq. (3.23)

$N_m$ number of quadrature points in mesh, Eq. (3.23)

$tol_{gi}$ Green iteration tolerance, Alg. 3.2

$\alpha$ Gaussian singularity subtraction parameter, Eq. (3.48)

$V_{shift}$ potential gauge shift, Eq. (3.47)

$Z^J_{valence}$ valence charge of nucleus $J$, Fig. 4.1

$N_{valence}$ number of valence electrons in system, Fig. 4.1

$r_{coarse}$ coarse mesh refinement ball radius for pseudopotential calculations, Fig. 4.4

$\Delta h_{inner}$ fine mesh spacing inside $r_{coarse}$, Fig. 4.4

$\Delta h_{outer}$ far field mesh spacing outside $r_{coarse}$, Fig. 4.4

$r_{fine}$ fine mesh refinement ball radius, Fig. 4.5

$d_{cf}$ refinement ratio between coarse and fine meshes, Fig. 4.5

$L_x$, $L_y$, $L_z$ domain bounds in the *x*, *y*, and *z* dimensions

$\ell_{init}$ initial refinement parameter for parallel mesh generation, Fig. 4.6

## Abstract

Electronic structure calculations are an integral step in the design and engineering of materials. Kohn-Sham Density Functional Theory (KS-DFT) is a computationally tractable first-principles formulation of electronic structure that is widely used to predict material properties. KS-DFT represents the electron density in terms of single-electron wavefunctions by replacing explicit electron-electron interactions with a mean-field interaction and an approximation of an exchange-correlation functional. The development of numerical methods for KS-DFT is an ongoing area of research; more efficient numerical methods will enable the simulation of larger and more challenging materials systems and improve KS-DFT's predictive capability.

The goal of this work was to develop an integral equation based numerical method for KS-DFT, both to investigate the feasibility and to explore any advantages of an integral equation approach compared to the preexisting numerical methods based on differential equations. We achieved this goal through the development of Treecode-Accelerated Green Iteration (TAGI). We used the method of Green's functions to convert the eigenvalue problem for the Kohn-Sham differential operator into a fixed-point problem for an equivalent integral operator. We developed real-space discretization techniques to numerically evaluate the integral operators with high accuracy, including adaptive mesh refinement schemes, a higher order Fejér quadrature rule, and singularity-subtraction schemes to weaken the Green's function singularities. Next, we leveraged fixed-point acceleration techniques to improve the convergence rates of the fixed-point iterations. Finally, we developed treecodes based on barycentric interpolation; these fast summation algorithms reduce the computational complexity of evaluating the discretized integral operators. We developed these barycentric treecodes to run efficiently on high performance computing architectures; in particular, parallelized the computations with a distributed memory Message Passing Interface (MPI) implementation, and accelerated the computations with Graphics Processing Units (GPUs). We achieved high GPU efficiency by leveraging an extra level of parallelism afforded by the barycentric approximations that is not present for previous treecodes based on multipole expansions.

We developed TAGI for two types of calculations; all-electron and Optimized Norm-Conserving Vanderbilt (ONCV) pseudopotential. In all-electron calculations, atomic nuclei are represented by singular Coulomb potentials, and wavefunctions are computed for each electron in the system. The singular nuclear potential, which gives rise to sharply varying wavefunctions near the nuclei,

requires substantial mesh refinement to achieve high accuracy. In pseudopotential calculations, the singular nuclear potentials are replaced with smooth, non-singular pseudopotentials. These are generated by absorbing an atom's chemically inert core electrons into its nucleus, leaving only the chemically active valence electrons to be computed. These calculations require significantly less local refinement near the nuclei, enabling TAGI calculations of larger systems.

We demonstrated TAGI by computing the ground-state energy of a variety of molecules for all-electron and pseudopotential calculations. We showed TAGI's ability to systematically converge to chemical accuracy through the adaptive mesh refinement schemes. These calculations were performed on up to eight compute nodes containing a total of thirty two GPUs. The techniques developed in this work have enabled TAGI to calculate chemically accurate ground state energies of molecules containing up to a few hundred electrons in several hours (all-electron $C_6H_6$ in 4 hours, pseudopotential $Si_{30}H_{40}$ in 3 hours, and pseudopotential $C_{60}$ in 8 hours). While TAGI does not outperform the more mature methods based on differential equations, this work constitutes a substantial proof of concept for treecode-accelerated integral equation based methods for KS-DFT and presents numerous opportunities for further improvement.

# Chapter 1

# Introduction

## 1.1 Application: Kohn-Sham Density Functional Theory

Electronic structure calculations based on first-principles formulations of quantum mechanics are an integral part of materials design and engineering. Many macroscopic physical properties of a material can be derived and computed from its electronic structure [1]. Through simulation, scientists are able to rapidly screen many candidate materials for desirable properties before attempting to synthesize them [2, 3, 4]. Improving the numerical methods for electronic structure calculations is an active area of research [5, 6, 7, 8], with goals to both accelerate the rate of screening and discovery and to extend calculations to larger and more complex materials systems. In this work we develop a Green's function and integral equation based method for electronic structure calculations, specifically for Kohn-Sham Density Functional Theory (KS-DFT), in order to assess the feasibility of this alternative approach and explore any potential advantages over the preexisting differential equation based methods.

Consider a material system containing $N$ electrons. Denoting $\mathbf{x}_i = (\mathbf{r}_i, s_i)$ as the combined spatial coordinates $\mathbf{r}_i$ and spin $s_i$ of electron $i$, the time-independent Schrödinger equation is a first-principles formulation of the electronic structure problem based on a many-body wavefunction $\Psi(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N)$ and a many-body Hamiltonian $\mathcal{H}^S$ that explicitly accounts for electron-electron interactions. The Schrödinger equation is given by

$$\mathcal{H}^S \Psi(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N) = E\Psi(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N), \tag{1.1}$$

where $E$ is the ground state energy. Despite the ubiquity of the Schrödinger equation in electronic structure theory, numerical methods for computing the many-body wavefunction scale exponentially in the number of electrons $N$ and are intractable for large systems. Density Functional Theory (DFT) is an alternative first-principles formulation of electronic structure that is based directly on the electron electron density $\rho(\mathbf{r})$. DFT follows from the Hohenberg-Kohn theorems [9], which state that (1) the external potential of a system is uniquely determined (up to an additive constant) by the system's ground state electron density, and (2) there exists an energy functional which is

minimized by the ground state electron density. The most widely used form of DFT is Kohn-Sham Density Functional Theory (KS-DFT) [10], which replaces the system of interacting electrons with a fictitious system of non-interacting electrons that give rise to an identical electron density. This formulation gives rise to the Kohn-Sham equations for a set of $N$ single-electron wavefunctions $\psi_i(\mathbf{r})$, and resulting electron density,

$$\mathcal{H}^{KS}[\rho(\mathbf{r})]\psi_i(\mathbf{r}) = \varepsilon_i\psi_i(\mathbf{r}), \quad i = 1, 2, \ldots, N, \qquad \rho(\mathbf{r}) = \sum_{i=1}^{N} |\psi_i(\mathbf{r})|^2, \tag{1.2}$$

where $\varepsilon_i$ are the eigenvalues corresponding to each single-body wavefunction. In principle, since the fictitious system of non-interacting electrons gives rise to the same electron density as the real material system, the Hohenberg-Kohn theorems state that the Kohn-Sham formulation is exact for the ground state properties of the system. However, the Kohn-Sham Hamiltonian $\mathcal{H}^{KS}[\rho(\mathbf{r})]$ contains an exchange-correlation functional in order to implicitly model electron-electron interactions, which is not known explicitly and is modeled in practice. Approximating the exchange-correlation functional is an active area of research [11, 12, 13], and better approximations enable Kohn-Sham DFT to more accurately predict ground state materials properties. Importantly, computing the set of single-body wavefunctions in Eq. (1.2), and subsequently the electron density, scales only cubically in the number of electrons. This has led to DFT becoming the workhorse for electronic structure calculations in recent decades [13], where it has aided in the research and development of materials in a wide range of applications, several of which are discussed below.

One scientific application that has benefited greatly from electronic structure calculations is renewable energy. DFT has been used extensively in the design and engineering of materials for a wide range renewable energy applications [2] ranging from batteries, to photovoltaics, to superconductors, to thermoelectrics. The success of many renewable energy sources is critically dependent on improving the efficiency of energy devices, for example the efficiency with which a photovoltaic extracts energy from sunlight and the efficiency of batteries to store excess energy during the daytime for use at nighttime. DFT calculations are used to computationally screen large numbers of potentially efficient materials for advantageous properties, accelerating the rate of discovery of efficient materials. One example where DFT has helped guide the design and engineering of energy related materials is batteries. Lithium-ion battery development has been aided by predictive DFT calculations such as those by [3, 4] to compute the energies of different lithium-ion crystal structures and resulting cathode voltages arising during charging. Similarly, DFT calculations have been performed for battery anodes such as the prediction of 2D carbide materials [14, 15]. Calculations of aluminum and magnesium compounds aided in the development of rechargeable batteries by computing Natural Bond Orbitals (NBO), relaxed geometries, and theoretical Raman spectra [16]. A second example of DFT calculations guiding material design

is for photovoltaics, which convert energy from sunlight. Designing materials to both increase the conversion efficiency and reduce the manufacturing cost is essential to the success of this renewable source. DFT calculations have been used in the screening of potentially efficient inorganic compounds [17, 18] and the design of efficient sensitizers for dye-sensitized solar cells [19, 20, 21]. A third example of DFT calculations aiding in energy material design is thermovoltaics, which can be used to convert waste heat into usable electricity and thereby improve efficiency of other energy devices. In particular, DFT calculations have been used to investigate structural, electronic, magnetic, phononic, and thermoelectric properties of rare earth multiferroic manganite compounds [22], group IV-VI semiconductors [23], and the multi-cation compound $Cu_2COSnS_4$ [24], to name a few.

Despite its tremendous impact thus far, there are still significant computational challenges facing KS-DFT. Although the cubic complexity is more tractable than the exponential complexity of the original Schrödinger problem, large-scale KS-DFT calculations strain the computational capability of the world's largest supercomputers. Alternative linearly scaling methods have been developed that efficiently treat large-scale insulating systems [25, 26], however many scientific inquiries require simulating large metallic systems that cannot be handled by these alternatives [2, 27]. Hence the development of highly efficient and parallelizable numerical methods for KS-DFT remains an active area of research.

There are a variety of numerical approaches for KS-DFT, each of which face challenges that limit the system size. For example, the most commonly used methods are based on plane-waves and discretization of Fourier space [7, 28, 29], however these face limitations in parallel scalability and inefficiencies in non-periodic settings. Real-space methods were developed with Gaussian bases to alleviate some of the plane-wave limitations [6]. Such methods scale well, but lack systematic convergence due to the incompleteness of the basis set. This has led to development of other real-space methods based on finite-differences [30, 31, 32, 33] and finite-elements [34, 35, 36, 37]. State of the art KS-DFT codes running on the world's largest supercomputers are capable of simulating metallic systems containing roughly 100,000 electrons [38, 39, 40, 5].

This work proposes, implements, and demonstrates a novel real-space KS-DFT method based on Green's functions and treecode-accelerated convolution integrals called Treecode-Accelerated Green Iteration (TAGI). Unlike the previously described methods which solve an eigenvalue problem for the Kohn-Sham differential equations, TAGI solves a fixed-point problem for an equivalent integral equation. This approach is made possible by the ongoing development of the parallelized and GPU-accelerated barycentric treecode presented in Chapter 2. While our preliminary implementation of TAGI is not as mature as the preexisting approaches based on differential equations, and we are not able to scale to such large problem sizes, we are able to demonstrate a significant proof of concept and scale to several hundred electrons. We are also able to systematically converge to desired accuracy through our adaptive mesh refinement schemes. The KS-DFT formulation and the details

of TAGI are the focus of Chapters 3 and 4. Specifically, Chapter 3 develops TAGI for all-electron calculations, which use the true singular nuclear potentials for each atom and explicitly account for all of the electrons. Chapter 4 presents the extension of TAGI to pseudopotential calculations, in which the "core" electrons are absorbed into the atomic nucleus; in this formulation the nuclear potential is regularized and only the "valence" electron wavefunctions are computed, enabling the simulation of larger systems.

### 1.2 Analytic Formulation: The Method of Green's Functions

The Kohn-Sham equations represent a nonlinear eigenvalue problem for the Kohn-Sham Hamiltonian $\mathcal{H}^{KS}$, which is a second order linear differential operator given by

$$\mathcal{H}^{KS} = -\frac{1}{2}\nabla^2 + V_{eff}[\rho](\mathbf{r}), \quad \mathbf{r} \in \mathbb{R}^3. \tag{1.3}$$

Here, $V_{eff}[\rho](\mathbf{r})$ is the effective potential for the non-interacting electrons; it depends on the electron density $\rho$, therefore Eq. (1.2) represents a nonlinear eigenvalue problem. The details of the expression for the effective potential are reserved for Chapter 3; for now it is sufficient to know that the operator is a second order linear differential operator. All of the preexisting methods described above have one thing in common; they discretize the differential operator numerically and solve the resulting finite-dimensional eigenvalue problem. However, there is an alternative approach for linear differential equations called the method of Green's functions [41] that begins by analytically inverting the differential operator to obtain an equivalent integral equation. There are numerical methods for solving the resulting integral equation, distinct from those described above. The properties of the discrete integral operator are inherently different than the properties of the discrete differential operator; hence the numerical methods for the integral equation face different challenges than those of the differential equation and it is worthwhile to investigate their performance in comparison.

There are two steps to the method of Green's functions; the first involves deriving or constructing a suitable Green's function, the second involves using that Green's function to analytically invert the differential equation. To illustrate, the following discussion presents these two steps for a modified Helmholtz equation in $\mathbb{R}^3$ with free-space boundary conditions, which is relevant for the integral equation formulation of the Kohn-Sham problem described in detail in Chapter 3.

**Step 1: Deriving the Green's function**. Consider the boundary value problem for the modified Helmholtz operator in $\mathbb{R}^3$,

$$(\nabla^2 - k^2)u(\mathbf{r}) = f(\mathbf{r}), \tag{1.4}$$

for $k > 0$, with free-space boundary conditions on $u(\mathbf{r})$, i.e. $u(\mathbf{r}) \to 0$ as $|\mathbf{r}| \to \infty$. Noting that the

operator is self-adjoint, the Green's function for the modified Helmholtz operator with free-space boundary conditions satisfies the equation

$$(\nabla^2 - k^2)G(\mathbf{r}, \mathbf{r}') = \delta(\mathbf{r} - \mathbf{r}'), \tag{1.5}$$

where $\delta(\mathbf{r} - \mathbf{r}')$ is the Dirac-delta function [42] and $G(\mathbf{r}, \mathbf{r}')$ is also subject to free-space boundary conditions, as discussed below. Note that $G(\mathbf{r}, \mathbf{r}') = G(\mathbf{r} - \mathbf{r}')$ for constant coefficient differential operators such as the modified Helmholtz operator. We write Eq. (1.5) in spherical coordinates $(r, \theta, \phi)$ centered at the singular point $\mathbf{r}'$, where $r = \sqrt{|\mathbf{r} - \mathbf{r}'|^2}$, $\theta$ is the polar angle, and $\phi$ the azimuthal angle. Since the modified Helmholtz operator is rotationally invariant, we enforce radial symmetry on the Green's function $G(\mathbf{r} - \mathbf{r}') = g(r)$, and the Helmholtz equation reduces to

$$\left(\frac{1}{r^2}\frac{d}{dr}\left(r^2\frac{d}{dr}\right) - k^2\right) g(r) = \delta(r). \tag{1.6}$$

For $r > 0$, $g(r)$ satisfies the homogeneous equation

$$\left(\frac{1}{r^2}\frac{d}{dr}\left(r^2\frac{d}{dr}\right) - k^2\right) g(r) = 0, \tag{1.7}$$

which simplifies to

$$\frac{d^2}{dr^2}g(r) + \frac{2}{r}\frac{d}{dr}g(r) - k^2 g(r) = 0. \tag{1.8}$$

This is a Bessel equation of order zero, with general solution

$$g(r) = C_1\frac{e^{-kr}}{r} + C_2\frac{e^{kr}}{r}. \tag{1.9}$$

The free-space boundary condition forces $C_2 = 0$, and $C_1$ can be determined from the normalization condition,

$$\int_{\mathcal{B}} \delta(\mathbf{r} - \mathbf{r}')d\mathbf{r} = 1, \tag{1.10}$$

for any ball $\mathcal{B}$ containing the singular point $\mathbf{r}'$. In particular, taking a ball of arbitrary radius $\epsilon$ centeres at $\mathbf{r}'$, substituting from Eq. (1.5), and using the divergence theorem yields

$$1 = \int_{\mathcal{B}_\epsilon} \delta(\mathbf{r} - \mathbf{r}')d\mathbf{r} = \int_{\mathcal{B}_\epsilon} (\nabla^2 - k^2)G(\mathbf{r}, \mathbf{r}')d\mathbf{r} = \int_{\partial\mathcal{B}_\epsilon} \frac{\partial G}{\partial n}(\mathbf{r}, \mathbf{r}')d\sigma - k^2\int_{\mathcal{B}_\epsilon} G(\mathbf{r}, \mathbf{r}')d\mathbf{r} \tag{1.11a}$$

$$= C_1\int_{\partial\mathcal{B}_\epsilon} \frac{e^{-k|\mathbf{r}-\mathbf{r}'|}}{|\mathbf{r} - \mathbf{r}'|}\left(-k - \frac{1}{|\mathbf{r} - \mathbf{r}'|}\right)d\sigma - C_1k^2\int_{\mathcal{B}_\epsilon} \frac{e^{(-k|\mathbf{r}-\mathbf{r}'|)}}{|\mathbf{r} - \mathbf{r}'|}d\mathbf{r}. \tag{1.11b}$$

Evaluating the surface integral on $\mathcal{B}_\epsilon$ gives

$$\int_{\partial\mathcal{B}_\epsilon} \frac{e^{-k|\mathbf{r}-\mathbf{r}'|}}{|\mathbf{r}-\mathbf{r}'|}\left(-k-\frac{1}{|\mathbf{r}-\mathbf{r}'|}\right)d\sigma = 4\pi e^{-k\epsilon}\left(-1-k\epsilon\right). \tag{1.12a}$$

Evaluating the volume integral gives

$$-k^2\int_{\mathcal{B}_\epsilon}\frac{e^{(-k|\mathbf{r}-\mathbf{r}'|)}}{|\mathbf{r}-\mathbf{r}'|}d\mathbf{r} = 4\pi e^{-k\epsilon}\left(1+k\epsilon-e^{k\epsilon}\right). \tag{1.13}$$

Hence, Eq. (1.11b) becomes

$$1 = 4\pi C_1 e^{-k\epsilon}\left(-1-k\epsilon+1+k\epsilon-e^{k\epsilon}\right), \tag{1.14}$$

giving $C_1 = -1/4\pi$ and

$$G(\mathbf{r},\mathbf{r}') = \frac{-e^{-k|\mathbf{r}-\mathbf{r}'|}}{4\pi|\mathbf{r}-\mathbf{r}'|}. \tag{1.15}$$

Now that the Green's function is derived we will use it to analytically invert the modified Helmholtz equation in Eq. (1.4) with the following procedure.

**Step 2: Analytically inverting the differential operator**. The Green's function $G(\mathbf{r},\mathbf{r}')$ can be used to analytically invert the modified Helmholtz equation (1.4) with the following procedure. Multiplying through by the Green's function and integrating yields

$$\int_{\mathbb{R}^3} G(\mathbf{r},\mathbf{r}')(\nabla^2-k^2)u(\mathbf{r}')d\mathbf{r}' = \int_{\mathbb{R}^3} G(\mathbf{r},\mathbf{r}')f(\mathbf{r}')d\mathbf{r}'. \tag{1.16}$$

Repeated integration by parts gives

$$\int_{\mathbb{R}^3}(\nabla^2-k^2)G(\mathbf{r},\mathbf{r}')u(\mathbf{r}')d\mathbf{r}'+\int_\Gamma\left(G(\mathbf{r},\mathbf{r}')\nabla u(\mathbf{r}')-\nabla G(\mathbf{r},\mathbf{r}')u(\mathbf{r}')\right)\cdot\mathbf{n}d\Gamma = \int_{\mathbb{R}^3}G(\mathbf{r},\mathbf{r}')f(\mathbf{r}')d\mathbf{r}', \tag{1.17}$$

where $\mathbf{n}$ is the outward pointing normal vector, and the boundary $\Gamma$ is taken to infinity. The free-space boundary condition on $u(\mathbf{r})$ ensures the first expression in the boundary term is zero since $u(\mathbf{r})\to 0$ as $\mathbf{r}\to\infty$ implies $\nabla u(\mathbf{r})\cdot\mathbf{n}\to 0$ as $\mathbf{r}\to\infty$. Similarly, the imposed free-space boundary conditions on $G(\mathbf{r},\mathbf{r}')$, i.e. $G(\mathbf{r},\mathbf{r}')\to 0$ as $|\mathbf{r}-\mathbf{r}'|\to\infty$, ensures that the second expression in the boundary term is also zero, leaving

$$\int_{\mathbb{R}^3}(\nabla^2-k^2)G(\mathbf{r},\mathbf{r}')u(\mathbf{r}')d\mathbf{r}' = \int_{\mathbb{R}^3}G(\mathbf{r},\mathbf{r}')f(\mathbf{r}')d\mathbf{r}'. \tag{1.18}$$

Then, since $(\nabla^2 - k^2)G(\mathbf{r}, \mathbf{r}') = \delta(\mathbf{r} - \mathbf{r}')$, the left hand side reduces to

$$\int_{\mathbb{R}^3} u(\mathbf{r}')\delta(\mathbf{r} - \mathbf{r}')d\mathbf{r}' = u(\mathbf{r}), \tag{1.19}$$

and hence Eq. (1.18) simplifies to

$$u(\mathbf{r}) = \int_{\mathbb{R}^3} G(\mathbf{r}, \mathbf{r}')f(\mathbf{r}')d\mathbf{r}'. \tag{1.20}$$

The solution $u(\mathbf{r})$ to the modified Helmholtz boundary value problem in Eq. (1.4) is given in Eq. (1.20) as a convolution with the Green's function and the inhomogeneous term $f(\mathbf{r})$. The solution given by Eq. (1.20) is exact, however in general it must be evaluated numerically which presents two significant practical challenges. First, the convolution must be evaluated accurately. This is made difficult by both the singularity in the Green's function in Eq. (1.15) and any complexity in the inhomogeneous term $f(\mathbf{r})$. In particular, in the electronic structure method developed in this work, the inhomogeneous term is comprised of functions with cusps and singularities. We achieve sufficient accuracy with a combination of methods including adaptive mesh refinement, higher order quadrature rules, and singularity subtraction schemes. Second, once the convolution is accurately discretized, the discrete sums must be evaluated efficiently. Naive evaluation costs $O(N^2)$ work per convolution for an $N$-point discretization. Fast summation methods evaluate the discrete sums in subquadratic work, but introduce approximation errors. To achieve a good balance of accuracy and efficiency we implement and use a GPU accelerated treecode based on barycentric interpolation, described below.

Provided the convolution can be evaluated accurately and efficiently, the method of Green's functions results in a viable alternative approach to boundary value problems for differential operators. As described in Chapter 3, this technique can be applied to Kohn-Sham Density Functional Theory to convert the eigenvalue problem for the Kohn-Sham Hamiltonian into an equivalent fixed-point problem for an integral operator. This reformulation enables the development of new numerical methods for KS-DFT, those based on integral operators and fixed-point problems rather than differential operators and eigenvalue problems. Given that improved numerical methods will expand the scientific impact of KS-DFT, exploring this alternative Green's function approach is valuable, as the challenges faced by the approaches differ and there may be advantages to the Green's function method.

### 1.3 Numerical Method: Treecode-Accelerated Discrete Convolutions

The analytic inversion using the method of Green's functions results in the convolution integral in Eq. (1.20) that must be evaluated numerically. The details of the discretizations used for the

electronic structure calculations are described in Chapters 3 and 4. For now, we introduce the treecode algorithm that accelerates the evaluation of the discrete sum following the discretization. Consider an $N$-point discretization of the convolution integral in Eq. (1.20), which gives the discrete sum,

$$u_i = \sum_{\substack{j=1 \\ j \neq i}}^{N} G(\mathbf{r}_i, \mathbf{r}_j) f_j w_j, \qquad i = 1, 2, \ldots, N, \tag{1.21}$$

where $\mathbf{r}_j$ and $w_j$ are the quadrature points and weights, $f_j = f(\mathbf{r}_j)$, and $u_i \approx u(\mathbf{r}_i)$. Direct evaluation of this sum requires $O(N^2)$ operations. Accurate discretization of the continuous integral can require large $N$, especially for higher dimensional calculations such as those in $\mathbb{R}^3$ considered throughout this work, and the direct sum becomes prohibitively expensive. Therefore subquadratic-scaling fast summation approximations are essential for the success of Green's function methods.

Several hierarchical fast summation methods are available for computing approximations to the direct sum in Eq. (1.21) in subquadratic work. Two common tree-based methods are the treecode, developed by Barnes and Hut [43], and the Fast Multipole Method, developed by Greengard and Rokhlin [44, 45]. Both algorithms are viable for this application; we employ a recently developed treecode based on barycentric interpolation because it is well suited for acceleration using GPU coprocessors. An overview of the key ideas behind the Barnes-Hut treecode is introduced here, while Chapter 2 describes in more detail the higher-accuracy version based on barycentric interpolation, as well as our progress in implementing this version for heterogeneous high performance computing architectures, specifically distributed compute nodes containing GPU coprocessors.

Treecodes were initially developed by computational astrophysicists for N-body gravitational simulations and are also used for electrostatics interactions. For an intuitive description of the treecode, the algorithm is presented in the context of interacting charged particles where $G(\mathbf{r}_i, \mathbf{r}_j)$ represents the interaction potential, rather than for the discrete convolution sum. Consider a set of particles with positions $\mathbf{r}_j$ and electric charges $q_j$. Equation (1.21), which was initially defined in the context of a discretized convolution integral, also gives the the electrostatic potential $u_i$ at each particle, after setting $f_j w_j = q_j$. Denote $\{\mathbf{r}_i\}$ as the *target* particles at which the potential is to be computed, and $\{\mathbf{r}_j\}$ as the *source* particles which contribute to the potential. The treecode begins by partitioning the source particles into a hierarchical tree of clusters, as diagrammed in 1-dimension in Fig. 1.1. The left side shows the particles and how they are partitioned at each level of the tree, and the right side shows the corresponding tree representation. Level 0, also called the root cluster, consists of all the source particles. Level 1 is constructed by partitioning the root into two child clusters, each containing a subset of the source particles. Subsequent levels are constructed by recursive partitioning. In this diagram, the partitioning is terminated at level 2, which represents the leaves of the tree.

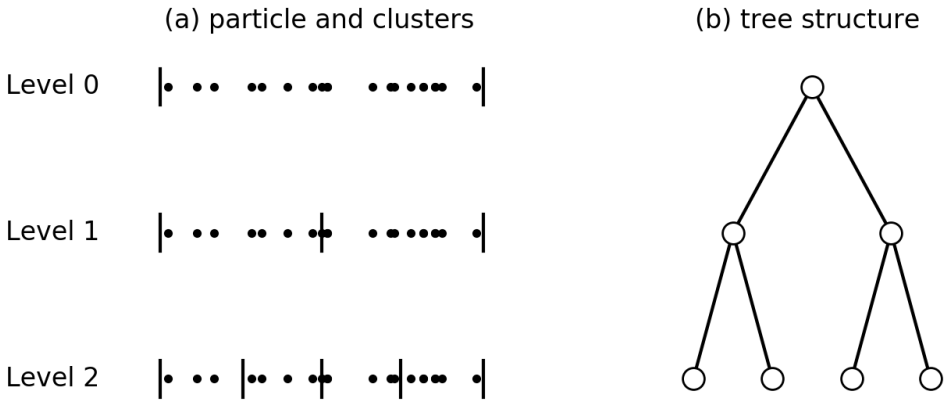


Figure 1.1: An example of tree partitioning of a set of particles in 1-dimension; (a) shows the particles and cluster partitions, (b) shows the corresponding hierarchical tree structure. Level 0 is the root of the tree and consists of all the particles. Level 1 divides the particles into two clusters, and level 2 into four clusters. In this example the level 2 clusters are the leaves of the tree.

The treecode then replaces particle-particle interactions with particle-cluster approximations using a recursive procedure described in detail in Chapter 2. For now it is sufficient to know that this procedure results in each target particle interacting with $O(\log N)$ clusters, hence the complexity for evaluating the sum for all $N$ target particles is $O(N \log N)$.

There are several options for computing the particle-cluster approximation. The first, and simplest approximation represents each cluster as a monopole. The sum of charges of all source particles in the cluster is assigned to a fictitious monopole particle at the cluster's center of mass. Figure 1.2 diagrams one such monopole particle-cluster approximation. In Fig. 1.2(a), a target particle interacts directly with each source particle in the source cluster, which involves computing three independent interactions. In Fig. 1.2(b), the target particle instead interacts with the fictitious monopole particle, which involves computing only one interaction, regardless of the number of source particles contained in the cluster. This reduction in computation comes at the cost of introducing an approximation error. One way to control the approximation error is to use higher accuracy cluster representations. In particular, the monopole approximation can be viewed as a first order approximation within a class of higher order methods based on multipole expansions [46, 47, 48, 49]. Alternatively, Chapter 2 introduces a recently developed class of higher order approximations based on barycentric interpolation [50, 51, 52], which is the version of the treecode developed and used throughout this work.

The remainder of the chapters are organized as follows. Chapter 2 presents the development of a GPU-accelerated and MPI-parallelized barycentric treecode library called BaryTree, publicly available on GitHub at github.com/Treecodes/BaryTree. Chapters 3 and 4 present the development

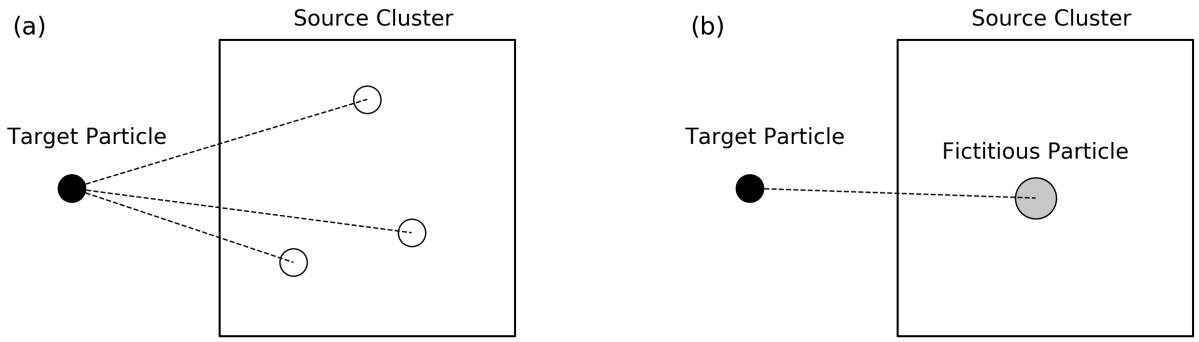


Figure 1.2: Monopole particle-cluster approximation. (a) the original interaction between a target particle (solid black circle) with a cluster of source particles (box containing open circles). (b) the monopole approximation, where the interactions with each individual source particle are replaced by a single interaction with the fictitious particle representing the cluster at the center of mass.

of the electronic structure application called Treecode-Accelerated Green Iteration (TAGI) that employs BaryTree to evaluate discrete convolutions. Specifically, Chapter 3 presents TAGI for all-electron calculations, in which the true nuclear potential is used for each atom, and wavefunctions for all of the electrons are explicitly computed. Chapter 4 then presents the extension of TAGI to pseudopotential calculations, in which the core electrons are absorbed into the nucleus, resulting in smooth and non-singular nuclear potentials, and only the wavefunctions of the valence electrons are computed. Chapter 5 summarizes the progress so far and provides several paths forward to further improve this integral equation method for Kohn-Sham Density Functional Theory.

# Chapter 2

# BaryTree: GPU-Accelerated Barycentric Treecodes

This chapter describes a GPU-accelerated and MPI-parallelized treecode for fast summation of long-range particle interactions based on barycentric Lagrange [50] and barycentric Hermite [51] interpolation. It follows closely the paper "A GPU-Accelerated Barycentric Lagrange Treecode," authored by Nathan Vaughn, Leighton Wilson, and Robert Krasny, which will appear in the Proceedings of the 21st IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing (PDSEC-2020) [52]. In addition to the kernel-independent barycentric Lagrange treecode (BLTC) presented in the paper, this chapter presents the weakly kernel-dependent barycentric Hermite treecode (BHTC). The code is publicly available on GitHub at github.com/Treecodes/BaryTree, as both a standalone executable and a library, with examples.

Discrete sums of the form given below in Eq. (2.1) occur in a variety of scientific computing applications. In the Kohn-Sham DFT application presented in this work, these sums occur following discretization of convolution integrals. However these sums also occur in applications that compute particle interactions, for example gravitational simulations or electrostatic calculations. The fast summation algorithm presented below applies to both discretized convolutions and interacting particles; for clarity, we describe the algorithm in terms of interacting charged particles.

## 2.1 Introduction

The calculation of particle interactions is essential in many areas of computational physics, for example computing gravitational or electrostatic potentials and forces. In a system with $N$ particles, the cost of direct summation scales like $O(N^2)$ and is often prohibitively slow, but the computations can be accelerated using improved hardware and algorithms. Direct summation has been implemented on graphics processing units (GPUs) with significant speedup, for example 25x over an optimized CPU implementation [53] and 250x over a portable C implementation [54]. The GPU implementation of direct summation enables simulations of larger problems but does not improve the algorithmic scaling. On the other hand several algorithms with subquadratic scaling are available, for example the fast multipole method (FMM) [44] and the treecode [43], which extend

the accessible problem sizes much further than the hardware speedup alone.

**Related previous work.** Algorithmic and hardware advances can be combined to accurately and efficiently compute larger $N$-body calculations. Parallelizing the hierarchical algorithms is significantly more complex than parallelizing the direct sum, but, substantial progress has been made throughout the years. The kernel-independent FMM has been parallelized on heterogeneous architectures and applied to systems of 30 billion particles by Biros and co-workers, [55, 56, 57], running on hundreds of thousands of CPU cores and up to 192 GPUs, using OpenMP and CUDA for intra-node parallelization. Darve and others in 2012 ported the most critical steps of the black-box FMM to GPUs using CUDA [58]. More recently, Fortin and Touche [59] have implemented a dual tree traversal code on AMD and Intel accelerators using OpenCL.

The Barnes–Hut treecode has been implemented on GPUs and scaled to billions of particles [60, 61, 62]. Hamada *et al.*'s 2009 Gordon Bell entry introduced CUDA implementations of a treecode for a gravitational $N$-body simulation and FMM for a turbulence simulation that ran on 256 GPUs and achieved at the time unprecedented performance. Bédorf *et al.* in 2012 presented a CUDA Barnes-Hut gravitational $N$-body code known as Bonsai running entirely on the GPU [61]. In 2014 Bonsai was used in a Milky Way Galaxy simulation that ran on 18600 GPUs on ORNL Titan. Burtscher and Pingali in 2011 presented a CUDA implementation of a Barnes-Hut $N$-body algorithm in *GPU Computing Gems* which focused on replacing the pointer-chasing recursion present in many CPU treecodes with iterating over array structures [63]. Yokota and Barba implemented a GPU FMM and treecode with multipole expansions for simulating leapfrogging vortex rings [64].

In addition to multipole expansions, treecodes can be extended to higher accuracy regimes by approximating a particle-cluster interaction with a Cartesian Taylor series expansion [46, 47, 49]. More recently polynomial interpolation for approximating particle-cluster interactions has been investigated [50], using the barycentric Lagrange form of the interpolating polynomial with second kind Chebyshev points [65]. Unlike Taylor series expansions, this approach is independent of the functional form of the interaction kernel, requiring only the function values of the kernel at the interpolation points. We note here that the concept of kernel-independent FMMs has been well developed by Biros, Darve, and others [66, 67, 68, 58, 69, 56, 57, 70]. The barycentric interpolation approach can be further extended to Hermite polynomials in a weakly kernel-dependent approach, requiring only the function values and $2^d - 1$ partial derivatives of the kernel at the interpolation points for a $d-$dimensional calculation [51].

**Present work.** The focus of this work is the development of a scalable implementation of the barycentric treecodes for modern high performance computing architectures. In particular, we employ an MPI + OpenACC framework that uses MPI for inter-node parallelization and OpenACC for intra-node acceleration on GPUs. For completeness, we include a shared memory parallelzation using OpenMP that is used to collect some benchmark results and is used in Chapter 3. We

implement the kernel-independent Barycentric Lagrange Treecode (BLTC) and the weakly kernel-dependent Barycentric Hermite Treecode (BHTC) in this framework, both of which are available in the BaryTree library. We compare these implementations to their CPU counterparts for the Coulomb kernel and demonstrate significant GPU acceleration. We also compare the BLTC and BHTC, and identify the regimes where each is preferred. Lastly, we demonstrate the parallel scalability on up to 32 GPUs, computing the interactions between 64 million particles in 16.2 seconds with an $L_2$ error of 5.9e-6, while maintaining 83% parallel efficiency.

The remainder of this chapter is organized as follows. Section 2.2 describes the two treecodes based on barycentric interpolation. Section 2.3 describes the shared memory implementation with OpenMP, the distributed memory implementation with MPI, and the GPU acceleration with OpenACC. Section 2.4 presents numerical results for the BLTC and BHTC implementations for test systems consisting of randomly distributed particles. Section 2.5 discusses the results of the MPI + OpenACC implementation, addresses several areas for further improvement, and describes a plan to extend this implementation to cluster-particle and cluster-clusters barycentric treecodes.

## 2.2 Barycentric Treecodes

This section describes two forms of the treecode, the Barycentric Lagrange Treecode (BLTC) and the Barycentric Hermite Treecode (BHTC), which are based on the barycentric forms of the Lagrange and Hermite interpolating polynomials. The treecodes are fast summation algorithms for the N-body interaction sums of the form

$$\varphi(\mathbf{x}_i) = \sum_{j=1}^{N} G(\mathbf{x}_i, \mathbf{y}_j) f_j, \quad i = 1, \ldots, N, \tag{2.1}$$

which arise in gravitational simulations, electrostatics, and discretized convolution integrals. Eq. (2.1) can be viewed as a set of particle-particle interactions, and direct evaluation required $O(N^2)$ operations. The treecodes described below compute fast approximations to $\varphi(\mathbf{x}_i)$ in $O(N \log N)$ operations. In particular, these treecodes approximate interactions between a target particle and a cluster of source particles using formulas derived from barycentric Lagrange and barycentric Hermite interpolation. As will be shown below, this form of the particle-cluster approximation is well suited for GPU acceleration. We begin with a description of this approximation for both Lagrange and Hermite interpolation, then in Section 2.2.4 describe the algorithm that makes use of these approximations to reduce the computational complexity to $O(N \log N)$.

### 2.2.1 Barycentric Lagrange Formulation

The BLTC is based on the barycentric form of the Lagrange interpolating polynomial. Given a function $f(x)$ evaluated at $n$+1 points $s_k$, the Lagrange interpolating polynomial is

$$p_n(x) = \sum_{k=0}^{n} f(s_k) L_k(x), \tag{2.2}$$

where $L_k(x)$ is given in barycentric form by

$$L_k(x) = \frac{a_k(x)}{\sum_{\ell=0}^{n} a_\ell(x)}, \quad k = 0, \ldots, n, \tag{2.3}$$

where

$$a_k(x) = \frac{w_k}{x - s_k}, \quad w_k = \frac{1}{\prod_{j=0, j\neq k}^{n} (s_k - s_j)}. \tag{2.4}$$

The $x = s_k$ singularity does not pose a problem in interpolation, and likewise does not pose a problem in the particle-cluster approximations that follow. The handling of the singularity will be discussed in Section 2.2.3.

The BLTC uses Chebyshev points of the second kind due to their good interpolation properties [71, 65]. For the interval $[-1, 1]$, the Chebyshev points of the second kind are given by

$$s_k = \cos\theta_k, \quad \theta_k = \pi k/n, \quad k = 0, \ldots, n, \tag{2.5}$$

and their corresponding interpolation weights are given by

$$w_k = (-1)^k \delta_k, \tag{2.6}$$

where $\delta_k = 1/2$ if $k = 0$ or $n$, and $\delta_k = 1$ otherwise. We note that the interpolation weights $w_k$ are $O(1)$ for Chebyshev points, independent of the interpolation degree, whereas for uniformly spaced points, the interpolation weights vary exponentially in the degree and lead to numerical instabilities. Furthermore, polynomial interpolation experiences the Runge phenomenon, where pointwise errors grow near the boundaries as the degree increases, unless the distribution of interpolation points approaches a density of $1/\sqrt{1-x^2}$ as $n \to \infty$ [65]. However, Chebyshev points have exactly this property, avoiding the Runge phenomenon that occurs for uniform points and bounding the interpolation error uniformly over the interval.

In many cases $G(\mathbf{x}, \mathbf{y})$ is singular at $\mathbf{x} = \mathbf{y}$, as in the Coulomb kernel $\frac{1}{|\mathbf{x}-\mathbf{y}|}$ and the Yukawa kernel $\frac{e^{-\kappa|\mathbf{x}-\mathbf{y}|}}{|\mathbf{x}-\mathbf{y}|}$ used in this work. However, away from the singularity, $G(\mathbf{x}, \mathbf{y})$ is smooth and can

be approximated locally with a polynomial. As a demonstration, consider the 1D case of a target particle located at $x = 0$, a cluster $C$ on the interval $[1, 3]$, and the kernel $G(x, y) = \frac{1}{|x-y|}$. For now it is sufficient to know that the cluster $C$ contains a localized set of source particles $y^j$, and the target particle $x$ is not contained within the bounds of the cluster. The cluster can be represented by a set of interpolation points, and the kernel $G(x, y)$ can be accurately represented by interpolating polynomials inside $C$, given by

$$G(x, y) \approx \sum_{k=0}^{n} G(x, s_k) L_k(y), \tag{2.7}$$

where $s_k$ are a set of interpolation points inside the cluster. Figure 2.1 demonstrates this kernel interpolation in 1D. Figure 2.1(a) shows the target particle at $x = 0$ (closed circle), the source cluster $C$ from $[1, 3]$ (dashed lines), and a set of interpolation points inside the cluster (open circles). Figure 2.1(b) shows the kernel $G(x, y)$ (solid black curve) as well as the interpolating polynomials of degree 1, 4 and 7 (blue, orange, green dashed curves). Inside the cluster, the interpolating polynomials converge to $G(x, y)$. Figure 2.1(c) shows the interpolation errors for each degree, demonstrating the convergence to $G(x, y)$ as the degree of interpolation is increased, as well as the property of Chebyshev interpolation that the interpolation error is bounded uniformly over the interval; there is no Runge phenomenon near the boundaries.

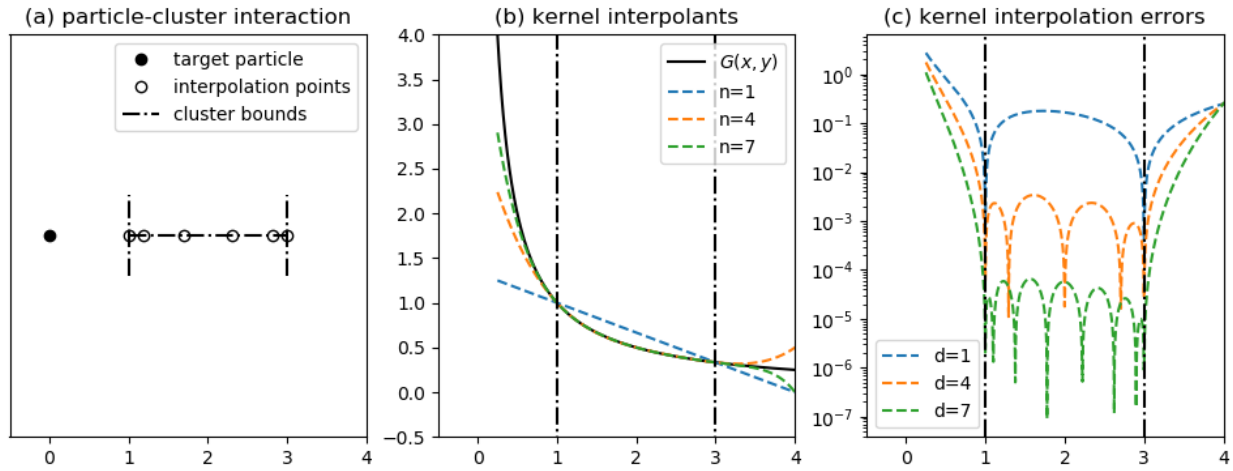


Figure 2.1: 1-dimensional example of interpolating the kernel $G(x, y) = \frac{1}{|x-y|}$ inside a cluster $C$ on the interval $[1, 3]$ due to a source point at $x = 0$; a) diagrams the particle-cluster interaction, showing a target particle at $x = 0$ (closed circle), a cluster on $[1, 3]$ (dot-dashed lines), and interpolation points in the cluster (open circles), b) shows the kernel $G(x, y)$ (solid black curve), and three interpolating polynomials of degree 1, 4, and 7 (dashed colors), c) shows the interpolation errors, which decay inside the cluster as the degree increases from 1 to 4 to 7.

Similarly, in 3D, consider a target particle $\mathbf{x} = (x_1, x_2, x_3)$ and a cluster $C$ of source particle

$\mathbf{y}_j$. Each source particle $\mathbf{y}_j$ has position $(y_1^j, y_2^j, y_3^j)$ and charge $f^j$. The potential $\varphi$ at $\mathbf{x}$ due to the particle-cluster interaction is given by

$$\varphi(\mathbf{x}, C) = \sum_{\mathbf{y}^j \in C} G(\mathbf{x}, \mathbf{y}^j) f^j. \tag{2.8}$$

The kernel $G(\mathbf{x}, \mathbf{y})$ can be approximated using a tensor product of $(n+1)^3$ interpolation points $\mathbf{s_k} = (s_{k_1}, s_{k_2}, s_{k_3})$ with the barycentric interpolation formula,

$$G(\mathbf{x}, \mathbf{y}) \approx \sum_{k_1,k_2,k_3} G(\mathbf{x}, \mathbf{s_k}) L_{k_1}(y_1) L_{k_2}(y_2) L_{k_3}(y_3), \tag{2.9}$$

where the sum over $k_1, k_2, k_3$ is performed in each index for $k_i = 0, \ldots, n$ for interpolation degree $n$. Using the interpolation points $\mathbf{s_k}$ and the barycentric Lagrange interpolation from Eq. (2.9), the particle-cluster interaction in Eq. (2.8) can be approximated by

$$\varphi(\mathbf{x}, C) \approx \sum_{\mathbf{y}^j \in C} \sum_{k_1,k_2,k_3} G(\mathbf{x}, \mathbf{s_k}) L_{k_1}(y_1^j) L_{k_2}(y_2^j) L_{k_3}(y_3^j) f^j. \tag{2.10}$$

Importantly, the summation order can be changed to

$$\varphi(\mathbf{x}, C) \approx \sum_{k_1,k_2,k_3} G(\mathbf{x}, \mathbf{s_k}) \widehat{f}^k, \tag{2.11}$$

where $\widehat{f}^k$, the modified weights, are given by

$$\widehat{f}^k = \sum_{\mathbf{y}^j \in C} L_{k_1}(y_1^j) L_{k_2}(y_2^j) L_{k_3}(y_3^j) f^j. \tag{2.12}$$

There are two important consequences of this rearrangement. First, each $\widehat{f}^k$ is independent of the target particle and can be precomputed, stored, and reused for all particles that interact with this cluster. Second, the structure of the approximation in Eq. (2.11) is the same as the structure of the direct calculation in Eq. (2.8), where in the first case the target interacts with the Chebyshev points and in the latter with the source particles. In both cases, the interactions are independent and can be computed simultaneously. This structure is essential to the efficient GPU implementation described in Section 2.3.3.

### 2.2.2 Barycentric Hermite Formulation

The BHTC [51] is based on the barycentric form of the Hermite interpolating polynomial. Given a function $f(x)$ and its derivative $f'(x)$ evaluated at $n+1$ points $s_k$, the Hermite interpolating

polynomial is

$$p_n(x) = \sum_{k=0}^{n} \Big( f(s_k) H_k(x) + f'(s_k) \widetilde{H}_k(x) \Big), \tag{2.13}$$

where $H_k(x)$ and $\widetilde{H}_k(x)$ in barycentric form are

$$H_k(x) = \frac{b_k(x)}{\sum_{\ell=0}^{n} b_\ell(x)}, \quad \widetilde{H}_k(x) = \frac{c_k(x)}{\sum_{\ell=0}^{n} b_\ell(x)}, \tag{2.14}$$

where

$$b_k(x) = \frac{u_k}{x - s_k} + \frac{v_k}{(x - s_k)^2}, \quad c_k(x) = \frac{v_k}{x - s_k}. \tag{2.15}$$

Using Chebyshev points of the second kind, the interpolation weights are given by

$$u_k = \begin{cases} \dfrac{s_k}{1 - s_k^2}, & k = 1 : n-1 \\ -\frac{1}{12}\left(1 + 2n^2\right), & k = 0 \\ \frac{1}{12}\left(1 + 2n^2\right), & k = n \end{cases}, \quad v_k = \delta_k^2, \tag{2.16}$$

where, again, $\delta_k = 1/2$ if $k = 0$ or $n$, and $\delta_k = 1$ otherwise. Similar to Eq. (2.9), a kernel $G(\mathbf{x}, \mathbf{y})$ can be approximated using interpolation points $\mathbf{s_k} = (s_{k_1}, s_{k_2}, s_{k_3})$ with the Hermite interpolation

$$\begin{aligned} G(\mathbf{x}, \mathbf{y}) \approx \sum_{k_1,k_2,k_3} \Big[ & H_{k_1}(y_1) \Big( H_{k_2}(y_2) \Big( H_{k_3}(y_3) G(\mathbf{x}, \mathbf{s_k}) + \widetilde{H}_{k_3}(y_3) G_3(\mathbf{x}, \mathbf{s_k}) \Big) \\ & + \widetilde{H}_{k_2}(y_2) \Big( H_{k_3}(y_3) G_2(\mathbf{x}, \mathbf{s_k}) + \widetilde{H}_{k_3}(y_3) G_{23}(\mathbf{x}, \mathbf{s_k}) \Big) \Big) \\ & + \widetilde{H}_{k_1}(y_1) \Big( H_{k_2}(y_2) \Big( H_{k_3}(y_3) G_1(\mathbf{x}, \mathbf{s_k}) + \widetilde{H}_{k_3}(y_3) G_{13}(\mathbf{x}, \mathbf{s_k}) \Big) \\ & + \widetilde{H}_{k_2}(y_2) \Big( H_{k_3}(y_3) G_{12}(\mathbf{x}, \mathbf{s_k}) + \widetilde{H}_{k_3}(y_3) G_{123}(\mathbf{x}, \mathbf{s_k}) \Big) \Big) \Big], \end{aligned} \tag{2.17}$$

where we denote the partial derivatives $\frac{\partial G(\mathbf{x}, \mathbf{s_k})}{\partial x_1}$ by $G_1(\mathbf{x}, \mathbf{s_k})$, $\frac{\partial^2 G(\mathbf{x}, \mathbf{s_k})}{\partial x_1 x_2}$ by $G_{12}(\mathbf{x}, \mathbf{s_k})$, and so forth. The method requires all mixed partial derivatives of $G(\mathbf{x}, \mathbf{y})$ of degree zero or one in the $x$, $y$, and $z$ dimensions resulting in eight terms. As in the Lagrange case, we approximate the particle-cluster interaction in Eq. (2.8) by interpolating the kernel and rearranging the sums, giving

$$\begin{aligned} \varphi(\mathbf{x}, C) \approx \sum_{k_1,k_2,k_3} \Big( & G(\mathbf{x}, \mathbf{s_k}) \widehat{f}^k + G_1(\mathbf{x}, \mathbf{s_k}) \widehat{f}_1^k + G_2(\mathbf{x}, \mathbf{s_k}) \widehat{f}_2^k + G_3(\mathbf{x}, \mathbf{s_k}) \widehat{f}_3^k \\ & + G_{12}(\mathbf{x}, \mathbf{s_k}) \widehat{f}_{12}^k + G_{13}(\mathbf{x}, \mathbf{s_k}) \widehat{f}_{13}^k + G_{23}(\mathbf{x}, \mathbf{s_k}) \widehat{f}_{23}^k + G_{123}(\mathbf{x}, \mathbf{s_k}) \widehat{f}_{123}^k \Big). \end{aligned} \tag{2.18}$$

where the modified weights are given by

$$\begin{aligned}
\widehat{f}^k &= \sum_{\mathbf{y}_j \in C} H_{k_1}(y_1^j) H_{k_2}(y_2^j) H_{k_3}(y_3^j) f^j, \quad \widehat{f}_1^k = \sum_{\mathbf{y}_j \in C} \widetilde{H}_{k_1}(y_1^j) H_{k_2}(y_2^j) H_{k_3}(y_3^j) f^j, \\
\widehat{f}_2^k &= \sum_{\mathbf{y}_j \in C} H_{k_1}(y_1^j) \widetilde{H}_{k_2}(y_2^j) H_{k_3}(y_3^j) f^j, \quad \widehat{f}_3^k = \sum_{\mathbf{y}_j \in C} H_{k_1}(y_1^j) H_{k_2}(y_2^j) \widetilde{H}_{k_3}(y_3^j) f^j, \\
\widehat{f}_{12}^k &= \sum_{\mathbf{y}_j \in C} \widetilde{H}_{k_1}(y_1^j) \widetilde{H}_{k_2}(y_2^j) H_{k_3}(y_3^j) f^j, \quad \widehat{f}_{13}^k = \sum_{\mathbf{y}_j \in C} \widetilde{H}_{k_1}(y_1^j) H_{k_2}(y_2^j) \widetilde{H}_{k_3}(y_3^j) f^j, \\
\widehat{f}_{23}^k &= \sum_{\mathbf{y}_j \in C} H_{k_1}(y_1^j) \widetilde{H}_{k_2}(y_2^j) \widetilde{H}_{k_3}(y_3^j) f^j, \quad \widehat{f}_{123}^k = \sum_{\mathbf{y}_j \in C} \widetilde{H}_{k_1}(y_1^j) \widetilde{H}_{k_2}(y_2^j) \widetilde{H}_{k_3}(y_3^j) f^j.
\end{aligned} \tag{2.19}$$

Similar to Eq. (2.12) for BLTC, we can precompute the modified weights for the Hermite approximations. While the structure of the approximation in Eq. (2.18) is not identical to the direct interaction in Eq. (2.8), as was the case for the BLTC, the BHTC still benefits from the fact that the interactions between a target particle and each of the interpolation points are independent and can be computed simultaneously, enabling an efficient parallel implementation of the BHTC.

### 2.2.3 Interpolation Singularities

Both the Lagrange and Hermite modified weights in Eq. (2.12) and Eq. (2.19) are singular when one of the coordinates of a source particle is coincident with one of the coordinates of a Chebyshev point ($y_i^j = s_{k_i}$ for $i = 1$, 2, or 3). As will be discussed below, when generating clusters, we use the minimal bounding box surrounding the particles, thereby guaranteeing that some particles share coordinates with some interpolation points resulting in singularities. However, these singularities can be handled in a straightforward way.

Consider the 1D case of $x$ coincident with $s_k$ for the set of Lagrange polynomials $L_j(x)$ given in Eq. (2.3). Taking the limit as $x \to s_k$ gives

$$\lim_{x \to s_k} L_j(x) = \lim_{x \to s_k} \frac{\frac{w_j}{x - s_j}}{\sum_{\ell=0}^{n} \frac{w_\ell}{x - s_\ell}} = \frac{w_j}{w_k} \lim_{x \to s_k} \frac{x - s_k}{x - s_j} = \begin{cases} 1 & j = k, \\ 0 & j \neq k. \end{cases} \tag{2.20}$$

Thus, when computing the Lagrange modified weights for a cluster, we check for each particle $j$ if a Chebyshev point $\mathbf{s_k}$ is coincident with that particle in any Cartesian direction $i$. If so, we set all terms $L_{\ell_i}(y_i^j)$ for $\ell_i = 0, \ldots, n$ to zero, except for $\ell_i = k_i$, which we set to 1.

Now, consider the 1D case of an $x$ coincident with $s_k$ for the set of Hermite polynomials $H_j(x)$

and $\widetilde{H}_j(x)$ given in Eq. (2.2.2). Taking the limit as $x \to s_k$, we get

$$\begin{aligned}\lim_{x\to s_k} H_j(x) &= \lim_{x\to s_k} \frac{\frac{u_j}{x-s_j} + \frac{v_j}{(x-s_j)^2}}{\sum_{\ell=0}^{n} \frac{u_\ell}{x-s_\ell} + \frac{v_\ell}{(x-s_\ell)^2}} \\ &= \frac{v_j}{v_k} \lim_{x\to s_k} \frac{(x-s_k)^2}{(x-s_j)^2} = \begin{cases} 1 & j = k \\ 0 & j \neq k, \end{cases}\end{aligned} \tag{2.21}$$

for the polynomials $H_j(x)$, and

$$\begin{aligned}\lim_{x\to s_k} \widetilde{H}_j(x) &= \lim_{x\to s_k} \frac{\frac{v_j}{x-s_j}}{\sum_{\ell=0}^{n} \frac{u_\ell}{x-s_\ell} + \frac{v_\ell}{(x-s_\ell)^2}} \\ &= \frac{v_j}{v_k} \lim_{x\to s_k} \frac{(x-s_k)^2}{x-s_j} = 0.\end{aligned} \tag{2.22}$$

for the polynomials $\widetilde{H}_j(x)$. Similar to the Lagrange case, if particle $j$ is coincident with $\mathbf{s_k}$ in direction $i$, we set all terms $H_{\ell_i}(y_i^j)$ and $\widetilde{H}_{\ell_i}(y_i^j)$ for $\ell_i = 0, \ldots, n$ to zero, except for $H_{k_i}(y_i^j)$, which we set to 1.

### 2.2.4 Treecode Description

**Source Clusters and Target Batches**. The treecode begins by constructing a hierarchical tree of source clusters with leaf clusters containing fewer than $N_L$ particles. The root cluster is the minimal bounding box containing all source particles. The root is recursively divided into child clusters where the recursion terminates when a cluster contains $N_L$ or fewer particles. The cluster division occurs at the midpoint of the three dimensions of the bounding box. This is diagrammed in 2D in Fig. 2.2 for a set of 300 non-uniformly distributed particles inside a ball of radius 1. A quadtree is constructed with $N_L = 10$. Figure 2.2(a) shows the root of the tree, a single cluster containing all 300 particles; Fig. 2.2(b) shows the tree with maximum depth 1, Fig. 2.2(c) shows the tree with maximum depth 2, and so on through maximum depth 5. At each level of the tree, a cluster is divided if it has more than $N_L$ particles and is not yet at the prescribed maximum depth.

We make a minor modification to the tree construction described above that improves efficiency. After division of a cluster, the bounding box of each child is shrunk to the minimal bounding box containing the particles. Shrinking improves the accuracy of the approximations, because the Chebyshev points represent a smaller volume, however this technique can result in clusters with bad aspect ratios. To avoid this, a cluster can divide into only 4 or 2 children if dividing into 8 would cause poor aspect ratios.

Following the source tree construction, the treecode constructs a set of localized target batches

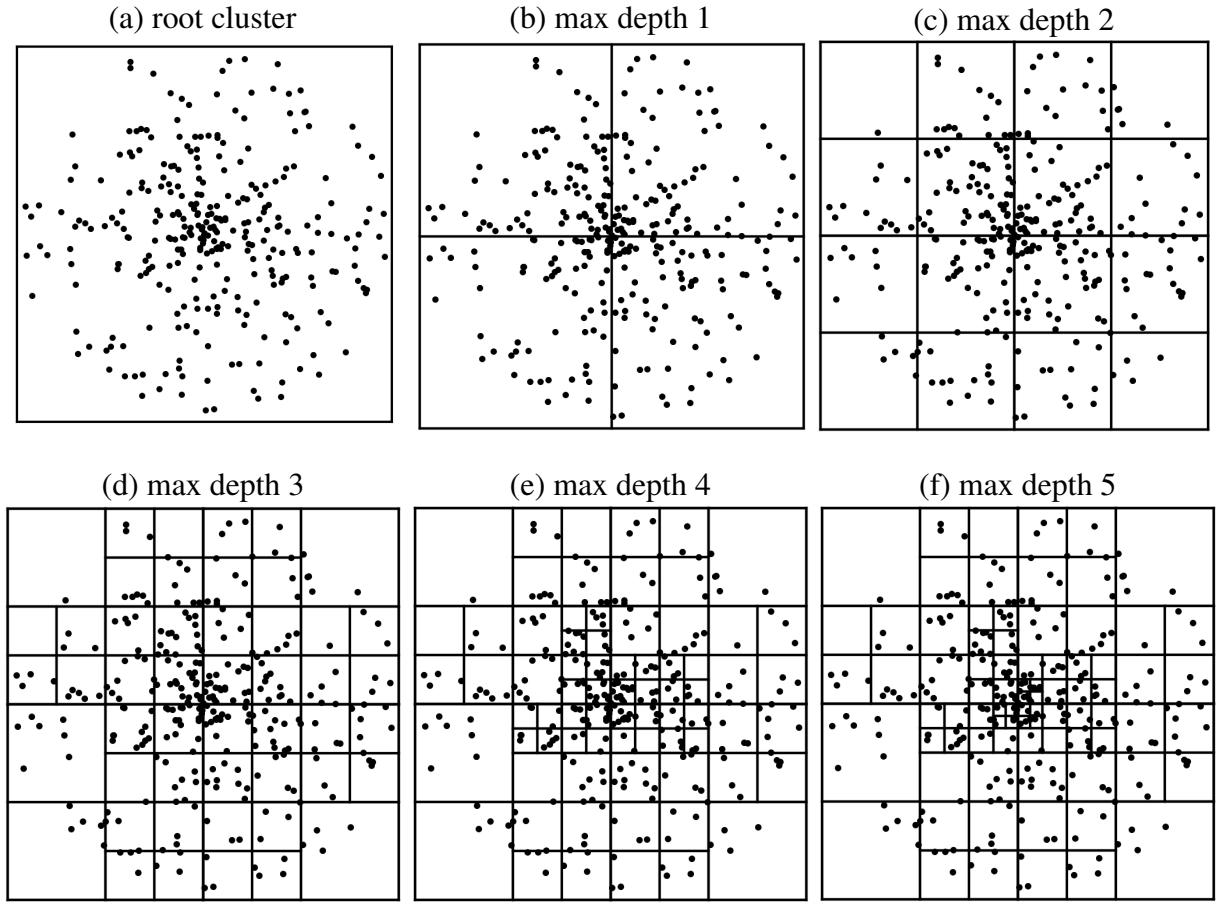


Figure 2.2: 2D tree construction diagram for 300 particles distributed inside a ball of radius 1, with the highest density at the center of the ball. At each level of the tree, any clusters containing more than $N_L = 10$ particles are divided into four children. (a) shows the root of the tree, consisting of the whole set of particles; (b) shows the tree of maximum depth 1, (c) shows the tree of maximum depth 2, and so on.

containing fewer than $N_B$ target particles per batch. In practice, we use the same partitioning routine described above for the source particles, however we only retain the leaf clusters (which will be the target batches), rather than retaining the whole hierarchical tree. The effect of target batching on the GPU implementation efficiency will be discussed below.

**Multipole Acceptance Criteria**. The particle-particle interactions in Eq. (2.1) are reorganized into batch-cluster interactions. For a given batch-cluster interaction, the approximation in Eq. (2.11) or Eq. (2.18) is used if the Multipole Acceptance Criteria (MAC) are accepted, which in this work are given by

$$\frac{r_B + r_C}{R} < \theta, \quad h(n+1)^3 < N_C, \tag{2.23}$$

where $r_B$ is the radius of the target batch, $r_C$ is the radius of the source cluster, $R$ is the distance between the batch and cluster centers, $\theta$ is the user-defined MAC parameter, $h$ is a prefactor depending on the approximation used (Lagrange or Hermite), $n$ is the interpolation degree, and $N_C$ is the number of source particles in the cluster. The first criterion $(r_B + r_C)/R < \theta$ ensures the accuracy of the approximation and is diagrammed in Fig. 2.3. The second criterion, $h(n+1)^3 < N_C$, is the cluster size checking which will be explained below, and ensures the efficiency of the approximation.

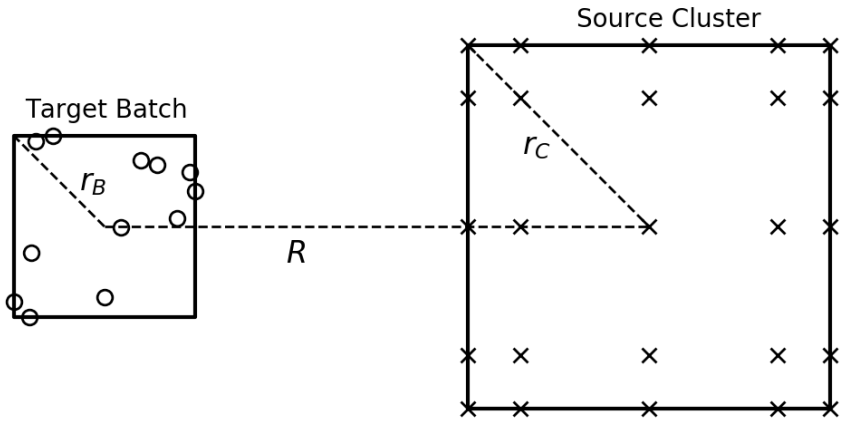


Figure 2.3: A 2D schematic of the particle-cluster interaction. The target batch of radius $r_B$ containing randomly distributed target particles (open circles) separated by a distance $R$ from a source cluster of radius $r_C$ containing Cartesian product of Chebyshev interpolation points ($\times$'s). The batch-cluster approximation will be accepted if $(r_B + r_C)/R < \theta$ for the user input MAC parameter $\theta$.

**Algorithm**. The treecode algorithm given in Algorithm 2.1 is the same for the BLTC and BHTC, the only difference being in their cluster approximations. The input consists of the particle data and treecode parameters. The output consists of the approximate potential. Line 4 constructs the

hierarchical tree of source clusters and set of localized target batches. In lines 5-6, the modified weights are computed for each cluster using Eq. (2.12) for BLTC and Eq. (2.19) for BHTC. In lines 7-8, each target batch interacts with the root cluster via the recursive function COMPUTEPOTENTIAL. There are three options within the COMPUTEPOTENTIAL function. If the MAC is satisfied, then the batch-cluster approximation is computed (Eq. (2.11) for BLTC, Eq. (2.18) for BHTC). If the MAC fails because $(r_B + r_C)/R \geq \theta$ then there are two possibilities: if the cluster is a leaf then the batch interacts directly with the cluster by Eq. (2.8), otherwise COMPUTEPOTENTIAL is called recursively for each of the cluster's children. If the MAC fails because $h(n+1)^3 \geq N_C$ then the interaction is computed directly by Eq. (2.8). This algorithm computes approximations to the potential in $O(N \log N)$ operations compared to the $O(N^2)$ direct summation of Eq. (2.1).

**Algorithm 2.1** Treecode

```
1:  input: particle data x_i, y_i, f_i, i = 1, ..., N
2:  input: treecode parameters θ, n, N_L, N_B
3:  output: approximate potential φ_i, i = 1, ..., N
4:  build tree of clusters {C} and set of batches {B}
5:  for each source cluster do
6:      compute weights in Eq. (2.12) or (2.19)
7:  for each target batch do
8:      COMPUTEPOTENTIAL(B, root_cluster)
9:
10: function COMPUTEPOTENTIAL(Batch, Cluster)
11:     if MAC is satisfied then
12:         compute approximation by Eq. (2.11) or (2.18)
13:     else if (r_B + r_C)/R ≥ θ then
14:         if Cluster is a leaf then
15:             compute interaction by direct sum in Eq. (2.8)
16:         else
17:             for each Child of Cluster do
18:                 COMPUTEPOTENTIAL(Batch, Child)
19:     else if h(n+1)^3 ≥ N_C then
20:         compute interaction by direct sum in Eq. (2.8)
```

### 2.3 Implementation Details

This section presents the implementation details of BaryTree for (1) shared memory systems, (2) distributed memory systems, and (3) GPUs. The shared memory parallelization uses OpenMP, the distributed memory parallelization uses MPI, and the GPU acceleration uses OpenACC. Both the shared and distributed memory implementations are combined with the GPU implementation by assigning one GPU per OpenMP thread, or one GPU per MPI rank, accordingly.

### 2.3.1 Shared Memory OpenMP Implementation

We begin by describing a shared memory implementation of BaryTree using OpenMP. It is significantly simpler than the distributed memory parallelization described below, but is limited to a single compute node. While scaling to modern high performance computing architectures demands the distributed memory approach, we include this shared memory parallelization as it was a precursor to the distributed memory version and was used throughout Chapter 3 for all-electron Treecode-Accelerated Green Iteration calculations.

**Modified Charges**. To compute the modified charges we loop over each cluster in the tree (Alg. 2.1, lines 5-6) and compute the expressions in Eq. (2.12) or (2.19). These calculations are independent for each cluster; therefore we can parallelize the loop with `#pragma omp parallel for`. For a tree containing $N_T$ clusters and a calculation using $t$ OpenMP threads, this assigns the first $N_T/t$ clusters to the first thread, the next $N_t/t$ clusters to the second thread, and so on. However, we note that the cost of computing modified charges is not equal for all clusters, and this naive distribution of the loop can result in load imbalance. In particular, for a cluster containing $N_C$ particles and $(n+1)^3$ interpolation points, computing the modified charges involves $O(N_C(n+1)^3)$ operations. Clusters near the top of the tree contain many more particles than those near the leaves of the tree, therefore there is a significant difference in the cost of computing the modified charges. To mitigate this load imbalance, we use OpenMP guided scheduling, `#pragma omp parallel for schedule(guided)`. This approach divides the loop into many "chunks," then assigns one chunk of the loop to each thread. When a thread completes its chunk it returns to the scheduler and is assigned another chunk, until all chunks are complete. With guided scheduling the chunk size decreases over time; specifically, the chunk size is proportional to the number of remaining loop iterations divided by the number of threads. Therefore, as the calculation progresses, the chunk sizes become smaller and smaller, until each chunk corresponds to only a single cluster. This scheduling pattern alleviates load imbalances that occur due to the non-uniform cost of each iterate in the loop; by assigning chunks upon completion, different threads may be assigned a different number of chunks, however to total work performed by each thread will remain balanced.

**Batch-Cluster Interactions**. We parallelize the main compute phase by distributing the batch-cluster interactions among the OpenMP threads. We loop over each target batch (Alg. 2.1, lines 7-8), and for each target batch we compute all of its batch-cluster interactions. For this shared memory parallelization, we distribute the target batches over the OpenMP threads using the same pragma as before, `#pragma omp parallel for schedule(guided)`. Hence, each thread will be assigned chunks of target batches. The thread will compute all interactions for these target batches, then request a new chunk, until the work is complete. Again, there can be an imbalance in the work required for each target batch, as the number of interactions per batch depends both on the location of the batch and the distribution of particles. However this potential imbalance is again mitigated

(a) four partitions (b) six partitions

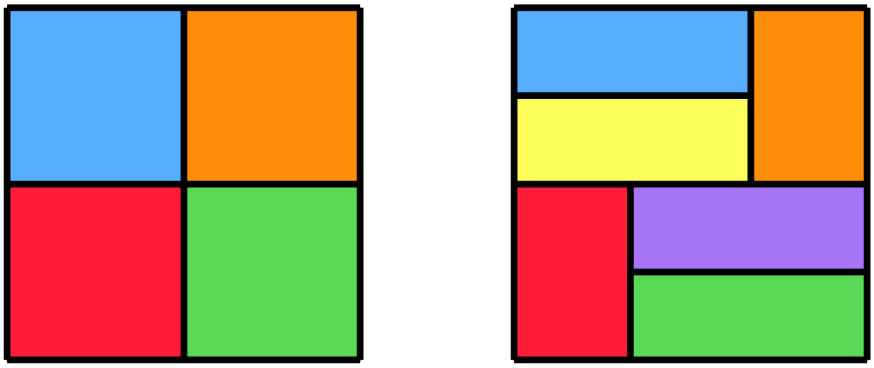

Figure 2.4: A diagram of the recursive coordinate bisection of the unit square for four and six processes. Coordinate bisections occur in the $y$ coordinate then $x$ coordinate, repeating until the target number of partitions is achieved. Each begins with a bisection of the $y$ coordinate at $y = 0.5$, assigning half the processors to the top and half to the bottom region. The second bisections differ depending on how many processes were assigned to each partition after the first bisection. The area owned by each process in (a) is 1/4, the area owned by each process in (b) is 1/6.

through the use of scheduling.

This shared memory parallelization is suitable for CPU or GPU calculations, where each OpenMP thread is assigned to either one CPU-core or one GPU. In the case of the GPU parallelization, the GPUs do no share memory locations, so the data is copied onto each GPU on the node. In the event that the data fits on a single compute node, but not within a single GPU, we can instead use the distributed memory MPI implementation described below.

### 2.3.2 Distributed Memory MPI Implemetation

In this section we describe the MPI implementation of the barycentric treecodes. We first describe the distributed memory framework based on Local Essential Trees (LETs) [72], then describe our MPI implementation using Remote Memory Access (RMA).

**Locally Essential Trees.** BaryTree uses recursive coordinate bisection (RCB) for domain decomposition and Locally Essential Trees [72] (LET) for reduced-order global communication. RCB proceeds by recursively partitioning the domain with a hyperplane that (1) is perpendicular to one of the coordinate axes and (2) balances the number of particles with the number of ranks for each side of the partition. Fig. 2.4 diagrams a 2D RCB decomposition of a unit square into four and six partitions, where the area assigned to each partition is balanced. Following the recursive coordinate bisection of the domain, each processor owns particles in a local and compact region

of the domain and constructs its local source tree. The union of all processors' source trees is the global source tree, which is never constructed but is a useful concept in the discussion to follow. The key observations about LETs are (1) that each target particle interacts with only a portion of the global source tree, and (2) that nearby target particles interact with similar sub-trees. A processor's LET is the union of the interaction sub-trees for all of its target particles. This LET accounts for all the remote data that must be acquired by the processor. By construction, the LET will contain only $O(\log N)$ remotely owned clusters, with neighboring processors exchanging many clusters and well-separated processors exchanging few clusters. Hence, while the construction of the LETs is an all-to-all communication pattern, the amount of data that must be obtained by each processor only grows logarithmically with the problem size.

**Remote Memory Access.** We use MPI passive target synchronization Remote Memory Access (RMA) to perform the construction and communication of the LETs. First introduced in the MPI-2 standard, RMA provides a one-sided communication model within MPI. In MPI one-sided operations, an origin process can *put* or *get* data on or from a target process, through specially declared memory *windows*, with no active involvement from the target process. In active target synchronization, or active RMA, the target process sets bounds on when its windows can be accessed; in passive target synchronization, or passive RMA, the target process sets no limitations on accesses to its windows, and instead the origin process *locks* the target window to perform operations on it. Passive RMA in particular is similar in spirit to the partitioned global address space (PGAS) model used in languages like UPC. In our implementation, we use passive RMA to communicate data between processes.

We perform the construction of the LETs in two steps, which we describe for a two process example. The tree arrays, which contain cluster midpoints and radii for all tree nodes, the source particles, and the cluster weights on both processes are all contained within RMA windows which can be accessed by other processes. In the first step, process 1 gets the tree array (containing cluster midpoints and radii) from process 2 and creates interaction lists, and vice versa. These interaction lists consist of all clusters on process 2 that a target particle on process 1 interacts with directly or via the approximation. Then, in the second step, process 1 uses the newly constructed interaction lists to get the necessary source particle and cluster weight data from process 2, filling process 1's LET. Simultaneously, process 2 gets the data from process 1 to build its LET. At the conclusion of the second step, every process contains all the data needed to perform its calculation, and each process proceeds to compute the potential at its target particles as in the serial case.

Using RMA provides a few potential advantages over traditional sends and receives. Depending on the underlying distribution of particles in the entire domain, the data that one process needs from another can be highly irregular in size. In addition, the cluster weights and source particles needed by a process from another will not be contiguous in the memory of the target process. In a standard

round-robin approach for two-sided communication, all pairs of communicating processes must wait for the most expensive communication to complete before each process can communicate with its next partner. Even with asynchronous sends and receives, both the sender and receiver must know the size and layout of the data on both sides of the transfer. In the passive RMA approach, we can asynchronously launch all of our communications with no input from the target process, and the origin process need only know the layout of the data to be received from the target process. There is no need to pack the data on the sender's side so that it can be received in a contiguous buffer on the receiver's side. Additionally, if the network hardware exists, the application can use Remote Direct Memory Access (RDMA) to execute the get operations more efficiently, with no need for any involvement from the target process CPU. However, despite these advantages, we note that the startup cost for creating the RMA windows is non-negligible, and for small problem sizes it may be more efficient to use two-sided communication and avoid the RMA window creation cost. We note further that our specific implementation of LET communication is not necessarily more efficient than a well-designed two-sided communication scheme that does not require communication between all nodes. However, we believe a more efficient scheme could also potentially benefit from using passive RMA as describe above.

### 2.3.3 GPU Implementation

This section describes the GPU implementation details of the BLTC and BHTC. Specifically, it describes two algorithmic modifications called target batching and cluster size checking, and several OpenACC implementation details involving host and device memory management, GPU compute kernels, and asynchronous streams. The GPU implementation is extended to multiple GPUs in a straightforward manner using the MPI implementation described above with one MPI rank per GPU.

**Target Batching.** To efficiently use GPUs it is important to saturate them with enough work to occupy as many of the compute units as possible. We accomplish this by simultaneously processing the interactions between a batch of target particles and a source cluster. The treecode was originally posed without target batching [43], allowing for each target particle to follow its own path through the tree based on its own evaluations of the MAC criterion. Attempting to parallelize this on a GPU would result in thread divergence, where different threads follow different logic paths, diminishing performance. To avoid this issue, while also saturating the GPU with work, we make use of the observation that nearby targets interact with nearly the same sub-trees. We group nearby target particles into batches, where a batch contains all the target particles in some cuboid, and apply the MAC in Eq. (2.23) to the entire batch, as opposed to making individual decisions for each target in the batch. While this is not necessarily optimal for each individual target, it minimizes thread divergence. We find that this is advantageous for the GPU implementation.

To perform target batching, we use the same partitioning routines that build the octree of sources to build an octree of target particles. The set of leaves of this octree are the target batches.

**Cluster Size Checking.** The second condition in the MAC ($h(n+1)^3 < N_C$) introduces a cluster size check to avoid computing approximations that are more expensive than the original direct interaction. For example, for a target batch of $N_B$ particles interacting with a source cluster of $N_L$ particles and $(n+1)^3$ interpolation points, the direct interaction consists of $N_B N_L$ pairwise interactions and the approximate interaction consists of $N_B(n+1)^3$ pairwise interactions. For large enough $n$ and small enough $N_L$, the approximation may be more expensive than the direct interaction. Therefore we introduce the second MAC condition, with a prefactor $h$ tuned for either Lagrange or Hermite.

For the Lagrange approximation, the targets interact with the interpolation points in Eq. (2.11) in exactly the same way that they interact with source particles in Eq. (2.8). Therefore, we set $h = 1$ and the MAC checks whether $N_L > (n+1)^3$. If this criteria is satisfied, then performing the approximation is cheaper than interacting with the cluster directly and the approximation is accepted. If this criteria fails, it is faster to compute the interaction directly. For the Hermite approximation, Eq. (2.18) involves many more floating point operations than the direct interaction. The value $h = 4$ was empirically determined to be a good choice in this case.

**Host and Device Data Management.** The host (CPU) and device (GPU) do not share the same memory. Data copying between host and device is expensive, so poor memory management can inhibit GPU acceleration. For a given MPI rank, all data movements are managed with OpenACC data regions. Data must be copied to and from the GPU twice. Algorithm 0 shows the steps for both MPI related communication and host-device communications, labelled HtD for host-to-device and DtH for device-to-host. First, in line 2 the source particles are copied onto the GPU, and in lines 3-4 the modified weights computed for each cluster. In line 5 these weights are then copied back to the CPU's RMA windows where other MPI ranks can access them during LET construction. The RMA windows into the local data are created in line 6. Lines 9-11 contain the MPI communication between ranks in order to construct the Locally Essential Trees. Then in line 12, following construction of the LET, the targets, LET sources, and LET clusters are copied onto the GPU before beginning the potential calculation. Finally, after all interactions are computed on the GPU in lines 13-14, the resulting potential is copied back to the CPU in line 15.

**Compute Kernels.** For clarity, we note that routines compiled to run on GPUs are called kernels, not to be confused with the interaction kernels $G(\mathbf{x}, \mathbf{y})$ that occur throughout this chapter. The GPU implementation uses four compute kernels, two during preprocessing and two during the tree evaluation. The preprocessing kernels are called once for each cluster and compute the modified weights given in Eq. (2.12) and Eq. (2.19). The tree evaluation kernels compute the interactions between a target batch and a source cluster, either via the cluster approximations in Eq. (2.11), (2.18)

**Algorithm 2.2** MPI + OpenACC BLTC

build tree of clusters $\{C\}$ and set of batches $\{B\}$ from local particles
HtD: copy source data
**for** each source cluster **do**
    compute modified charges $\widehat{f_{\mathbf{k}}}$ in Eq. (2.12) on GPU
DtH: copy modified charges
create MPI RMA windows to local data
**for** each remote rank **do**
    MPI: get tree arrays from remote rank
construct interaction lists from tree arrays
**for** each remote rank **do**
    MPI: get required particle and cluster data from remote rank and fill into LET
HtD: copy LET
**for** each target batch **do**
    COMPUTEPOTENTIAL($B$, LET_root) on GPU
DtH: copy final potential

or direct sum in Eq. (2.8). The kernels are generated with OpenACC directives, compiled with the PGI C compiler. For example, we enclose the compute regions with `#pragma acc kernels` and identify the parallelizable loops with `#pragma acc loop independent`.

From the hardware perspective, GPUs consist of multiple streaming multiprocessors (SM), each of which works concurrently and independently. Further, each SM consists of many compute cores, each of which works concurrently and shares a common memory bank in addition to their local registers. In the case of the NVIDIA Tesla P100 GPUs used in this chapter, each GPU has 60 SMs, each of which has 32 double-precision compute cores, for a total of 1,920 double-precision compute cores that can perform concurrent computations. From the software perspective, work is organized into thread-blocks and threads. A thread-block is executed by a streaming multiprocessor, and each thread is executed by a compute core. Next we describe how the various treecode computations are mapped to thread-blocks and threads to take advantage of this parallelism.

**Preprocessing Kernels.** We describe the two preprocessing kernels for the BLTC to show how the modified weights are calculated in Eq. (2.12), noting that the BHTC uses the same general procedure to compute each of the eight terms in Eq. (2.19). The first preprocessing kernel computes the intermediate quantity

$$\widetilde{f^j} = \frac{f^j}{\sum\limits_{k_1=0}^{n} a_{k_1}(y_1^j) \sum\limits_{k_2=0}^{n} a_{k_2}(y_2^j) \sum\limits_{k_3=0}^{n} a_{k_3}(y_3^j)}, \tag{2.24}$$

for each source particle in the cluster. The $j$th block is responsible for the $j$th source particle in the

cluster. Within each block, the threads parallelize over the interpolation degree $n$, computing each term of the three denominator sums simultaneously, followed by a reduction.

The second preprocessing kernel computes the modified weights for each of the interpolation points

$$\widehat{f}^k = \sum_{\mathbf{y}_j \in C} a_{k_1}(y_1^j) a_{k_2}(y_2^j) a_{k_3}(y_3^j) \widetilde{f}^j. \tag{2.25}$$

Each block is responsible for a single Chebyshev point, giving $(n+1)^3$ blocks. Within each block, the threads are parallelized over the source particles in the cluster, with the $j$th thread computing $a_{k_1}(y_1^j) a_{k_2}(y_2^j) a_{k_3}(y_3^j) \widetilde{f}^j$, followed by a reduction over the threads to compute $\widehat{f}^k$. If the number of source particles exceeds the number of threads per block in the kernel launch, each thread will be responsible for multiple source particles. This holds for the kernels below where the threads are parallelized over a cluster's source particles or interpolation points. For interpolation degree $n$ and a cluster containing $N_L$ source particles, the first preprocessing kernel performs $O((n+1)N_L)$ work and the second performs $O((n+1)^3 N_L)$ work for each cluster.

**Batch-Cluster Direct Kernel.** The first tree evaluation kernel is responsible for direct interactions between the target particles in a batch and the source particles in a cluster. This kernel is launched whenever the MAC fails. The kernel computes Eq. (2.8) for every target particle in the batch. This calculation involves an outer loop over the target particles in the batch and an inner loop over the source particles in the cluster. The outer loop is naturally parallelizable as the potential at different target particles is independent, while the inner loop can be parallelized with a reduction. The GPU kernel is structured as follows. The $i$th block is responsible for the $i$th target particle $\mathbf{x}_i$ in the batch. The threads are parallelized over the source particles in the cluster where the $j$th thread computes $G(\mathbf{x}_i, \mathbf{y}_j) f^j$, then a reduction is performed over the threads. For a batch containing $N_B$ target particles and a cluster containing $N_L$ source particles, this kernel call performs $O(N_L N_B)$ work.

The diagram in Fig. 2.5 shows the structure of the batch-cluster direct kernel. Fig. 2.5(a) shows an interaction matrix for a set of target batches and a set of source clusters. This is a visual aid for the structure of the work; this matrix is never formed. Each row corresponds to a target particle, which are organized into batches (bold horizontal partitions). Each column corresponds to a source particle, which are organized into clusters (bold vertical partitions). Highlighted in blue is the $(i, j)$ batch-cluster interaction which is computed by one launch of the batch-cluster direct kernel. Fig. 2.5(b) shows the structure of the kernel calculation. Each row corresponds to a target particle and is assigned to one block. Each element in the row corresponds to the interaction with one source particle and is assigned to a different thread. The green highlighted row corresponds to a single block which is executed on the streaming multiprocessor to compute the interactions for a single target particle. A reduction is performed along the row after all threads complete their interactions.

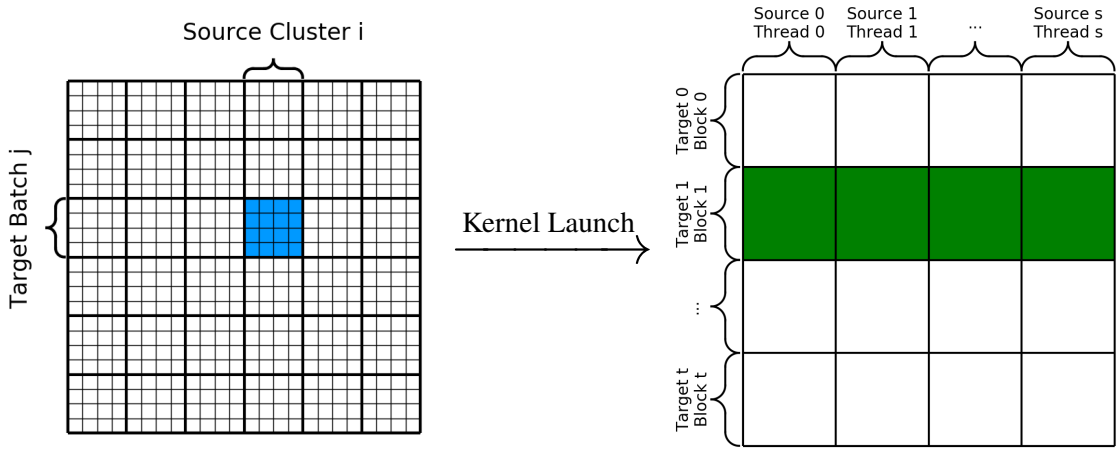


Figure 2.5: a) A diagram showing the GPU kernel launch pattern for the direct interactions. Highlighted in blue is the interaction between the $j$th target batch and the $i$th source cluster, which is computed by a single kernel launch on the GPU. (b) A diagram showing how the batch-cluster interaction is performed by the GPU. Highlighted in green is one block, which is computed in parallel on one single streaming multiprocessor.

**Batch-Cluster Approximation Kernel.** The second tree evaluation kernel is responsible for the approximate interactions between a batch of target particles and a cluster represented by interpolation points. This kernel is launched whenever the MAC passes for a batch of targets and a cluster of sources. For every target particle in the batch, the interaction is approximated with either Eq. (2.11) for BLTC or Eq. (2.18) for BHTC. Importantly, the structure of the approximate calculation is the same as the structure of the direct calculation, where the inner loop over source particles is replaced by an inner loop over Chebyshev points; and the diagram in Fig. 2.5 applies to this kernel as well. The GPU kernel is structured as follows. The $i$th block is responsible for the $i$th target particle $\mathbf{x}_i$ in the batch. The threads are parallelized over the $(n+1)^3$ Chebyshev points in the cluster where the $\mathbf{k}$th thread computes $G(\mathbf{x}_i, \mathbf{s_k})\widehat{f}^k$, then a reduction is performed over the threads to sum the potential at $\mathbf{x}_i$. For a batch containing $N_B$ target particles, this kernel call performs $O((n+1)^3 N_B)$ work.

The independent nature of the higher-order particle-cluster approximations is the distinguishing feature of the barycentric treecodes that enable efficient GPU implementations. The target particle can interact with each interpolation point simultaneously allowing for parallelization that is not possible in some other higher order approaches, such as Taylor treecodes which rely on fundamentally serial recurrence relations to compute high order approximations.

**Direct-Sum Kernel.** To compute reference values we also parallelized the direct sum on the GPU. The simplicity of the direct sum allows for easy and efficient parallelization with OpenACC

using just a single kernel around the nested loops over the $N$ target particles and $N$ source particles. Each block is responsible for one target particle, with the threads parallelized over the source particles. The single kernel call performs all $O(N^2)$ work.

**Asynchronous Streams.** During the treecode evaluation the CPU is responsible for looping through the interaction lists and launching the kernels on the GPU. When a kernel is launched, the calculation is performed on the GPU and the CPU waits to regain control until after the calculation completes. There are several sources of inefficiency in this process: 1) the CPU is sitting idle while the GPU is working, 2) there is an initialization cost associated with each kernel call before the GPU begins its calculation and 3) a single kernel might not saturate the GPU with work. The result is that the GPU is only performing calculations approximately 75-80% of the time during the calculation, with the remaining time spent waiting or initializing. We improve this by using asynchronous streams (`#pragma acc kernels async(streamID)`) which allows the CPU to queue the kernel on the GPU and immediately regain control without waiting for the calculation to complete. The CPU will then queue the next interaction from the interaction list on a different stream, before the first has completed. As we loop through the interaction lists we cycle `streamID` through the number of available streams, which, for the GPUs used in this work, is four. With this approach the initialization times are overlapped with the computation on other streams, reducing the GPU idle time. Furthermore, the GPU may decide to work on multiple streams at the same time if it has available resources, further improving efficiency. To handle memory access conflicts and race conditions, we use an atomic update (`#pragma acc atomic`) when updating the potential for a given target particle. Asynchronous streams significantly increase the performance of our GPU implementation. For example, in the 1 million particle test case that is described in Section 2.4, for the BLTC with the Coulomb kernel, $\theta = 0.7$, and $n = 7$, asynchronous streams reduce the computation time from 39.99 seconds to 29.56 seconds, a 35% improvement.

**Code Availability.** The code, as both a standalone executable and a library, with examples, is publicly available on GitHub at github.com/Treecodes/BaryTree.

### 2.4 Results

We demonstrate the barycentric treecodes on a series of test cases ranging from 1 million to 64 million particles. In each case the particles are randomly uniformly distributed in the $[-1, 1]^3$ cube, with charges randomly uniformly distributed on $[-1, 1]$. All reported times are wall clock time in seconds. For these tests, the targets and sources refer to the same set of particles, although the code is not restricted to this case. Further, we restrict results to the Coulomb kernel $1/(|\mathbf{x} - \mathbf{y}|)$ and Yukawa kernel $e^{-k|\mathbf{x}-\mathbf{y}|}/(|\mathbf{x} - \mathbf{y}|)$ for clarity and consistency, but note that the code is fully capable of treating more complex kernels. The kernel-independent BLTC can be used for any kernel, and

the weakly kernel-dependent can be used for any kernel for which the required partial derivatives are available.

**Single GPU vs. Single 6-Core CPU**. We begin by comparing the GPU implementations to portable CPU implementations for a $10^6$ particle test case. We use the Flux HPC Cluster at the University of Michigan to perform the calculations. The CPU calculations are run on a 6-core 2.67 GHz Intel Xeon X5650 processor using the OpenMP shared memory parallelization described in Section 2.3.1. The GPU calculations are run on a single NVIDIA Titan V.

Fig. 2.6 shows the computation time versus error in the potential for each of the cases considered. The error is the relative $L^2$ error measured with respect to the direct sum calculations, given by

$$E = \left( \sum_{i=1}^{N} (\varphi_i^{ds} - \varphi_i^{tc})^2 / \sum_{i=1}^{N} (\varphi_i^{ds})^2 \right)^{1/2} \tag{2.26}$$

where $\varphi_i^{ds}$ are the value computed by direct sum and $\varphi_i^{tc}$ are the value computed by the treecode. Fig. 2.6(a,b) shows the BLTC and BHTC for the Coulomb kernel, while Fig. 2.6(c,d) shows the BLTC and BHTC for the Yukawa kernel. The batch size and cluster size parameters are $N_B = N_C = 2000$. Each curve represents constant $\theta$ for $\theta$ in $\{0.5, 0.7, 0.9\}$, where the interpolation degree is swept from 1 to 14 or until machine precision is achieved, with solid lines corresponding to CPU results and dashed lines to GPU results. We draw several conclusions from the following figures: (1) on their respective architectures, the treecode outperforms the direct sum for 1M particles, (2) the BLTC and BHTC are both capable of achieving very high accuracy, (3) the GPU implementation tends to be at least 100x faster than the CPU for this problem size of 1 million particles on these architectures. We note that while the GPU direct sum is faster than the CPU treecode for this problem size, this will not be the case for large enough problems due to the $O(N^2)$ scaling of the direct sum.

**BLTC vs. BHTC**. We now perform a comparison of the BLTC and BHTC on system of 10 million particles interacting via the Coulomb kernel, $N_L = N_B = 4000$. Again, we hold $\theta$ fixed and increase the interpolation degree, measuring the time versus accuracy. We sweep $\theta$ from 0.5 to 0.9 and $p$ from 1 to 14, or until machine precision is reached. As in Fig. 2.6, we plot the time versus relative $L^2$ error. The optimal treecode parameters depend on the desired accuracy, evident by the intersection of curves in Fig. 2.6. For clarity in comparing the BLTC to BHTC, we plot the optimal envelope instead of the entire curves of constant $\theta$. A data point belongs to the optimal envelope if there are no treecode parameters that achieve better accuracy in less time. Fig. 2.7 shows this envelope for the 10 million particle test case. The BHTC is more efficient than the BLTC for accuracies of 1e-5 or better. The discrepancy grows for higher accuracies, resulting in up to a $2\times$ speedup before reaching machine precision.

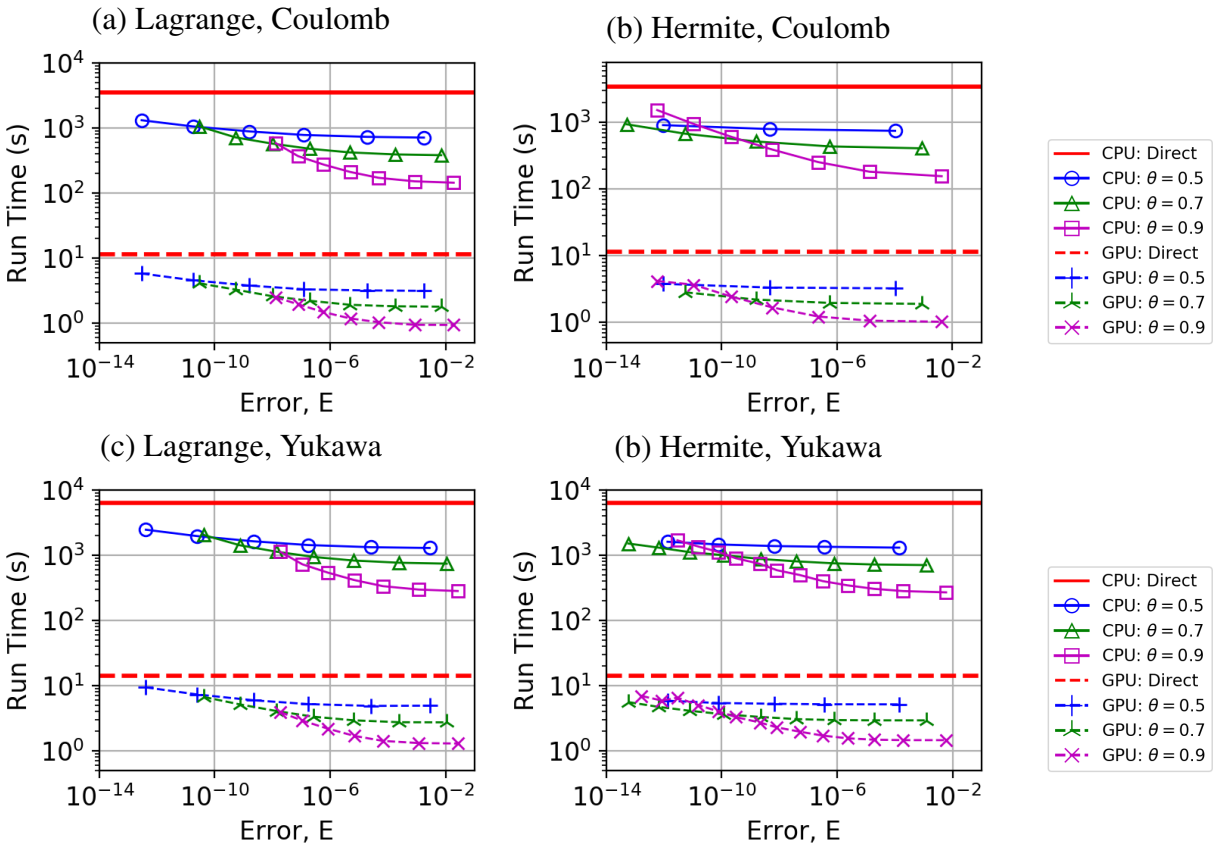


Figure 2.6: Run time versus accuracy for the 1 million random particle example, curves of constant $\theta$ for (a) the BLTC for the Coulomb kernel, (a) the BHTC for the Coulomb kernel, (a) the BLTC for the Yukawa kernel, and (d) the BHTC for the Yukawa kernel. 6-core 2.67 GHz Intel Xeon X5650 CPU results are shown with solid lines, NVIDIA Titan V GPU results with dashed lines.

We note that while the BHTC outperforms the BLTC in the high accuracy regime, it lacks the kernel-independence of the BLTC. If the kernel's partial derivatives are readily available and high accuracy is desired, the BHTC is preferred. Otherwise, the BLTC is preferred.

**Single node parallel scaling of GPU code**. This subsection documents the parallel efficiency of the BLTC on a single GPU node running with 1, 2 or 4 GPUs, using the shared memory OpenMP scheme described above in Section 2.3.1 to parallelize across GPUs with one thread assigned to each GPU. The test system has 10 million particles randomly located in a cube interacting via the Coulomb kernel. The work is divided into two stages; stage 1 encompasses the precomputing tasks in lines 5-6 of Algorithm 2.1 and stage 2 encompasses the batch-cluster computing in lines 7-8. Figure 2.8 shows the parallel efficiency of each stage and the entire computation as the number of GPUs increases from 1 to 4. The precompute stage scales less efficiently than the compute phase, due to some serial computation embedded in these tasks (85% on 2 GPUs, 63% on 4 GPUs), but this accounts for only a small fraction of the total computation time. The compute stage has close to

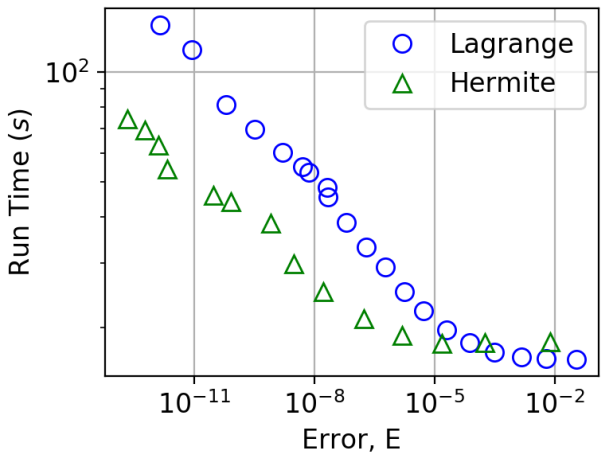


Figure 2.7: Comparing the BLTC and BHTC computation time versus accuracy for 10M particles with the Coulomb kernel. $\theta$ is swept from 0.5 to 0.9 and $p$ from 1 to 14 (or until machine precision is achieved), and the optimal envelopes are plotted.

ideal scaling (98% on 2 GPUs, 94% on 4 GPUs); moreover this stage accounts for a large fraction of the total computation time and therefore the treecode achieves 90% efficiency for the entire computation on 4 GPUs.

**Single node comparison of GPU and CPU codes**. We compare the treecode times running on a GPU node and a CPU node on Comet. The CPU calculations use a standard compute node with two Intel Xeon E5-2680v3 CPUs for a total of 24 cores, while the GPU calculations use a GPU node with four NVIDIA Tesla P100s. The CPU code is run with 24 OpenMP threads, one for each core, while the GPU code is run with 4 OpenMP threads, one for each GPU. To compare the nodes, calculations were performed for three problem sizes, $N = 10^5, 10^6, 10^7$, using the barycentric Lagrange treecode BLTC with MAC $\theta = 0.8$ interpolation degree $n = 8$, and cluster and batch sizes $N_L = N_B = 2000$. The treecode parameter settings yield 6-7 digit accuracy and the CPU and GPU implementations achieve identical errors.

Table 2.1 presents the computation run time and error for CPU and GPU nodes, for the Coulomb potential (left) and Yukawa potential (right, $\kappa = 0.5$). The GPU node achieves a speedup of $37\times$ over the CPU node for the Coulomb potential, and $49\times$ for the Yukawa potential; the speedup is larger for the Yukawa potential because in that case the GPU more efficiently handles the additional floating point operations required to evaluate the exponential function.

**Weak Scaling**. We demonstrate the weak scaling of the MPI + OpenACC treecode on Comet by holding the number of particles per GPU fixed and increasing the number of GPUs from 1 to 32. We use the BLTC with treecode parameters $\theta = 0.8$, $n = 8$, $N_L = N_B = 4000$. Fig. 2.9 shows the computation times for the Coulomb potential (dashed lines) and Yukawa potential (solid lines) with the number of particles per GPU set to 8, 16, and 32 million (circles, triangles, squares).

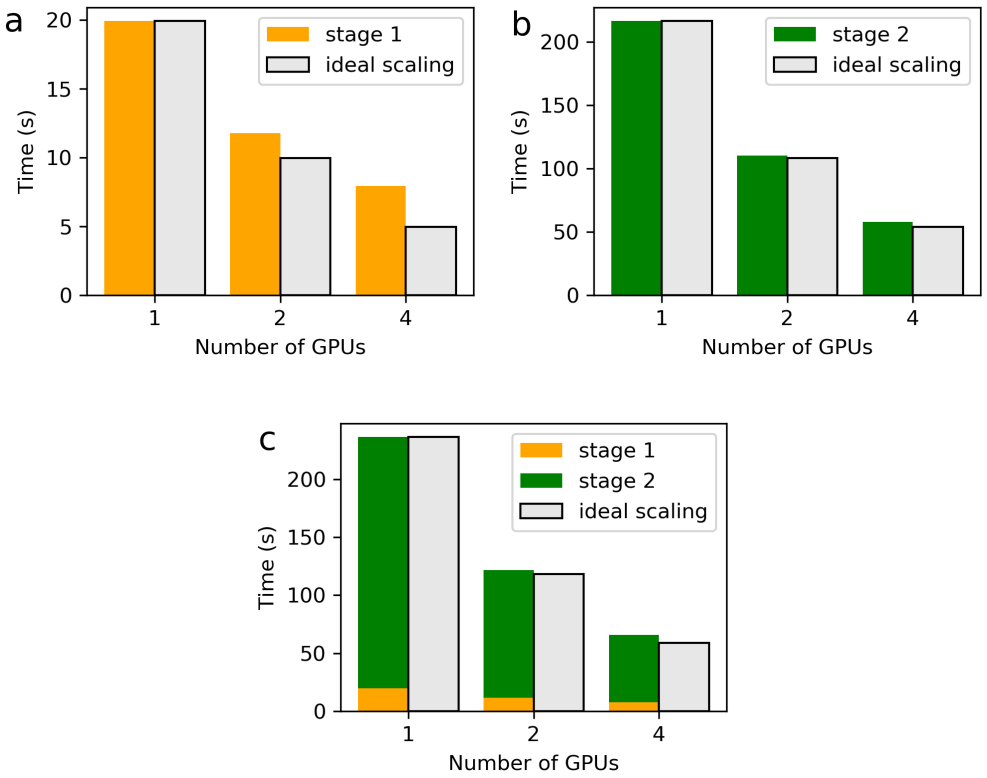


Figure 2.8: Shared memory parallel efficiency of BLTC on a single GPU node for 10 million particles interacting via Coulomb kernel. Treecode MAC parameter $\theta = 0.7$ and interpolation degree $n = 7$ yield treecode approximation error 2.31e-06 ($L_2$ error with respect to direct sum). The results show computation time (s) and ideal scaling time (s) using 1, 2 and 4 GPUs for (a) stage 1 (precompute), (b) stage 2 (compute), and (c) total time. For comparison, the direct sum time on 4 GPUs is 1668 s.

Importantly, as the problem size grows, the computation times only modestly increase. This is consistent with the expected scaling. Since the total work $W$ scales like $W \propto N \log N$, and the total number of particles $N$ is proportional to the number of ranks $p$, $N \propto p$, the total work per rank is $W/p \propto \log N \propto \log p$. Hence the work per rank grows logarithmically during the weak scaling study, in agreement with the observed results.

**Strong Scaling**. We demonstrate the strong scaling of MPI + OpenACC BaryTree on Comet using up to 32 NVIDIA P100 GPUs. The test systems consists of 16 million and 64 million particles interacting via the Coulomb and Yukawa kernels. We use the Lagrange treecode with parameters $\theta = 0.8$, $n = 8$, $N_L = N_B = 4000$, giving relative $L^2$ errors 4.0e-6 and 5.9e-6 respectively. Note that for these large systems the error was sampled at a random subset consisting of 10% of the target points. Fig. 2.10(a) and 2.11(a) show the strong scaling efficiency of these calculations by showing total time versus the number of GPUs. The efficiency is measured with respect to a single GPU and compared to ideal speedup (dotted lines). For the Coulomb kernel, as the number of

**Coulomb potential**

| $N$ | error, $E$ | CPU (s) | GPU (s) | speedup |
|---|---|---|---|---|
| $10^5$ | 1.44e-07 | 4.05 | 0.18 | 22.03 |
| $10^6$ | 5.31e-07 | 44.86 | 1.31 | 34.36 |
| $10^7$ | 1.22e-06 | 640.89 | 17.13 | 37.42 |

**Yukawa potential** ($\kappa = 0.5$)

| $N$ | error, $E$ | CPU (s) | GPU (s) | speedup |
|---|---|---|---|---|
| $10^5$ | 1.61e-07 | 4.73 | 0.17 | 27.35 |
| $10^6$ | 5.56e-07 | 74.68 | 1.51 | 49.43 |
| $10^7$ | 1.75e-06 | 1048.45 | 21.34 | 49.13 |

Table 2.1: Single node performance of the BLTC for Coulomb potential (left) and Yukawa potential (right, $\kappa = 0.5$), treecode parameters MAC $\theta = 0.8$ and degree $n = 8$ yielding 6-7 digit accuracy, results show CPU and GPU run times (s), calculations are performed on Comet using either a single CPU node (two Intel Xeon E5-2680v3 CPUs, total 24 cores) or a single GPU node (four NVIDIA Tesla P100s).

GPUs is increased to 32, the 64M particle example maintains higher efficiency (83%) than the 16M particle example (64%). The Yukawa kernel maintains slightly higher efficiencies of 84% and 73%. Figure 2.10(b) and 2.11(b) shows the distribution of time spent in each phase of the calculation as the number of GPUs increases from 1 to 32 for the 64M particle example. The setup phase (orange) includes the data movements and communication required for each rank to begin its local calculation. Specifically, the setup consists of organizing the local source particles into an octree and target particles into batches, the construction and communication of the LET, and the creation of the interaction lists for each target batch. The precompute phase (green) consists of computing the modified weights for each locally owned source cluster. The compute phase (blue) consists of computing the potential at each target particle. Each bar is colored based on the percent of time spent in each phase, with the total time listed atop the bar. Up to 32 ranks (2M particles/rank) the compute phase dominates the total time (>80% of total time). However, as the number of GPUs increases, the distribution of work shifts towards the setup and precompute phases. The fraction of time spent in the setup phase grows because the communication costs grow; more interactions are with remotely owned data that must be communicated. The fraction of time spent in the precompute phase grows because the modified weight kernels do not saturate the GPUs with work as the number of particles per rank decreases.

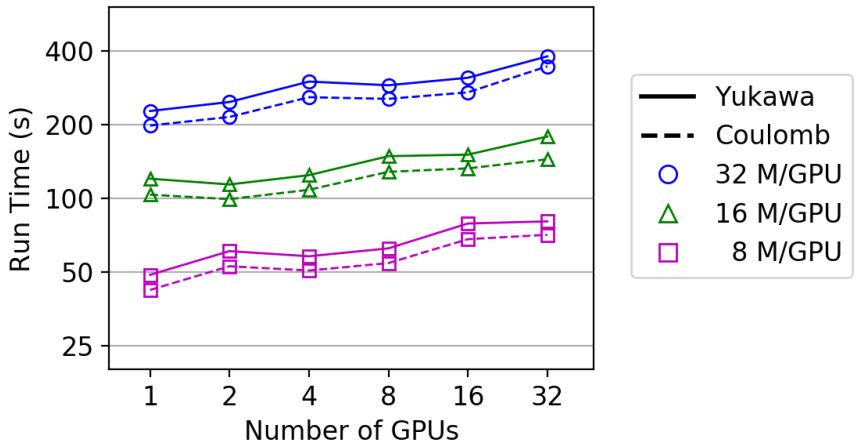


Figure 2.9: Weak scaling for the GPU accelerated treecode. Computation times for the Coulomb and Yukawa potentials with particles per GPU set to 8, 16, and 32 million, increasing the number of GPUs from 1 to 32.

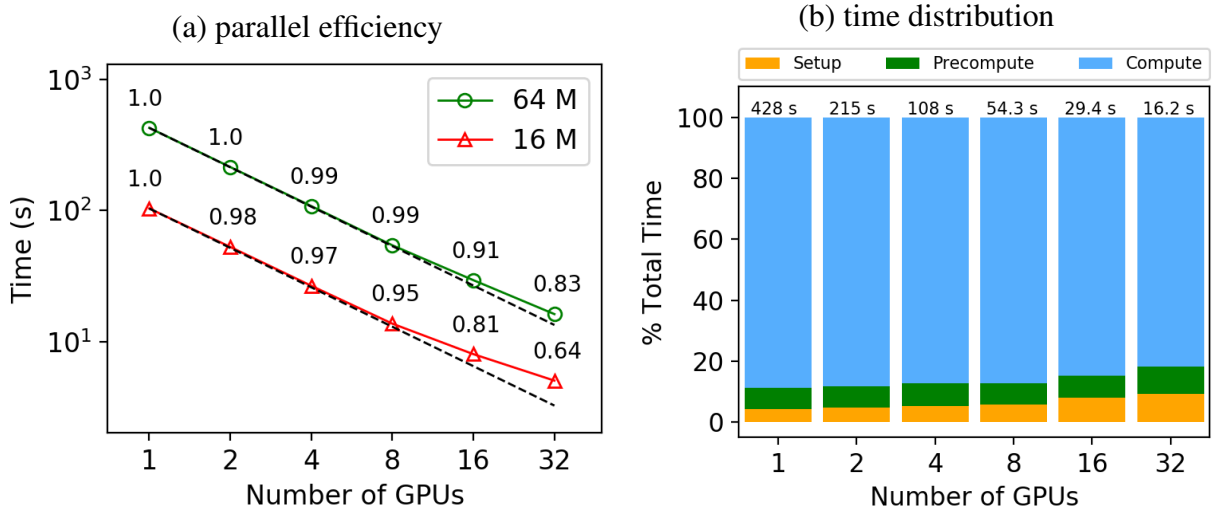


Figure 2.10: Coulomb kernel, strong scaling of the BLTC up to 8 nodes (32 GPUs) demonstrated on 16M and 64M particles using $n = 8$ and $\theta = 0.8$, giving relative 2-norm errors 4.0e-6 and 5.9e-6. (a) 16M and 64M, computation times labeled with their efficiency relative to a single GPU. (b) 64M, the percent of time spent in the setup, precompute, and compute phases as the number of GPUs increases, with the total time labeled above each bar.

## 2.5 Conclusion

We have presented an MPI + OpenACC implementation of two treecodes based on barycentric interpolation, the barycentric Lagrange treecode (BLTC) and the barycentric Hermite treecode (BHTC), both part of the BaryTree library. The distributed memory parallelization achieves efficiency by only communicating data required to fill Locally Essential Trees, implemented with

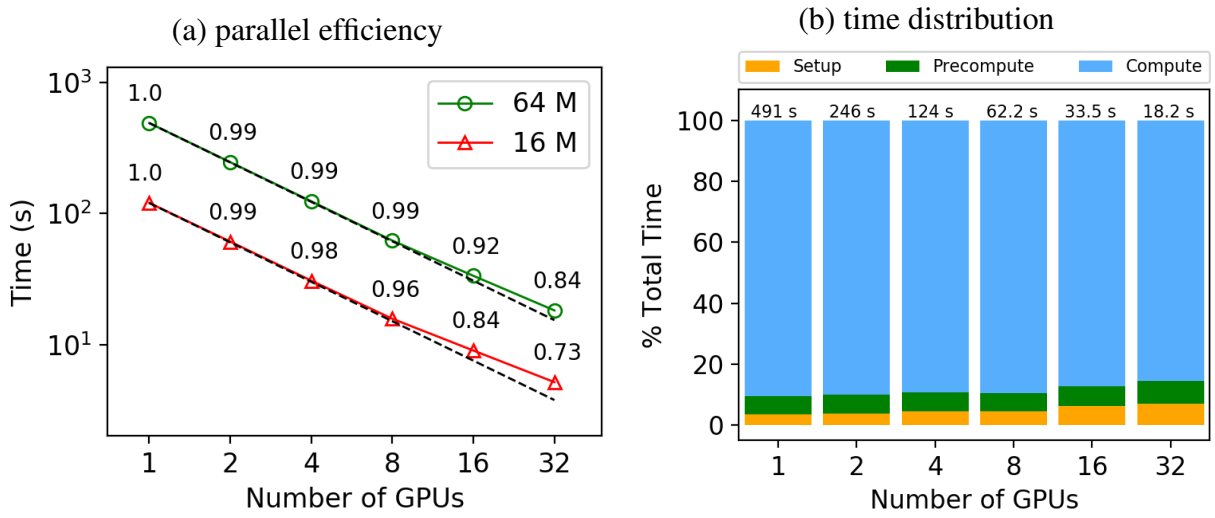


Figure 2.11: Yukawa kernel, strong scaling of the BLTC up to 8 nodes (32 GPUs) demonstrated on 16M and 64M particles using $n = 8$ and $\theta = 0.8$. giving a relative 2-norm error 5.9e-6. (a) 16M and 64M, computation times labeled with their efficiency relative to a single GPU. (b) 64M, the percent of time spent in the setup, precompute, and compute phases as the number of GPUs increases, with the total time labeled above each bar.

Remote Memory Accesses (RMA). The GPU acceleration achieves good efficiency due to the batch processing of target particles and the unique structure of the barycentric particle-cluster approximations, specifically the independence of interactions between a target particle and each interpolation point. First we demonstrated significant speedups of the GPU-accelerated BLTC and BHTC over the CPU counterparts for randomly distributed particles interacting via the Coulomb kernel. Next we compared the BLTC and BHTC, and observed the main advantages of each to be that unlike the BHTC, the BLTC is kernel-independent, however the BHTC outperforms the BLTC in the high accuracy regime. We then demonstrated good parallel scaling up to eight nodes containing four P100s each for a total of 32 GPUs. For the test system consisting of 64 million particles, the treecode computes the potential in 16.2 seconds, with a relative error of 5.9e-6, while maintaining a parallel efficiency of 83%.

This treecode is the workhorse behind the Green's function based electronic structure calculations described in Chapters 3 and 4, in which a vast majority of the overall work resides in evaluating discrete convolution sums.

# Chapter 3

# Treecode-Accelerated Green Iteration for All-Electron Kohn-Sham Density Functional Theory

This chapter presents the development of a Green's function based method for Kohn-Sham Density Functional Theory calculations. Specifically, this chapter focuses on all-electron calculations, in which all electrons are explicitly accounted for and the nuclear potentials are the singular Coulomb potentials, while Chapter 4 extends this work to pseudopotential calculations in which "core" electrons are absorbed into the nucleus, resulting in a smooth and non-singular nuclear pseudopotential. The code developed here, called Treecode-Accelerated Green Iteration (TAGI), relies heavily on the GPU-accelerated treecode called BaryTree developed in Chapter 2. TAGI and BaryTree were developed concurrently; therefore not all features of BaryTree were available during the development of TAGI. In particular, this chapter restricts calculations to a single compute node using the shared-memory parallelization presented in Section 2.3.1, rather than the fully distributed memory MPI capability described in Section 2.3.2. The extension of TAGI to multiple compute nodes is left for Chapter 4.

## 3.1 Introduction

Electronic structure calculations complement materials engineering experiments by predicting properties such as binding energy, inter-atomic forces, magnetization, and doping effects. Density Functional Theory (DFT) [9], which describes a system and its properties by its electron density, has been the workhorse of ground state electronic structure computations. For an $N_e$-electron system, the Kohn-Sham approach to DFT [10] reduces the $3N_e$-dimensional problem for the many-body wavefunction to a 3-dimensional problem for the electron density. In particular, the system of $N_e$ interacting electrons is replaced by a fictitious system of $N_e$ non-interacting electrons giving rise to the same electron density. In principle, the Kohn-Sham formulation is exact for the ground state properties of materials systems, but it requires knowledge of the exchange-correlation functional, which is not known explicitly and is modeled in practice. Approximating the exchange-correlation functional is an active area of research [11, 12, 13], and better approximations enable Kohn-Sham

DFT to more accurately predict ground state materials properties.

**Previous related work.** There are many options for performing either all-electron or pseudopotential DFT calculations, where, in the latter case, only the valence electrons are computed. Often a basis set is used to represent the wavefunctions and electron density [73]. For periodic systems, the plane-wave basis is widely used for pseudopotential calculations [28, 74, 75, 76, 7], and for all-electron calculations that require higher resolution to capture the rapidly oscillating wavefunctions, the augmented plane wave basis [77] and its variants are employed [78, 79, 80, 81, 82]. For non-periodic systems, Gaussian basis sets are widely used in quantum chemistry codes [83, 84] as they afford analytic evaluation of many integral and differential operators. A more recent option is the finite-element basis [34, 35, 36, 37], which efficiently treats periodic or non-periodic boundary conditions, and pseudopotential or all-electron systems using higher order finite-elements [38, 39, 40, 5].

The previously described methods are based on solving the Kohn-Sham eigenvalue equation, a single-particle Schrödinger-like differential equation. In this work we consider an alternative approach in which the eigenvalue problem in differential form is converted into a fixed-point problem in integral form by convolution with the modified Helmholtz Green's function. While integral equation methods are extensively used for the wave equations arising in classical scattering [45, 85, 86, 87, 88, 89, 90] and quantum scattering [91, 92, 93, 94, 95, 96], these methods have received much less attention for eigenvalue problems corresponding to ground state calculations of the Schrödinger or Kohn-Sham equations. The integral equation approach was first applied by Kalos [97] to solve the Schrödinger equation for 3- and 4-electron systems using Monte Carlo minimization. Later, Zhao et al. [98] used this approach to investigate various 1-electron systems in 3D, where the convolution integrals were computed using the Multi-Level Fast Multipole Method.

In recent work, the integral equation approach was extended to the Hartree-Fock and Kohn-Sham equations, where the electron density was updated in self-consistent field (SCF) iterations, and the fixed-point problem for the wavefunctions and eigenvalues in each SCF was solved by a process called Green Iteration. Harrison et al. [99] implemented Green Iteration for the Kohn-Sham equations in a multiwavelet basis that provides local refinement for each wavefunction, and this is now incorporated in the MADNESS code [100]. The convergence of Green Iteration for the many-body Schrödinger equation was investigated by Mohlenkamp and Young [101, 102], who proved that the iteration converges for $N_e = 1$ and $N_e = 2$, provided the interaction potential belongs to the function space $L^2(\mathbb{R}^3) + L^\infty(\mathbb{R}^3)$ and the $L^\infty(\mathbb{R}^3)$ piece can be taken to be arbitrarily small. Khoromskij [103] later extended this proof to Kohn-Sham DFT, where now the electron-electron interaction potential is replaced by the exchange-correlation potential which must satisfy the same function space requirements. Subsequently, Rakhuba and Oseledets [104, 105] applied Green Iteration to the Hartree-Fock and Kohn-Sham equations in a Tucker tensor basis that uses low rank approximations of the wavefunctions.

**Present work.** We present a new integral equation based method called Treecode-Accelerated Green Iteration (TAGI) for all-electron Kohn-Sham DFT calculations. The key features of TAGI that enable accurate and efficient calculations are (1) adaptive mesh refinement, (2) high order quadrature, (3) singularity subtraction for convolution integrals, (4) gradient-free eigenvalue update, (5) Anderson mixing for SCF and Green Iteration, and (6) treecode computation of discrete convolution sums.

TAGI is a real-space method in which the fields are represented directly at quadrature points. TAGI uses adaptive mesh refinement to efficiently represent the fields, which vary rapidly near the nuclei but decay smoothly in the far-field. The adaptive refinement scheme results in a set of cuboid cells, which are discretized with Chebyshev points of the first kind, and all integrals are evaluated with the Fejér ("classical" Clenshaw-Curtis) quadrature rule [106, 107]. The convolution integrals have singular kernels (Coulomb and Yukawa), which impede the accuracy of the quadrature rule, and we employ singularity subtraction to reduce the error in the quadrature sums. A standard singularity subtraction scheme is used for the Yukawa kernel [108, 109] and we developed a modified version for the Coulomb kernel. To further improve accuracy, we use a gradient-free eigenvalue update [99] within Green Iteration to eliminate the error arising from numerical differentiation in the standard gradient eigenvalue update. We analyze the convergence rate of Green Iteration and use a fixed-point acceleration technique to alleviate slow convergence. Finally, the discrete convolution sums are efficiently evaluated using a Barycentric Lagrange Treecode [50] (BLTC), which reduces the computational complexity from $O(N^2)$ to $O(N \log N)$ while introducing a small and controllable approximation error. Furthermore, the BLTC is accelerated on GPUs with OpenACC [52] and across multiple GPUs on a single node with OpenMP. We demonstrate the impact on accuracy and efficiency of each of the previously described features on the carbon monoxide molecule, and then perform ground state energy calculations for several atoms and molecules, demonstrating TAGI's ability to achieve chemical accuracy of 1 mHa/atom.

The remainder of this chapter is organized as follows. Section 3.2 presents Kohn-Sham DFT, and the standard Self-Consistent Field iteration for computing the ground state density and wave-functions. Section 3.3 presents the integral equation formulation we employ and Green Iteration for the resulting fixed-point problem. Section 3.4 describes the numerical techniques developed in this work to enhance the accuracy of the integral formulation, and demonstrates these ideas on the carbon monoxide molecule. Section 3.5 investigates the convergence rate of Green Iteration and demonstrates the fixed-point acceleration technique used in TAGI. Section 3.6 describes the treecode algorithm for computing fast approximations of the convolution integrals and demonstrates the efficiency of the GPU-accelerated implementation used in this work. Section 3.7 applies TAGI to several atoms and small molecules, achieving chemical accuracy of 1 mHa/atom with respect to reference values. Section 3.8 provides a summary of our findings, and discusses a path forward for

this approach to further improve performance and scale to larger systems.

### 3.2 Kohn-Sham Density Functional Theory

The input to Kohn-Sham DFT consists of the positions and atomic numbers of the atoms in the system, and the output consists of the ground-state electron density along with the Kohn-Sham single-electron wavefunctions, from which the desired observables (including ground-state energy and ionic forces) can be computed. The Kohn-Sham equations are

$$\mathcal{H}[\rho]\psi_i(\mathbf{r}) = \varepsilon_i\psi_i(\mathbf{r}), \quad i = 1, 2, \ldots, \quad \mathcal{H}[\rho] = -\frac{1}{2}\nabla^2 + V_{eff}[\rho]\,, \tag{3.1}$$

where $\mathcal{H}[\rho]$ is the Kohn-Sham Hamiltonian, $\rho = \rho(\mathbf{r})$ is the electron density, $\varepsilon_i$ are the Kohn-Sham eigenvalues, and $\psi_i(\mathbf{r})$ are the Kohn-Sham eigenfunctions, also referred to as the Kohn-Sham wavefunctions. Here, we restrict ourselves to a spin-independent formulation on non-periodic systems, but the general ideas presented in this work can be extended to a spin-dependent formulation and periodic geometries in a straightforward manner. The effective Kohn-Sham potential has the form,

$$V_{eff}[\rho](\mathbf{r}) = V_H[\rho](\mathbf{r}) + V_{ext}(\mathbf{r}) + V_{xc}[\rho](\mathbf{r}), \tag{3.2}$$

where the first two terms are the Hartree potential due to the electron density and the external potential due to the $N_A$ atomic nuclei located at $\mathbf{R}_j$ with charges $Z_j$, respectively,

$$V_H[\rho](\mathbf{r}) = \int \frac{\rho(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|}d\mathbf{r}', \quad V_{ext}(\mathbf{r}) = \sum_{j=1}^{N_A} \frac{-Z_j}{|\mathbf{r} - \mathbf{R}_j|}, \tag{3.3}$$

and the third term is the exchange-correlation potential $V_{xc}[\rho] = \partial E_{xc}[\rho]/\partial\rho$ depending on the exchange-correlation energy $E_{xc}[\rho]$. The electron density depends on the eigenvalues and wavefunctions,

$$\rho(\mathbf{r}) = 2\sum_{i=1}^{N_w} f(\varepsilon_i, \mu_F)|\psi_i(\mathbf{r})|^2, \quad f(\varepsilon, \mu_F) = \frac{1}{e^{(\varepsilon-\mu_F)/k_BT} + 1}, \tag{3.4}$$

where $f(\varepsilon, \mu_F)$ is the fractional occupation computed by Fermi-Dirac statistics [28, 25], with Fermi energy $\mu_F$, Boltzmann constant $k_B$, and temperature $T$. The Fermi energy $\mu_F$ is determined from the constraint on the total number of electrons $N_e$,

$$2\sum_{i=1}^{N_w} f(\varepsilon_i, \mu_F) = N_e\,. \tag{3.5}$$

The sums in Eq. (3.4) and Eq. (3.5) run over the $N_w$ lowest energy wavefunctions, where $N_w$ is chosen so that the fractional occupation of any higher energy wavefunction is negligible.

The preceding equations constitute a non-linear eigenvalue problem and the standard solution method uses the Self-Consistent Field iteration (SCF) outlined in Algorithm 3.1. The iteration takes the atomic positions and an initial guess for the electron density as input. The output is the converged electron density and wavefunctions, from which observables are computed. The iteration starts in line 1. In line 2, at the $n$th step of the iteration, the effective potential $V_{eff}[\rho_{in}^{(n)}]$ is constructed from the input electron density of the current iterate by Eq. (3.2). In line 3, the eigenvalue problem in Eq. (3.1) is solved for the eigenpairs $(\varepsilon_i, \psi_i)$. In line 4, the Fermi energy $\mu_F$ and fractional occupations $f(\varepsilon_i, \mu_F)$ are computed. In line 5, these quantities are used to compute a new output density $\rho_{out}^{(n)}$ by Eq. (3.4). In line 6, the scheme checks whether the density has converged to a desired tolerance; if so, then the iteration stops and returns the latest density; otherwise a new input density $\rho_{in}^{(n+1)}$ is constructed by the Anderson mixing scheme described below and the iteration continues. The present work follows this approach, but focuses on the solution of the eigenvalue problem (line 3), which is the most computationally intensive step in the SCF iteration, using treecode-accelerated Green Iteration (TAGI) described below.

**Algorithm 3.1** Self-Consistent Field Iteration (SCF)

**input:** atomic positions and initial guess for electron density $\rho_{in}^{(0)}$
**output:** electron density $\rho_{out}^{(n)}$ and Kohn Sham wavefunctions $\psi_i^{(n)}, i = 1, \ldots, N_w$

1: for $n = 0, 1, 2, \ldots$
2: given $\rho_{in}^{(n)}$, construct effective potential $V_{eff}[\rho_{in}^{(n)}]$ by Eq. (3.2)
3: using $V_{eff}[\rho_{in}^{(n)}]$, solve eigenvalue problem $\mathcal{H}[\rho_{in}^{(n)}]\psi_i^{(n)} = \varepsilon_i^{(n)}\psi_i^{(n)}, i = 1, \ldots, N_w$
4: using $\varepsilon_i^{(n)}$, compute Fermi energy $\mu_F$ and fractional occupations $f(\varepsilon_i^{(n)}, \mu_F)$ by Eq. (3.5)
5: using $f(\varepsilon_i^{(n)}, \mu_F), \psi_i^{(n)}$, construct new density $\rho_{out}^{(n)}$ by Eq. (3.4)
6: if $||\rho_{out}^{(n)} - \rho_{in}^{(n)}||_2 < tol_{scf}$ , return $\rho_{out}^{(n)}$
7: else construct new density $\rho_{in}^{(n+1)}$ by Anderson mixing and return to step 2

Following the $n$th step in the SCF iteration, the output electron density $\rho_{out}^{(n)}$ has been computed and a new input electron density $\rho_{in}^{(n+1)}$ must be initialized for the next step in the SCF iteration. The simplest choice is $\rho_{in}^{(n+1)} \to \rho_{out}^{(n)}$, but this fails to converge except for the simplest systems, and it is common practice in DFT simulations to instead use a density mixing scheme. Density mixing schemes construct $\rho_{in}^{(n+1)}$ from the history of previous input and output densities in an attempt to minimize the next residual, $||\rho_{in}^{(n+1)} - \rho_{out}^{(n+1)}||$. Among various options, in this work we use Anderson mixing [28, 110, 111]; a brief description follows.

The input density for the $(n+1)$st SCF iteration is given by

$$\rho_{in}^{(n+1)} = \beta\bar{\rho}_{out} + (1-\beta)\bar{\rho}_{in}, \tag{3.6}$$

where $\beta \in [0, 1]$ is a mixing parameter, and

$$\bar{\rho}_{in} = \sum_{k=0}^{n} c_k \rho_{in}^{(n-k)}, \quad \bar{\rho}_{out} = \sum_{k=0}^{n} c_k \rho_{out}^{(n-k)}, \tag{3.7}$$

are weighted averages of previous input and output densities. The weights $c_k$ are computed by solving the optimization problem to minimize $||\bar{\rho}_{out} - \bar{\rho}_{in}||_2$. Applying the constraint $\sum_{k=0}^{n} c_k = 1$, Eq. (3.7) is rewritten as

$$\bar{\rho}_{in} = \rho_{in}^{(n)} + \sum_{k=1}^{n} c_k(\rho_{in}^{(n-k)} - \rho_{in}^{(n)}), \quad \bar{\rho}_{out} = \rho_{out}^{(n)} + \sum_{k=1}^{n} c_k(\rho_{out}^{(n-k)} - \rho_{out}^{(n)}). \tag{3.8}$$

This converts the constrained optimization for $n$ parameters into an unconstrained optimization problem for $n-1$ parameters, improving the conditioning of the linear system which is to be solved for the $c_k$'s [112]. Next, consider the quantity $\bar{\rho}_{out} - \bar{\rho}_{in}$. Rearranging Eq. (3.8) yields

$$\bar{\rho}_{out} - \bar{\rho}_{in} = (\rho_{out}^{(n)} - \rho_{in}^{(n)}) + \sum_{k=1}^{n} c_k \left( (\rho_{out}^{(n-k)} - \rho_{in}^{(n-k)}) - (\rho_{out}^{(n)} - \rho_{in}^{(n)}) \right), \tag{3.9}$$

Defining $F = \rho_{out} - \rho_{in}$, this is rewritten as

$$\bar{F} = F^{(n)} + \sum_{k=1}^{n} c_k \left( F^{(n-k)} - F^{(n)} \right). \tag{3.10}$$

Minimizing $||\bar{F}||_2 = ||\bar{\rho}_{out} - \bar{\rho}_{in}||_2$ amounts to solving an overdetermined system of equations for the weights $c_k$ and leads to the normal equations,

$$\sum_{k=1}^{n} \left( F^{(n)} - F^{(n-m)}, F^{(n)} - F^{(n-k)} \right) c_k = \left( F^{(n)} - F^{(n-m)}, F^{(n)} \right), \quad m = 1, \ldots, n, \tag{3.11}$$

where $(f, g) = \int f(\mathbf{r}) g(\mathbf{r}) d\mathbf{r}$. In TAGI, these inner products are numerically computed with the discretization scheme described below. After obtaining the weights $c_k$, the density is constructed from Eq. (3.6). In practice a partial history can be used, $k = 0 : k_{max}$, instead of the complete history as presented above, $k = 0 : n$. Throughout this work we use default values of $k_{max} = 10$ and $\beta = 0.5$ for the history cutoff and mixing parameter.

Having obtained the converged $\varepsilon_i, \psi_i(\mathbf{r}), \rho(\mathbf{r})$ from the converged SCF iteration, the ground-state energy of the system is

$$E = E_{kin} + E_{xc} + E_H + E_{ext} + E_{ZZ}. \tag{3.12}$$

In this expression, the first two terms are the kinetic energy and exchange-correlation energy,

respectively,

$$E_{kin} = \sum_{i=1}^{N_w} f(\varepsilon_i, \mu_F) \int \psi_i(\mathbf{r}) \left( -\frac{1}{2}\nabla^2 \right) \psi_i(\mathbf{r}) d\mathbf{r}, \quad E_{xc}[\rho] = \int \varepsilon_{xc}[\rho](\mathbf{r})\rho(\mathbf{r}) d\mathbf{r}, \tag{3.13}$$

where $\varepsilon_{xc}[\rho](\mathbf{r})$ is the exchange-correlation energy per electron for the chosen DFT functional, and the remaining three terms are the Hartree energy, external electrostatic energy, and nuclear repulsion energy, respectively,

$$E_H[\rho] = \frac{1}{2} \int V_H[\rho](\mathbf{r})\rho(\mathbf{r}) d\mathbf{r}, \quad E_{ext}[\rho] = \int V_{ext}(\mathbf{r})\rho(\mathbf{r}) d\mathbf{r}, \quad E_{ZZ} = \frac{1}{2} \sum_{i,j \neq i} \frac{Z_i Z_j}{|\mathbf{R}_i - \mathbf{R}_j|}. \tag{3.14}$$

This work employs the Local Density Approximation (LDA) [113, 114] for $V_{xc}[\rho], \varepsilon_{xc}[\rho]$, given by

$$\varepsilon_{xc}[\rho](\mathbf{r}) = \varepsilon_x[\rho](\mathbf{r}) + \varepsilon_c[\rho](\mathbf{r}), \tag{3.15}$$

where

$$\varepsilon_x[\rho](\mathbf{r}) = -\frac{3}{4} \left( \frac{3}{\pi} \right)^{1/3} \rho^{1/3}(\mathbf{r}), \tag{3.16}$$

and

$$\varepsilon_c[\rho](\mathbf{r}) = \begin{cases} \frac{\gamma}{(1+\beta_1\sqrt{r_s}+\beta_2 r_s)} & r_s \geq 1, \\ A \log r_s + B + C r_s \log r_s + D r_s & r_s < 1, \end{cases} \tag{3.17}$$

where $r_s = \left( \frac{3}{4\pi} \rho(\mathbf{r}) \right)^{1/3}$ and the parameters $\gamma$, $\beta_1$, $\beta_2$, $A$, $B$, $C$, and $D$ are fit from Monte Carlo calculations [113]. In practice these are computed using the *Libxc* package [115, 116].

### 3.3 Solution of Eigenvalue Problem by Green Iteration

Several methods are available for solving the eigenvalue problem in each SCF iteration (step 4 in Algorithm 3.1). Among real-space methods, finite-difference [30, 32, 33] and finite-element [37, 40] methods represent the differential operator as a sparse matrix and use iterative techniques to compute the eigenpairs $(\varepsilon_i, \psi_i)$. By contrast, in this work the differential equation is converted into an integral equation by convolution with the modified Helmholtz Green's function [97], and then an iterative technique called Green Iteration is applied to obtain the eigenpairs [99, 101, 103, 104]. We describe these steps below.

Following Kalos [97], the Kohn-Sham equations in Eq. (3.1) are rewritten in the form

$$\left( \frac{1}{2}\nabla^2 + \varepsilon_i \right) \psi_i = V_{eff}[\rho]\psi_i, \tag{3.18}$$

where $\rho$ is the electron density for a given SCF iteration. Since the bound state eigenvalues of the Kohn-Sham Hamiltonian are negative, $\varepsilon_i < 0$, Eq. (3.18) is a modified Helmholtz equation with Green's function,

$$G_{\varepsilon_i}(\mathbf{r}, \mathbf{r}') = -\frac{e^{-\sqrt{-2\varepsilon_i}|\mathbf{r}-\mathbf{r}'|}}{2\pi|\mathbf{r}-\mathbf{r}'|}, \tag{3.19}$$

where free-space boundary conditions are assumed. Note that the factor of 2 difference between the definitions of the modified Helmholtz Green's function in Eq. (3.19) and Eq. (1.15) is due to the $\frac{1}{2}$ coefficient of the Laplacian in Eq. (3.18). Then, convolution with Eq. (3.18) yields the integral form of the Kohn-Sham eigenvalue problem,

$$\psi_i(\mathbf{r}) = \mathcal{G}(\varepsilon_i)\psi_i(\mathbf{r}), \quad i = 1, \ldots, N_w, \quad \psi_i \perp \psi_j,\ j < i, \tag{3.20}$$

where

$$\mathcal{G}(\varepsilon)\psi(\mathbf{r}) = \int G_\varepsilon(\mathbf{r}, \mathbf{r}')V_{eff}[\rho](\mathbf{r}')\psi(\mathbf{r}')d\mathbf{r}', \tag{3.21}$$

defines a 1-parameter family of linear integral operators.

Note that Eq. (3.20) can be viewed as a fixed-point problem and this motivates the solution method called Green Iteration described in Algorithm 3.2. The scheme takes as input the effective potential $V_{eff}[\rho]$ for the current SCF and an initial guess for the eigenpairs $(\varepsilon_i^{(0)}, \psi_i^{(0)})$, and provides the converged eigenpairs $(\varepsilon_i, \psi_i)$ as output. Line 1 is the outer loop over wavefunctions and line 2 is the iteration for a given wavefunction. Line 3 applies the integral operator $\mathcal{G}(\varepsilon_i^{(n)})$ to the current wavefunction $\psi_i^{(n)}$. Line 4 updates the eigenvalue; several methods are available and we compare some of them below. Line 5 is the deflation step that orthogonalizes the new wavefunction $\psi_i^{(n+1)}$ against the previously converged wavefunctions, and line 6 normalizes it. Line 7 checks for convergence; if the tolerance is satisfied, then the eigenpair is stored and the process returns to line 1; otherwise the iteration in line 2 continues.

**Algorithm 3.2** Green Iteration

**input**: effective potential $V_{eff}[\rho]$ for current SCF
**input**: initial guess for eigenpairs $(\varepsilon_i^{(0)}, \psi_i^{(0)}), i = 1, \ldots, N_w$
**output**: eigenpairs $(\varepsilon_i, \psi_i), i = 1, \ldots, N_w$

1: for $i = 1, 2, \ldots, N_w$
2: for $n = 0, 1, 2, \ldots$
3: compute $\psi_i^{(n+1)} = \mathcal{G}(\varepsilon_i^{(n)})\psi_i^{(n)}$
4: update eigenvalue $\varepsilon_i^{(n+1)}$
5: orthogonalize $\psi_i^{(n+1)}$ against previously converged wavefunctions $\psi_j, j < i$
6: normalize $\psi_i^{(n+1)}$
7: if $||\psi_i^{(n+1)} - \psi_i^{(n)}||_2 < tol_{gi}$ set $(\varepsilon_i, \psi_i) = (\varepsilon_i^{(n)}, \psi_i^{(n)})$ and return to line 1
8: else return to line 2 and continue iteration

## 3.4 Spatial Discretization Techniques

This section focuses on the spatial discretization techniques used in TAGI. These include the initialization scheme for the electron density and wavefunctions, the quadrature and adaptive mesh refinement techniques, the singularity subtraction schemes used to evaluate the convolution integrals, and the gradient-free approach used to update the eigenvalues. The section concludes by demonstrating the effect of these techniques using the carbon monoxide molecule as an example. Note that Hartree atomic units are used throughout this work.

### 3.4.1 Initial Electron Density and Eigenpairs

The SCF iteration uses an initial guess for the electron density of the form,

$$\rho^{(0)}(\mathbf{r}) = \sum_{j=1}^{N_a} \rho_j(|\mathbf{r} - \mathbf{R}_j|), \tag{3.22}$$

where $\rho_j(|\mathbf{r} - \mathbf{R}_j|)$ is a radial 1-atom electron density associated with the $j$th atom. These 1-atom densities are precomputed by solving a radial version of the Kohn-Sham problem for each atomic species. In addition, Green Iteration requires an initial guess for the eigenpairs, $(\varepsilon_i^{(0)}, \psi_i^{(0)}(\mathbf{r})), i = 1, \ldots, N_w$. The number of wavefunctions $N_w$ is determined as follows. Since each wavefunction is occupied up to two electrons, there is a lower bound, $N_w \geq N_e/2$, however there is no sharp upper bound. In practice $N_w$ should be chosen large enough to accommodate all states with significant fractional occupation $f(\varepsilon_i, \mu_F)$. To this end, $N_w$ is initialized to be larger than $N_e/2$, and upon obtaining the eigenpairs, if the fractional occupation of the highest state is negligibly small, then $N_w$ is considered large enough; otherwise, $N_w$ is increased and the process is repeated until the check is satisfied. The initial guess for the eigenpairs depends on whether or not this is the first step in the SCF iteration. In the first step, the wavefunctions are initialized using 1-atom wavefunctions obtained in the radial solve for the initial electron density, $\psi_{n\ell}(r)$, multiplied by appropriate spherical harmonics, and the initial eigenvalues are computed by the Rayleigh quotient with the Kohn-Sham Hamiltonian $\mathcal{H}$, $\varepsilon_i^{(0)} = \langle \psi_i^{(0)}, \mathcal{H}\psi_i^{(0)} \rangle$. In subsequent steps of the SCF iteration, the eigenpairs of the previous step are taken as the initial guess.

### 3.4.2 Spatial Discretization and Quadrature Schemes

The energy integrals and convolution integrals will be evaluated on a set of cuboid cells representing a bounded computational domain. Using the Hartree energy in Eq. (3.14) as an

example,

$$E_H = \frac{1}{2}\int V_H(\mathbf{r})\rho(\mathbf{r})d\mathbf{r} \approx \frac{1}{2}\sum_{i=1}^{N_c}\int_{C_i} V_H(\mathbf{r})\rho(\mathbf{r})d\mathbf{r} \approx \frac{1}{2}\sum_{i=1}^{N_c}\sum_{j=1}^{(p+1)^3} V_H(\mathbf{r}_{ij})\rho(\mathbf{r}_{ij})w_{ij}, \tag{3.23}$$

where $N_c$ is the number of cells, $(p+1)^3$ is the number of quadrature points in each cell, indices $i, j$ refer to quadrature point $j$ in cell $i$, and $w_{ij}$ are the quadrature weights. The total number of mesh points is denoted by $N_m = (p+1)^3 N_c$. The quadrature scheme uses Chebyshev points of the first kind; on the interval $[-1, 1]$ these are given by

$$x_j = \cos\theta_j, \quad \theta_j = \frac{(j+1/2)\pi}{p+1}, \quad j = 0 : p. \tag{3.24}$$

Within each cell, the integrals are evaluated using the Fejér (or "classical" Clenshaw-Curtis) quadrature rule [106, 107] with quadrature weights $w_{ij}$. The $p+1$ point Fejér quadrature rule integrates $p$th-degree polynomials exactly, so we refer to this as a $p$th-order quadrature rule. For completeness, we provide the method for computing the Fejér quadrature weights in 1-dimension using the method of Discrete Cosine Transforms, or, equivalently, expansions in Chebyshev polynomials. Consider the 1-dimensional integral of a smooth, but non-periodic function $f(x)$,

$$\int_{-1}^{1} f(x)dx. \tag{3.25}$$

**Step 1**. Change of variables, $x \to \cos\theta$, $dx \to -\sin\theta d\theta$, yields

$$\int_{-1}^{1} f(x)dx = \int_{0}^{\pi} f(\cos\theta)\sin\theta d\theta. \tag{3.26}$$

**Step 2**. The Fourier cosine transform of $f(\cos\theta)$ is given by

$$f(\cos\theta) = \frac{a_0}{2} + \sum_{i=1}^{\infty} a_i \cos(i\theta), \tag{3.27}$$

where the Fourier cosine coefficients $a_i$ are given by

$$a_i = \frac{2}{\pi}\int_{0}^{\pi} f(\cos\theta)\cos(i\theta)d\theta. \tag{3.28}$$

Substitution into Eq. (3.25) and simplifying gives

$$\int_{-1}^{1} f(x)dx = \int_0^{\pi} f(\cos\theta)\sin\theta d\theta = \int_0^{\pi} \left[\frac{a_0}{2} + \sum_{i=1}^{\infty} a_i \cos(i\theta)\right] \sin\theta d\theta \tag{3.29a}$$

$$= \frac{a_0}{2}\int_0^{\pi} \sin\theta d\theta + \sum_{i=1}^{\infty} a_i \int_0^{\pi} \cos(i\theta)\sin\theta d\theta = a_0 + \sum_{i=1}^{\infty} a_{2i}\frac{2}{1-(2i)^2}. \tag{3.29b}$$

Hence, the integral in Eq. (3.25) is given exactly in terms of the Fourier cosine coefficients $a_i$. These coefficients must be computed numerically using Eq. (3.28), but, importantly, these integrands are periodic over their domain, admitting accurate results using a uniformly spaced and uniform weight quadrature scheme (trapezoid or midpoint). Clenshaw-Curtis uses the trapezoid rule and Fejér uses the midpoint.

**Step 3**. Compute the Discrete Cosine Transform (DCT) coefficients $\widehat{a}_i$. Uniform discretization of $\theta$ for the midpoint method gives

$$\theta_j = \frac{(j+1/2)\pi}{p+1}, \quad j = 0, ..., p, \quad \Delta\theta = \frac{\pi}{p+1}, \tag{3.30}$$

and the discretized integral for $a_i$ becomes

$$\widehat{a}_i \approx \frac{2}{\pi}\sum_{j=0}^{p} f(\cos(\theta_j))\cos(i\theta_j)\Delta\theta. \tag{3.31}$$

Recall from the change of variables,

$$x_j = \cos(\theta_j) = \cos\left(\frac{(j+1/2)\pi}{p+1}\right), \tag{3.32}$$

i.e. the midpoint discretization of $\theta$ gives Chebyshev points of the first kind in $x$. Denoting $f(\cos(\theta_j)) = f(x_j) = f_j$, substitution into Eq. (3.31) gives

$$\widehat{a}_i = \frac{2}{\pi}\sum_{j=0}^{p} f_j \cos(i\theta_j)\frac{\pi}{p+1} = \frac{2}{p+1}\sum_{j=0}^{p} f_j \cos(i\theta_j). \tag{3.33}$$

**Step 4**. Define $\mathcal{W}_i$ as

$$\mathcal{W}_i = \begin{cases} 1 & \text{if } i = 0, \\ \frac{2}{1-i^2} & \text{if } i \text{ is even}, \\ 0 & \text{if } i \text{ is odd}. \end{cases} \tag{3.34}$$

Substitution into the original integral (3.25) and rearrangement of the sums gives

$$\int_{-1}^{1} f(x)dx \approx \widehat{a}_0 + \sum_{i=1}^{\infty} \widehat{a}_{2i} \frac{2}{1-(2i)^2} \tag{3.35a}$$

$$= \sum_{i=0}^{\infty} \mathcal{W}_i \widehat{a}_i \tag{3.35b}$$

$$\approx \sum_{i=0}^{\infty} \mathcal{W}_i \left[ \frac{2}{p+1} \sum_{j=0}^{p} f_j \cos(i\theta_j) \right] \tag{3.35c}$$

$$= \sum_{j=0}^{p} f_j \sum_{i=0}^{\infty} \mathcal{W}_i \left[ \frac{2}{p+1} \cos(i\theta_j) \right] \tag{3.35d}$$

$$= \sum_{j=0}^{p} f_j \sum_{i=0}^{\infty} \mathcal{W}_i \left[ \frac{2}{p+1} \cos\left( \frac{i(j+1/2)\pi}{p+1} \right) \right]. \tag{3.35e}$$

**Step 5**. Truncate the sums for $\widehat{a}_i$. Carrying the sum over $i$ beyond $p$ has no benefit due to aliasing, i.e. the DCT of $f(cos\theta)$ is limited by the frequency of the discrete sampling of $\theta_j$,

$$\int_{-1}^{1} f(x)dx \approx \sum_{j=0}^{p} f_j \left( \sum_{i=0}^{p} \mathcal{W}_i \left[ \frac{2}{p+1} \cos\left( \frac{i(j+1/2)\pi}{p+1} \right) \right] \right). \tag{3.36}$$

**Step 6**. Define the $j$th quadrature weight $w_j$ as

$$w_j = \sum_{i=0}^{p} \mathcal{W}_i \left[ \frac{2}{p+1} \cos\left( \frac{i(j+1/2)\pi}{p+1} \right) \right], \tag{3.37}$$

then the Fejér quadrature rule for the integral in Eq. (3.25) is given by

$$\int_{-1}^{1} f(x)dx \approx \sum_{j=0}^{p} f_j w_j. \tag{3.38}$$

Note that $w_j$ is independent of $f(x)$ and can be precomputed once and reused for other integrands. Additionally, $w_j$ can be mapped to the interval $[a, b]$ by the linear transformation

$$w_j \to \frac{b-a}{2} w_j, \tag{3.39}$$

and the quadrature rule can be extended to 3-dimensions ($w_{ijk}$) using a tensor product of 1-dimensional weights,

$$w_{ijk} = w_i w_j w_k. \tag{3.40}$$

A tensor product grid of $(p+1)^3$ Chebyshev points is adapted to each cell; Fig. 3.1 shows a 2D schematic. Note that the Chebyshev points lie entirely inside the cell and never coincide with a vertex; as explained below this is important because the cells are chosen so that the atoms are located at cell vertices, thereby avoiding the singularity of the nuclear potential.

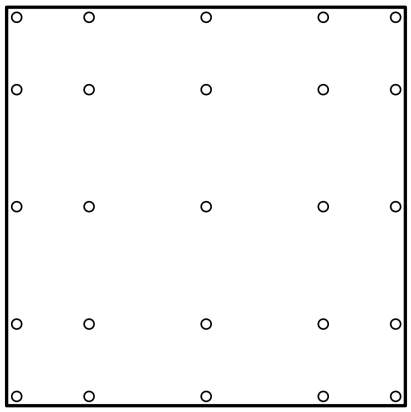

Figure 3.1: A tensor product grid of Chebyshev points of the first kind in Eq. (3.24) with $p = 4$ in a 2D cell.

The cells are defined using an adaptive refinement scheme illustrated in Fig. 3.2 for a 1-atom example. The goal of the scheme is to produce cells that resolve the regions with significant electron density and wavefunction variation, primarily near the atoms. Level 0 is a large cube surrounding the atoms in the system, with dimensions chosen to ensure that the electron density and wavefunctions are sufficiently small at the boundary. The cube is refined by bisecting it in the three coordinate directions, resulting in eight child cells. Several levels of uniform refinement are performed, and subsequent refinement is done adaptively in the following manner. Given a cell $C$, we temporarily create the child cells $C_i, i = 1:8$, and check the following criterion,

$$\left| \int_C t(\mathbf{r}) d\mathbf{r} \; - \sum_{i=1}^{8} \int_{C_i} t(\mathbf{r}) d\mathbf{r} \right| < tol_m, \tag{3.41}$$

where $t(\mathbf{r})$ is a test function specified below and $tol_m$ is a user-specified tolerance. The integrals in Eq. (3.41) are evaluated using the Fejér quadrature rule. If Eq. (3.41) is satisfied, then refinement is not needed and the child cells are discarded; otherwise the child cells are retained and the process continues. Figure 3.2 shows the schematic of a possible outcome where the initial cell is refined at level 1, but only the child cell containing the atom is refined at level 2. Once the tolerance is satisfied for every cell, a final refinement step occurs; those cells containing an atom are subdivided so that the atoms lie at cell vertices; this ensures that the Chebyshev grid points never coincide

with an atom position and hence the fields (effective potential, wavefunctions, electron density) are smooth on the interior of the cells. If the refinement scheme creates any cells with large aspect ratio, these cells are refined along their longest dimension.

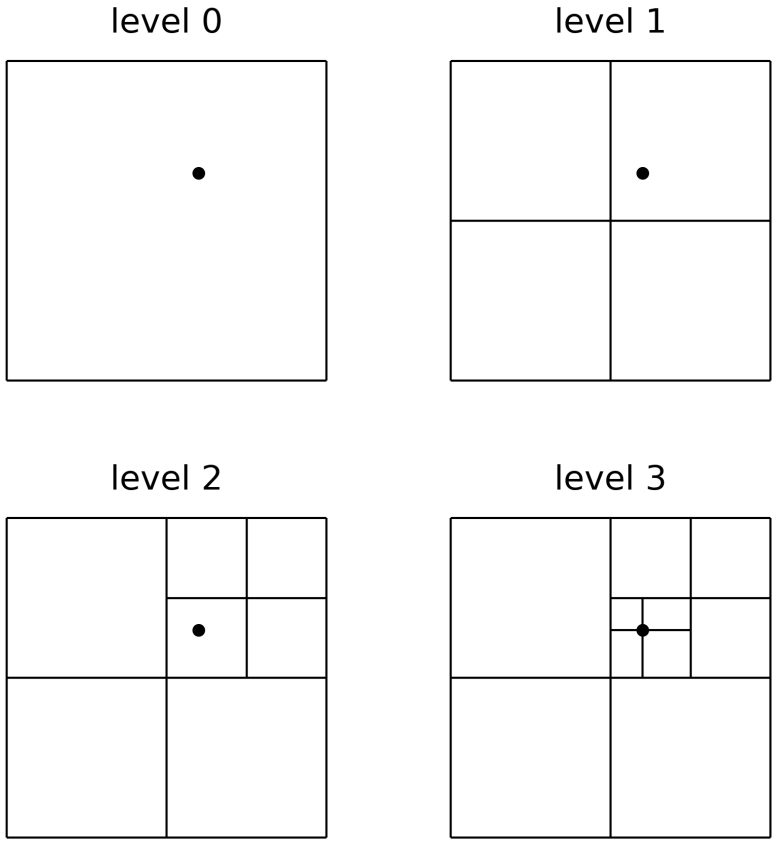


Figure 3.2: Illustration of the adaptive refinement scheme for a 1-atom system with the atom located at (•). Four levels of refinement are shown where the final refinement level puts the atom at a cell vertex.

Several options for the refinement test function were considered and we decided to use

$$t(\mathbf{r}) = V_{ext}(\mathbf{r})\sqrt{\rho^{(0)}(\mathbf{r})}, \tag{3.42}$$

where $\rho^{(0)}(\mathbf{r})$ is the initial electron density in Eq. (3.22) and $V_{ext}(\mathbf{r})$ is the external potential in Eq. (3.3). This choice is motivated by several considerations. First, it resembles the function $V_{eff}(\mathbf{r})\psi(\mathbf{r})$ appearing in the integral form of the Kohn-Sham equations (3.21); this is because near a nucleus, $\sqrt{\rho^{(0)}(\mathbf{r})}$ has the characteristics of an $s$-orbital atomic wavefunction, capturing the cusp and decay rate, and although $V_{eff}(\mathbf{r})$ is not known, $V_{ext}(\mathbf{r})$ is known and contains the Coulomb singularities that must be resolved. Second, this test function is accessible at the start of

the computation and can be evaluated at arbitrary grid points as needed in the refinement scheme.

Figure 3.3 shows an example of coarse and fine meshes for the benzene molecule ($C_6H_6$) obtained using the refinement scheme described above with 4th order quadrature. The molecule lies in the $z = 0$ plane and a truncated portion of the mesh in that plane is shown. The coarse mesh is generated with $tol_m = 1\text{e}{-4}$ and the fine mesh with $tol_m = 3\text{e}{-6}$. The resulting cell density is highest near the twelve nuclei, and the carbon atoms are more highly refined than the hydrogen atoms, as expected since the test function $V_{ext}(\mathbf{r})\sqrt{\rho^{(0)}(\mathbf{r})}$ grows faster at heavier nuclei. Compared to a variety of other refinement schemes we considered, this approach gave the best combination of accuracy and efficiency. Further below we will demonstrate convergence with respect to both the order of the quadrature rule $p$ and the mesh tolerance parameter $tol_m$.

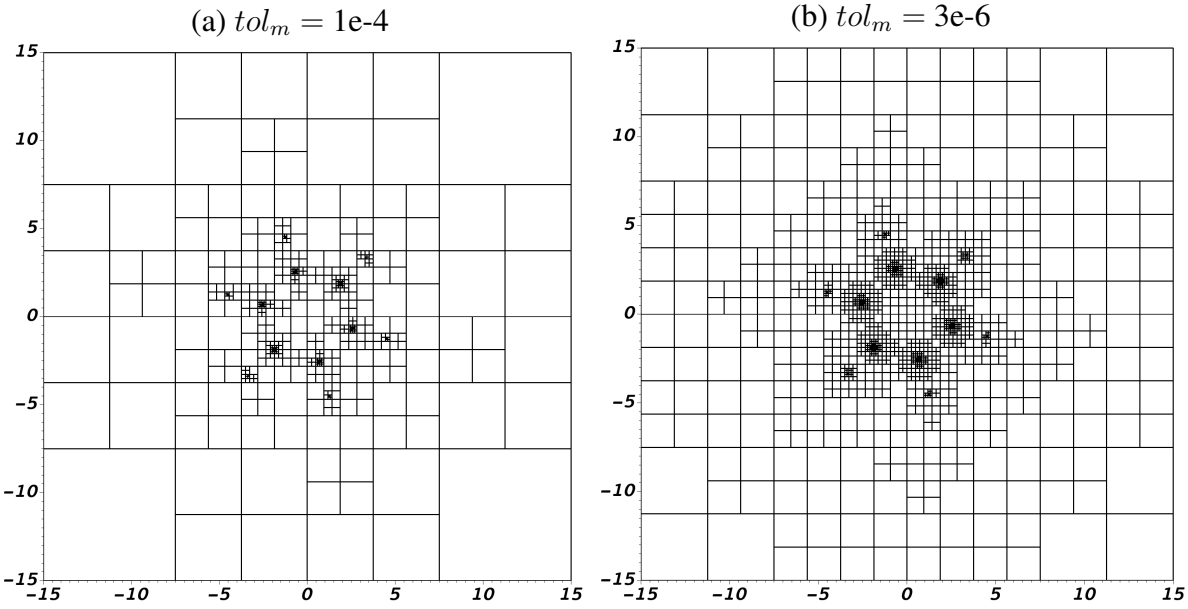


Figure 3.3: Example of the mesh refinement scheme for the benzene molecule ($C_6H_6$). 2D slices of the mesh are shown in the plane of the molecule generated with 4th order quadrature in Eq. (3.42) and (a) $tol_m$ =1e-4, (b) $tol_m$ =3e-6.

### 3.4.3 Singularity Subtraction

Achieving the necessary accuracy for DFT calculations requires careful treatment of the singular integrals arising in Green Iteration,

$$\psi^{(n+1)}(\mathbf{r}) = \mathcal{G}(\varepsilon^{(n)})\psi^{(n)}(\mathbf{r}) = -\int V_{eff}(\mathbf{r}')\psi^{(n)}(\mathbf{r}')\frac{e^{-\sqrt{-2\varepsilon^{(n)}}|\mathbf{r}-\mathbf{r}'|}}{2\pi|\mathbf{r}-\mathbf{r}'|}d\mathbf{r}', \tag{3.43}$$

and the Hartree potential,

$$V_H(\mathbf{r}) = \int \frac{\rho(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|} d\mathbf{r}'. \tag{3.44}$$

The singular $\mathbf{r}' = \mathbf{r}$ term in the quadrature sums is skipped, but this error due to skipping the singularity is reduced by weakening the singularities before discretization. For the integral involving the Yukawa kernel in Eq. (3.43) we implemented a standard singularity subtraction scheme [108, 109],

$$\int f(\mathbf{r}') \frac{e^{-k|\mathbf{r}-\mathbf{r}'|}}{|\mathbf{r}-\mathbf{r}'|} d\mathbf{r}' = \int \left(f(\mathbf{r}') - f(\mathbf{r})\right) \frac{e^{-k|\mathbf{r}-\mathbf{r}'|}}{|\mathbf{r}-\mathbf{r}'|} d\mathbf{r}' + f(\mathbf{r}) \int \frac{e^{-k|\mathbf{r}-\mathbf{r}'|}}{|\mathbf{r}-\mathbf{r}'|} d\mathbf{r}'. \tag{3.45}$$

The second term on the right in Eq. (3.45) is evaluated analytically,

$$f(\mathbf{r}) \int \frac{e^{-k|\mathbf{r}-\mathbf{r}'|}}{|\mathbf{r}-\mathbf{r}'|} d\mathbf{r}' = \frac{4\pi f(\mathbf{r})}{k^2}, \tag{3.46}$$

while the singularity in the first term on the right has been weakened, so the quadrature scheme yields a more accurate result. Note however that the exponential decay rate in the Yukawa kernel is $k = \sqrt{-2\varepsilon}$, and a problem arises if $\varepsilon \to 0$, since in that case the singularity subtraction scheme in Eq. (3.45) tends to the indeterminate form $\infty - \infty$. This is resolved by introducing a constant gauge shift $V_{shift}$ in the effective potential,

$$V_{eff}(\mathbf{r}) \to V_{eff}(\mathbf{r}) + V_{shift}. \tag{3.47}$$

The wavefunctions are unaffected and the eigenvalues simply shift by this amount (the shift is removed before computing energies). Throughout this work we set $V_{shift} = -0.5$, ensuring that the eigenvalues of the occupied states are bounded away from zero.

The scheme described above however does not work for the Hartree potential in Eq. (3.44) which corresponds to $k = 0$. In this case, we employ a modified form of singularity subtraction using a Gaussian function,

$$\int f(\mathbf{r}') \frac{1}{|\mathbf{r}-\mathbf{r}'|} d\mathbf{r}' = \int \left(f(\mathbf{r}') - f(\mathbf{r}) e^{-|\mathbf{r}-\mathbf{r}'|^2/\alpha^2}\right) \frac{1}{|\mathbf{r}-\mathbf{r}'|} d\mathbf{r}' + f(\mathbf{r}) \int \frac{e^{-|\mathbf{r}-\mathbf{r}'|^2/\alpha^2}}{|\mathbf{r}-\mathbf{r}'|} d\mathbf{r}', \tag{3.48}$$

where $\alpha$ is a scaling parameter. As before, the second term on the right is evaluated analytically,

$$f(\mathbf{r}) \int \frac{e^{-|\mathbf{r}-\mathbf{r}'|^2/\alpha^2}}{|\mathbf{r}-\mathbf{r}'|} d\mathbf{r}' = 2\pi f(\mathbf{r}) \alpha^2, \tag{3.49}$$

and the singularity in the first term has been weakened. In addition, the Gaussian remains smooth

for $\mathbf{r}' \to \mathbf{r}$, unlike other options, and this ensures the accuracy of the quadrature scheme.

The choice of the scaling factor $\alpha$ is guided by the following considerations. Recall that the first integral on the right in Eq. (3.48) is computed using a quadrature scheme on a truncated computational domain, and hence the function $f(\mathbf{r})e^{-|\mathbf{r}-\mathbf{r}'|^2/\alpha^2}$ should have certain properties. If $\alpha$ is small, then the Gaussian is narrow and the quadrature scheme would struggle to resolve the variation in this function. On the other hand if $\alpha$ is large, then the Gaussian is wide and this would require increasing the size of the computational domain. In more detail, the function $f(\mathbf{r})e^{-|\mathbf{r}-\mathbf{r}'|^2/\alpha^2}$ must be sufficiently small when $\mathbf{r}'$ is near the domain boundary to ensure that the effect of the domain truncation is small; there are two cases, (1) when $\mathbf{r}$ lies in the domain interior, then $f(\mathbf{r})$ is not necessarily small, but $e^{-|\mathbf{r}-\mathbf{r}'|^2/\alpha^2}$ is small as long as $\alpha$ is not too large, (2) when $\mathbf{r}$ lies near the domain boundary, then $f(\mathbf{r})$ is small while $e^{-|\mathbf{r}-\mathbf{r}'|^2/\alpha^2}$ is bounded. The conclusion is that the Gaussian scaling factor $\alpha$ should not be too small in relation to the spatial discretization and should not be too large in relation to the computational domain size; this work uses domains of size $[-20, 20]^3$ a.u. to $[-30, 30]^3$ a.u. with $\alpha = 1$ a.u., which was determined empirically.

### 3.4.4 Gradient-Free Eigenvalue Update

Recall that line 4 in Green Iteration updates the eigenvalue $\varepsilon_i^{(n+1)}$; in this subsection we describe three methods for this purpose. The first method uses the Rayleigh quotient [101],

$$\varepsilon_i^{(n+1)} = \frac{\langle \psi_i^{(n+1)}, \mathcal{H}\psi_i^{(n+1)} \rangle}{\langle \psi_i^{(n+1)}, \psi_i^{(n+1)} \rangle} = \frac{-\frac{1}{2}\langle \psi_i^{(n+1)}, \nabla^2 \psi_i^{(n+1)} \rangle + \langle \psi_i^{(n+1)}, V_{eff}\psi_i^{(n+1)} \rangle}{\langle \psi_i^{(n+1)}, \psi_i^{(n+1)} \rangle}, \tag{3.50}$$

where $\mathcal{H}$ is the Kohn-Sham differential operator defined in Eq. (3.1), $V_{eff}$ is the effective potential in the current SCF, and $\psi_i^{(n+1)}$ is the wavefunction computed in line 3 of Green Iteration. The second method applies integration by parts in Eq. (3.50) to obtain,

$$\varepsilon_i^{(n+1)} = \frac{\frac{1}{2}\langle \nabla\psi_i^{(n+1)}, \nabla\psi_i^{(n+1)} \rangle + \langle \psi_i^{(n+1)}, V_{eff}\psi_i^{(n+1)} \rangle}{\langle \psi_i^{(n+1)}, \psi_i^{(n+1)} \rangle}. \tag{3.51}$$

In the present framework the gradient $\nabla\psi_i^{(n+1)}$ in Eq. (3.51) and Laplacian $\nabla^2\psi_i^{(n+1)}$ in Eq. (3.50) are computed by spectral differentiation using the values of the wavefunction at the Chebyshev points in each cell [107]. The third method is a gradient-free update suggested by Harrison et al. [99],

$$\varepsilon_i^{(n+1)} = \varepsilon_i^{(n)} - \frac{\langle V_{eff}\psi_i^{(n)}, \psi_i^{(n)} - \psi_i^{(n+1)} \rangle}{\langle \psi_i^{(n+1)}, \psi_i^{(n+1)} \rangle}. \tag{3.52}$$

In this case, which computes a $\Delta\varepsilon_i$ rather than $\varepsilon_i^{(n+1)}$ itself, the initial guess $\varepsilon_i^{(0)}$ in the first SCF iteration can be given using either Eq. (3.50) or Eq. (3.51). The gradient-free eigenvalue

update enables the total energy to also be computed in a gradient-free manner using the alternative expression,

$$E = E_{band} - E_H + E_{xc} - \int \rho(\mathbf{r}) V_{xc}[\rho](\mathbf{r}) d\mathbf{r} + E_{ZZ}, \tag{3.53}$$

where the band energy is the weighted sum of the eigenvalues,

$$E_{band} = 2 \sum_{i=1}^{N_w} f(\varepsilon_i, \mu_F) \varepsilon_i. \tag{3.54}$$

In contrast to the original expression for the total energy in Eq. (3.12), the gradient-free expression in Eq. (3.53) avoids explicitly computing the kinetic energy $E_{kin}$ in Eq. (3.13) which contains the Laplacian; the kinetic energy is now contained implicitly in the band energy, which is obtained with the gradient-free method. Later below we show that the gradient-free method has the best accuracy of the three approaches described here.

### 3.4.5 Accuracy Results

This subsection demonstrates the effects of the previously described numerical techniques on the carbon monoxide molecule ($N_A = 2$, $N_e = 14$, $N_w = 8$). The computations use domain $[-20, 20]^3$ a.u., temperature $T = 200\ K$, gauge shift $V_{shift} = -0.5$, Green iteration tolerance $tol_{gi}$ =1e-7, SCF tolerance $tol_{scf}$ = 1e-6, and Anderson mixing parameter $\beta = 0.5$. Except where specified, the computations use singularity subtraction and the gradient-free eigenvalue update. We report the energy error $|E_{TAGI} - E_{ref}|$, where $E_{TAGI}$ is computed using TAGI and $E_{ref} = -112.47193$ Ha is the reference value converged to 1e-4 Ha, which was computed using DFT-FE [37, 40].

#### 3.4.5.1 Quadrature Rule and Adaptive Mesh Refinement Scheme

We first demonstrate the effect of the order $p$ of the quadrature rule and the tolerance $tol_m$ of the adaptive mesh refinement scheme described in section 3.4.2. To test the effect of the quadrature rule order we generate a mesh using order $p = 4$ and tolerance $tol_m$ = 3e-7, and then on this mesh the order $p$ is varied; Table 3.1(a) shows that the error is reduced from 1.313 mHa with $p = 4$ to 0.179 mHa with $p = 7$. To test the effect of the mesh refinement tolerance we fix the quadrature order to $p = 4$ and vary the mesh refinement tolerance $tol_m$; Table 3.1(b) shows that the error is reduced from 3.946 mHa with $tol_m$ = 3e-6 to 0.674 mHa with $tol_m$ = 1e-7.

#### 3.4.5.2 Singularity Subtraction

Next we demonstrate the effect of the singularity subtraction schemes described in section 3.4.3. The quadrature order is set to $p = 4$ and a sequence of mesh refinements is performed. The ground

(a)

| $p$ | $tol_m$ | # Cells | # Points | Error (mHa) |
|---|---|---|---|---|
| 4 | 3e-7 | 5293 | 661625 | 1.313 |
| 5 | 3e-7 | 5293 | 1143288 | 0.605 |
| 6 | 3e-7 | 5293 | 1815499 | 0.311 |
| 7 | 3e-7 | 5293 | 2710016 | 0.179 |

(b)

| $p$ | $tol_m$ | # Cells | # Points | Error (mHa) |
|---|---|---|---|---|
| 4 | 3e-6 | 2962 | 370250 | 3.946 |
| 4 | 1e-6 | 3676 | 459500 | 2.550 |
| 4 | 3e-7 | 5293 | 661625 | 1.313 |
| 4 | 1e-7 | 7428 | 928500 | 0.674 |

Table 3.1: Error in the total energy for the carbon monoxide molecule using a (a) fixed mesh, increasing quadrature order $p$ from 4 to 7, and (b) fixed quadrature order $p = 4$, decreasing mesh refinement parameter $tol_m$ from 3e-6 to 1e-7. The resulting number of cells and points in the adaptively refined mesh are given in columns 3 and 4.

state calculation is performed with and without singularity subtraction; in both cases the singular term in the discrete convolution sums is skipped. Table 3.2 shows that singularity subtraction yields a significant improvement in the accuracy of the total energy, over two orders of magnitude for mesh size $N_m = 661625$ which achieves chemical accuracy.

| | | | Error (mHa) | |
|---|---|---|---|---|
| $tol_m$ | # Cells | # Points | Non-SS | SS |
| 3e-6 | 2962 | 370250 | 823 | 3.946 |
| 1e-6 | 3676 | 459500 | 702 | 2.550 |
| 3e-7 | 5293 | 661625 | 558 | 1.313 |
| 1e-7 | 7428 | 928500 | 429 | 0.674 |

Table 3.2: Error in the total energy for the Carbon monoxide molecule without singularity subtraction (column 4) and with singularity subtraction (column 5).

#### 3.4.5.3 Gradient-Free Eigenvalue Update

Finally, we compare the eigenvalue update methods described in section 3.4.4, Laplacian update (Eq. (3.50)), gradient update (Eq. (3.51)), and gradient-free update (Eq. (3.52)). The ground state calculation is performed for a sequence of refined meshes. Figure 3.4(a) shows the energy error versus the number of mesh points $N_m$ for order $p = 4$ and figure 3.4(b) shows this for order $p = 6$. We make the following three observations. First, for a given mesh, the gradient-free update achieves significantly better accuracy than the gradient and Laplacian updates for both order $p = 4$ and $p = 6$, and as the mesh is refined the gradient-free update achieves chemical accuracy around $N_m = 600,000$. Second, for $p = 4$, the gradient update error saturates around 6

mHa/atom as the mesh is refined, indicating that the refinement scheme is not adequately refining the correct regions to reduce the error in the kinetic energy. Third, for $p = 6$, the gradient update recovers its convergence rate and is able to achieve chemical accuracy as the mesh is refined, indicating that the higher order gradients have reduced the error in the kinetic energy that was present for $p = 4$. We note that the adaptive mesh refinement scheme and choice of test function described in section 3.4.2 were developed using feedback from the gradient-free eigenvalue update. Different meshing schemes that prioritize accurate gradients or Laplacians of the wavefunctions could achieve better results for their respective eigenvalue update methods than this refinement scheme. Nevertheless, for each mesh refinement scheme we investigated we found the gradient-free update to be the most accurate and we use this update throughout the work.

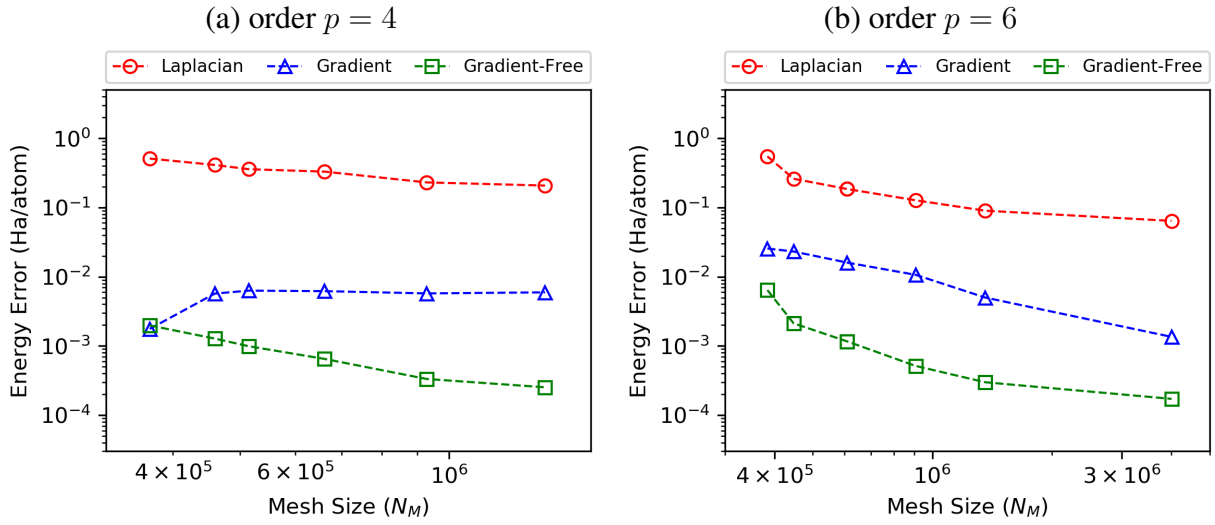


Figure 3.4: Error in the total energy per atom for the Carbon monoxide molecule versus the number of mesh points $N_m$ while using different eigenvalue update methods in Green Iteration for (a) quadrature order $p = 4$ and (b) quadrature order $p = 6$.

## 3.5 Convergence Rate of Green Iteration

Previously we explained how in each SCF iteration, the Kohn-Sham eigenproblem in Eq. (3.1) can be converted into a fixed-point problem for the integral operator in Eq. (3.20) and that the fixed-point problem is solved by Green Iteration. This section examines the convergence rate of Green Iteration; first an example exhibiting slow convergence is presented, then the cause of the problem is identified by reference to power iteration, and finally Anderson mixing is applied to the wavefunctions to accelerate convergence.

### 3.5.1 Observed Convergence Rates

To illustrate the slow convergence of Green Iteration, we consider the first SCF iteration for the carbon monoxide molecule. Figure 3.5 plots the residual of the first seven wavefunctions determined by Green Iteration versus the iteration number. In this case the first two wavefunctions converge rapidly, but the subsequent wavefunctions converge slowly; in particular the 4th wavefunction converges extremely slowly. The result is that Green Iteration requires a total of 1246 iterations to ensure that the first seven wavefunction residuals fall below 1e$-$8. This is a tighter tolerance than is used in practice, however it helps illustrate the issue. In the next subsection we examine the cause of this slow convergence.

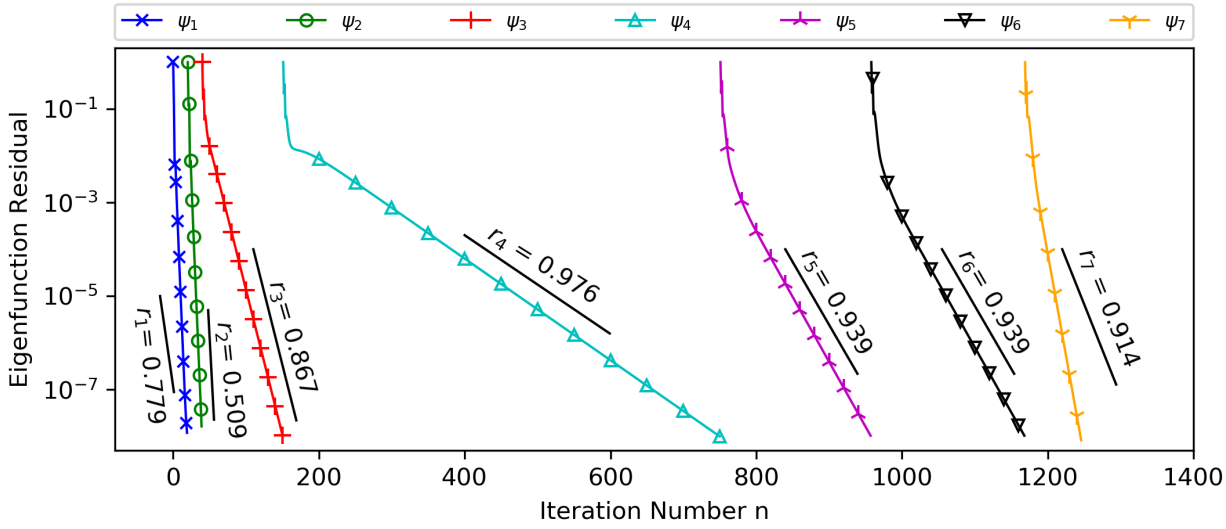


Figure 3.5: Convergence of the eigenfunction residual $||\psi_i^{(n+1)} - \psi_i^{(n)}||_2$ during Green Iteration in the first SCF iterations for the carbon monoxide molecule. The observed residuals (symbols) and the predicted convergence rates $r_i$ (black lines).

### 3.5.2 Convergence Analysis

Recall line 3 of Green Iteration (Algorithm 3.2), $\psi_i^{(n+1)} = \mathcal{G}(\varepsilon_i^{(n)})\psi_i^{(n)}$, which updates the $i$th eigenfunction using the operator defined in Eq. (3.21). The parameter $\varepsilon_i^{(n)}$ changes in each step of the iteration, but as $\varepsilon_i^{(n)} \to \varepsilon_i$, the scheme converges to power iteration for the operator $\mathcal{G}(\varepsilon_i)$ with deflation against the previously determined eigenfunctions $\psi_j, j < i$ as indicated in line 5 of the algorithm. This suggests that the convergence rate of $\psi_i^{(n)}$ depends on the spectral gap of $\mathcal{G}(\varepsilon_i)$, as is the case for power iteration [117]. To demonstrate this it is useful to define a 1-parameter family of curves $\mu_i(\varepsilon)$ and functions $\phi_i(\varepsilon)$ satisfying the linear eigenvalue equation,

$$\mathcal{G}(\varepsilon)\phi_i(\varepsilon) = \mu_i(\varepsilon)\phi_i(\varepsilon), \quad i = 1, \ldots, N_w, \tag{3.55}$$

subject to conditions specified below. Figure 3.6 shows the curves $\mu_i(\varepsilon)$ for the operator arising in the first SCF iteration of the carbon monoxide molecule, where the curves $\mu_i(\varepsilon)$ are plotted versus $\varepsilon$ for $i = 1 : 8$. Note that for each parameter value $\varepsilon$, the eigenvalues $\mu_i(\varepsilon)$ are computed by power

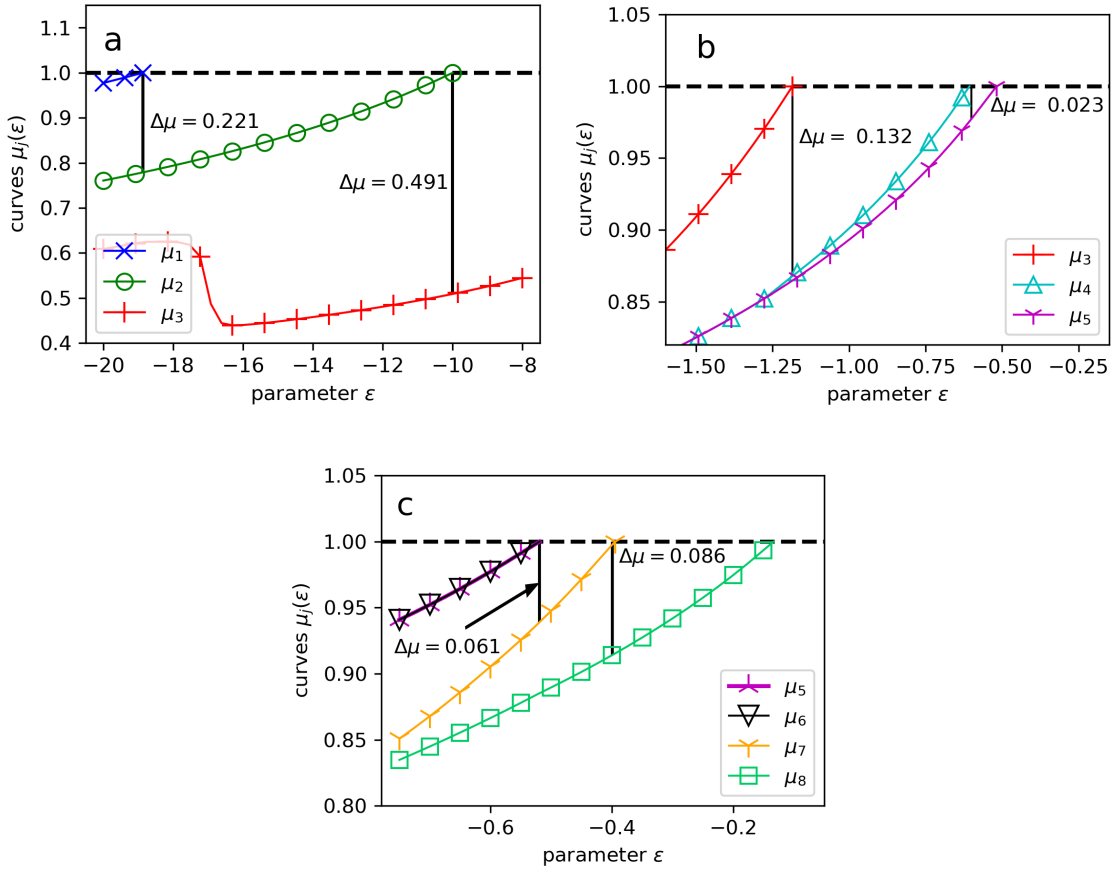


Figure 3.6: First SCF iteration for the Carbon monoxide molecule. The curves $\mu_i(\varepsilon)$ defined by eigenvalue problem Eq. (3.55) for the integral operator $\mathcal{G}(\varepsilon)$ are plotted versus parameter $\varepsilon$. Intersections with the dashed line $\mu = 1$ yield fixed-points $\varepsilon_i$ of Green Iteration. The spectral gap of the integral operator, $\Delta\mu_i = 1 - \mu_{i+1}(\varepsilon_i)$, is indicated at the fixed-points $\varepsilon_i$, (a) $i = 1, 2, 3$, (b) $i = 3, 4, 5$, (c) $i = 5, 6, 7, 8$. Numerical values are given in Table 3.3.

iteration applied to the operator $\mathcal{G}(\varepsilon)$, subject to the modified orthogonality condition stated below. The key observation is that the fixed-points of Green Iteration occur when one of the curves $\mu_i(\varepsilon)$ intersects the line $\mu = 1$. In particular, Fig. 3.6(a) shows fixed-points around $\varepsilon = -19$ (blue curve) and $\varepsilon = -10$ (green curve), Fig. 3.6(b) shows fixed-points around $\varepsilon = -1.2$ (red curve), $\varepsilon = -0.6$ (cyan curve), and $\varepsilon = -0.5$ (purple curve), and so on. As described below, the spectral properties of the integral operator $\mathcal{G}(\varepsilon)$ at these fixed-point values of $\varepsilon$ affect the convergence rate of Green Iteration.

The following conditions are applied to the integral operator eigenpairs $(\mu_i, \phi_i)$. For each $\varepsilon$, the eigenvalues are ordered by their magnitude $\mu_1(\varepsilon) \geq \cdots \geq \mu_{N_w}(\varepsilon)$. Note that if $\mu_i(\varepsilon) = 1$ for some

index $i$ and parameter value $\varepsilon$, then Eq. (3.55) reduces to the fixed-point problem in Eq. (3.20), $\psi_i = \mathcal{G}(\varepsilon_i)\psi_i$, in which case we have $\varepsilon = \varepsilon_i$ and $\phi_i(\varepsilon) = \psi_i$ [101, 103]. In addition, the usual orthogonality condition for power iteration, $\phi_i(\varepsilon) \perp \phi_j(\varepsilon)$ for $i \neq j$, is modified to be consistent with the deflation step in Green Iteration; that is, $\psi_i \perp \phi_j(\varepsilon)$ for $i < j$ and $\varepsilon_i < \varepsilon$. Hence, we terminate the curves in Fig. 3.6 when $\mu_i(\varepsilon_i) = 1$ and $\phi_i(\varepsilon_i) = \psi_i$; all subsequent orthogonalization for $\varepsilon > \varepsilon_i$ will use $\psi_i$.

Figure 3.6 also indicates the spectral gap of the operator $\mathcal{G}(\varepsilon_i)$, defined by $\Delta\mu_i = 1 - \mu_{i+1}(\varepsilon_i)$; due to the continuity of the curves $\mu_i(\varepsilon)$, these are correlated with the spectral gap of the KS-Hamiltonian, defined by $\Delta\varepsilon_i = \varepsilon_{i+1} - \varepsilon_i$; hence both gaps are relatively large for $i = 1, 2$ in Fig. 3.6(a), and relatively small for $i = 3, ..., 7$ in Fig. 3.6(b,c). Note further that the spectrum of the CO molecule contains a degeneracy; $\varepsilon_5 = \varepsilon_6$, hence $\psi_5$ and $\psi_6$ span a degenerate subspace. This degeneracy manifests itself in the spectral analysis in several ways. First, $\psi_5$ and $\psi_6$ converge with identical rates in Green Iteration (Fig. 3.5 parallel purple and black), and second, the $\mu_5(\varepsilon)$ and $\mu_6(\varepsilon)$ curves are identical (Fig. 3.6(c) overlapping purple and black). In the case of a degeneracy, the convergence rate of the wavefunctions to the degenerate subspace is governed by the spectral gap to the next distinct eigenvalue. In this example, we define the spectral gaps $\Delta\mu_5$ and $\Delta\mu_6$ with respect to the 7th eigenvalue, $\Delta\mu_5 = 1 - \mu_7(\varepsilon_5)$ and $\Delta\mu_6 = 1 - \mu_7(\varepsilon_6)$. Finally, note that at a fixed-point parameter $\varepsilon_i$, the largest eigenvalue of $\mathcal{G}(\varepsilon_i)$ is $\mu_i = 1$, so the convergence rate of power iteration is $r_i = \mu_{i+1}/\mu_i = 1 - \Delta\mu_i$; hence a large gap $\Delta\mu_i$ leads to rapid convergence of $\psi_i$ and a small gap $\Delta\mu_i$ leads to slow convergence.

Table 3.3 gives the values of the fixed-points, the spectral gaps, the observed and predicted convergence rates, the accuracy of the predictions, and the number of iterations required to achieve the 1e-8 tolerance in Green Iteration. The predicted convergence rates $r_i$, also shown in Fig. 3.5, were computed using the power iteration considerations above, $r_i = \mu_{i+1}/\mu_i = 1 - \Delta\mu_i$. In several cases ($\psi_1,\ \psi_2,\ \psi_7$), the observed convergence is faster than the predicted rate; this is attributed to the iteration not entering the asymptotic power-iteration regime before the tolerance was met. In the slower converging cases ($\psi_3,\ \psi_4,\ \psi_5,\ \psi_6$), the predicted convergence rates accurately agree with the observed rates, with percent errors $0.115\%$, $0.082\%$, $0.053\%$, and $0.053\%$, confirming that the convergence rates of the eigenfunctions $\psi_i$ in Green Iteration are controlled by the spectral gap $\Delta\mu_i$ in the integral operator. These spectral gaps in the integral operator are correlated to the spectral gaps $\Delta\varepsilon_i$ in the differential operator by the continuity of $\mu_i(\varepsilon)$. Hence, Green Iteration may converge slowly whenever a small spectral gap exists in the Hamiltonian; the next subsection describes a method to overcome this drawback.

| | Spectral Gaps | | | Convergence Rates | | | |
|---|---|---|---|---|---|---|---|
| index, $i$ | $\varepsilon_i$ | $\Delta\varepsilon_i$ | $\Delta\mu_i$ | observed $r_i$ | predicted $r_i$ | % error | Number of Iterations |
| 1 | -18.870 | 8.862 | 0.221 | 0.460 | 0.779 | 69.3 | 19 |
| 2 | -10.008 | 8.822 | 0.491 | 0.450 | 0.509 | 13.1 | 20 |
| 3 | -1.186 | 0.579 | 0.132 | 0.867 | 0.868 | 0.115 | 110 |
| 4 | -0.607 | 0.087 | 0.023 | 0.9752 | 0.976 | 0.082 | 601 |
| 5 | -0.520 | 0.124 | 0.061 | 0.9385 | 0.939 | 0.053 | 208 |
| 6 | -0.520 | 0.124 | 0.061 | 0.9385 | 0.939 | 0.053 | 211 |
| 7 | -0.396 | 0.262 | 0.086 | 0.819 | 0.914 | 11.6 | 77 |

Table 3.3: First SCF iteration for the Carbon monoxide molecule. Eigenvalue index, Hamiltonian eigenvalues $\varepsilon_i$, Hamiltonian spectral gap $\Delta\varepsilon_i$, integral operator spectral gap $\Delta\mu_i$, observed convergence rate, predicted convergence rate, accuracy of the prediction, number of iterations for the wavefunction to converge to 1e-8 tolerance.

### 3.5.3 Wavefunction Mixing

While Green Iteration resembles power iteration as noted above, it is a fixed-point iteration and hence is amenable to standard fixed-point acceleration techniques. We define the vector $\mathbf{x} = (\varepsilon, \psi)$, and the inner product between two vectors $\mathbf{x}_1 = (\varepsilon_1, \psi_1)$ and $\mathbf{x}_2 = (\varepsilon_2, \psi_2)$ to be $(\mathbf{x}_1, \mathbf{x}_2) = \varepsilon_1\varepsilon_2 + \int \psi_1(\mathbf{r})\psi_2(\mathbf{r})d\mathbf{r}$. We then use Anderson mixing to update the eigenpairs $(\varepsilon_i^{(n)}, \psi_i^{(n)})$ after each step of Green Iteration, in the same way that the electron density is updated after each step of the SCF iteration. Figure 3.7 shows the effect of applying Anderson mixing to the wavefunctions with mixing parameter $\beta = 0.5$, for the same computation as above, the first SCF iteration of the carbon monoxide molecule. The total number of iterations is reduced from 1246 (Green Iteration) to 188 (Green Iteration with wavefunction mixing). Wavefunctions $\psi_3 - \psi_6$ still converge the slowest, however they converge significantly faster than without Anderson mixing.

In practice the wavefunction mixing scheme requires a good initial guess to ensure convergence. In the first SCF iteration, to achieve a good initial guess, Green Iteration can be performed without wavefunction mixing until convergence to a user-defined tolerance is achieved, at which point the computed wavefunction is in the basin of attraction of the fixed-point scheme and Anderson wavefunction mixing can be safely applied. In subsequent SCF iterations the initial guess for the eigenpairs tend to be much better and delaying the use of wavefunction mixing is not necessary. Furthermore, the tolerance for Green Iteration $tol_{gi}$ does not have to be the same throughout an SCF iteration. We find that starting with a loose tolerance and gradually tightening it after each step in the SCF is beneficial. The gradual reduction of $tol_{gi}$ increases the number of steps in the SCF for the electron density to converge to $tol_{scf}$, but it significantly reduces the cost of the first few steps of the SCF iteration, and results in an overall reduction of computation time.

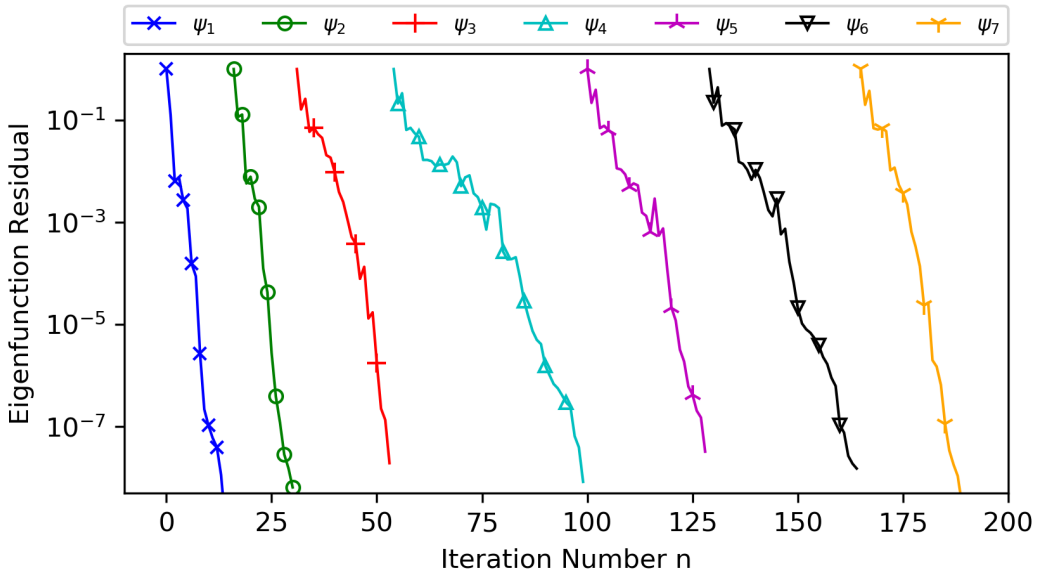


Figure 3.7: Convergence of the eigenfunction residual $||\psi_i^{(n+1)} - \psi_i^{(n)}||_2$ during Green Iteration in the first SCF iterations for the carbon monoxide molecule using wavefunction mixing with mixing parameter $\beta = 0.5$. Wavefunction mixing reduces the total number of iterations from 1246 to 188.

## 3.6 Treecode Acceleration

Treecode-Accelerated Green Iteration requires computing convolutions of the form,

$$\varphi(\mathbf{r}) = \int G(\mathbf{r}, \mathbf{r}') f(\mathbf{r}') d\mathbf{r}', \tag{3.56}$$

where $G(\mathbf{r}, \mathbf{r}')$ is either the Yukawa kernel in the integral operator $\mathcal{G}(\varepsilon)$ in Eq. (3.21) needed in line 3 of Green Iteration, or the Coulomb kernel in the Hartree potential in Eq. (3.3). For the Yukawa kernel, $\varphi(\mathbf{r}) = \psi(\mathbf{r})$ is the wavefunction and $f(\mathbf{r}) = V_{eff}[\rho](\mathbf{r})\psi(\mathbf{r})$ is the effective potential times the wavefunction, while for the Coulomb kernel, $\varphi(\mathbf{r}) = V_H[\rho](\mathbf{r})$ is the Hartree potential and $f(\mathbf{r}) = \rho(\mathbf{r})$ is the electron density. TAGI uses the barycentric treecodes described in Chapter 2, and while many of the details remain the same, there are some key differences due to the singularity subtraction schemes that warrant further explanation. Further, since we employ different singularity subtraction schemes for the Coulomb and Yukawa kernels, these each require separate treatment.

### 3.6.1 Discrete Singularity Subtraction Sums

In the case of the Yukawa kernel[1], the standard singularity subtraction scheme described in Section 3.4.3 gives

$$\varphi(\mathbf{r}) = f(\mathbf{r}) \int \frac{e^{-\kappa|\mathbf{r}-\mathbf{r}'|}}{|\mathbf{r}-\mathbf{r}'|} d\mathbf{r}' + \int [f(\mathbf{r}') - f(\mathbf{r})] \frac{e^{-\kappa|\mathbf{r}-\mathbf{r}'|}}{|\mathbf{r}-\mathbf{r}'|} d\mathbf{r}', \tag{3.57}$$

where the first term on the right hand side is evaluated analytically,

$$f(\mathbf{r}) \int \frac{e^{-\kappa|\mathbf{r}_i-\mathbf{r}'|}}{|\mathbf{r}_i-\mathbf{r}'|} d\mathbf{r}' = \frac{4\pi f(\mathbf{r})}{\kappa^2}, \tag{3.58}$$

while the singularity has been weakened in the second term on the right. Numerical discretization of this scheme on the finite computational domain gives

$$\varphi_i \approx \sum_{\substack{j=1 \\ i\neq j}}^{N_m} f_j \frac{e^{-\kappa|\mathbf{r}_i-\mathbf{r}_j|}}{|\mathbf{r}_i-\mathbf{r}_j|} w_j = \frac{4\pi f_i}{\kappa^2} + \sum_{\substack{j=1 \\ i\neq j}}^{N_m} [f_j - f_i] \frac{e^{-\kappa|\mathbf{r}_i-\mathbf{r}_j|}}{|\mathbf{r}_i-\mathbf{r}_j|} w_j, \tag{3.59}$$

where $\varphi_i \approx \varphi(\mathbf{r}_i)$, $f_j = f(\mathbf{r}_j)$, and $w_j$ are the quadrature weights. This scheme introduces an additional finite domain size effect; the added expression is evaluated analytically on $\mathbb{R}^3$ while the subtracted expression is evaluated discretely on the finite computational domain. Since, in this work, $f(\mathbf{r})$ decays with free-space boundary conditions, the computational domain can be chosen large enough that the truncation error in the discrete expression is negligible. Similarly, in the case of the Coulomb kernel, the singularity subtraction scheme developed in Section 3.4.3 gives

$$\varphi(\mathbf{r}) = f(\mathbf{r}) \int \frac{e^{-|\mathbf{r}-\mathbf{r}'|^2/\alpha^2}}{|\mathbf{r}-\mathbf{r}'|} d\mathbf{r}' + \int \left[ f(\mathbf{r}') - f(\mathbf{r}) e^{-|\mathbf{r}-\mathbf{r}'|^2/\alpha^2} \right] \frac{1}{|\mathbf{r}-\mathbf{r}'|} d\mathbf{r}', \tag{3.60}$$

where again the first term on the right hand side is evaluated analytically,

$$f(\mathbf{r}) \int \frac{e^{-|\mathbf{r}-\mathbf{r}'|^2/\alpha^2}}{|\mathbf{r}-\mathbf{r}'|} d\mathbf{r}' = 2\pi f(\mathbf{r}) \alpha^2, \tag{3.61}$$

and the singularity has been weakened in the second term. Numerical discretization gives

$$\varphi_i \approx \sum_{\substack{j=1 \\ i\neq j}}^{N_m} f_j \frac{1}{|\mathbf{r}_i-\mathbf{r}_j|} w_j = 2\pi f_i \alpha^2 + \sum_{\substack{j=1 \\ i\neq j}}^{N_m} \left[ f_j - f_i e^{-|\mathbf{r}_i-\mathbf{r}_j|^2/\alpha^2} \right] \frac{1}{|\mathbf{r}_i-\mathbf{r}_j|} w_j. \tag{3.62}$$

[1] In this section we use $\kappa$ as the Yukawa kernel parameter instead of $k$ to avoid ambiguity below with the interpolation point index $\mathbf{s}_\mathbf{k}$, $\mathbf{k} = (k_1, k_2, k_3)$.

As described in Chapter 2, computing $\varphi$ by direct summation requires $O(N_m^2)$ operations, and several methods have been developed to reduce the cost including the treecode [43] and fast multipole method [44]. This work employs a recently developed GPU accelerated barycentric Lagrange treecode (BLTC) [50, 52], adapted to the singularity subtraction schemes, which reduces the operation count to $O(N_m \log N_m)$ using barycentric Lagrange interpolation [65]. Following convention, throughout this section the points $\mathbf{r}_i$ are referred to as target particles, the points $\mathbf{r}_j$ are referred to as source particles, and Eq. (3.62) and Eq. (3.59) express the particle-particle interactions. Below we describe how we extend the work from Chapter 2 to accommodate the singularity subtraction kernels used for the discrete convolution integrals in this work and present accuracy and timing results specific to the singularity subtraction kernels and the adaptively refined meshes used for all-electron TAGI calculations. Chapter 2 and the following references can be consulted for more details [50, 52, 49].

### 3.6.2 Source Clusters and Target Batches

Recall from Chapter 2, the treecode starts by dividing the source particles into a hierarchical tree of source clusters, where the root cluster is the minimal bounding box enclosing the computational domain. In this case, the root consists of all $N_m$ quadrature points. The root is then divided into child clusters by bisection in each dimension, and the child clusters are recursively subdivided until they contain fewer than $N_L$ particles; these are the leaves of the tree. After division each cluster is shrunk to the minimal bounding box containing its particles. Note that the source clusters in the treecode are rectangular boxes, and in general they are different than the cells in the adaptive mesh. As described above, for efficiency purposes on GPUs, the target particles are also organized into a set of localized batches containing fewer than $N_B$ particles, and then the particle-particle interactions are organized into batch-cluster interactions between the target particles in a batch and the source particles in a cluster. In this work we set $N_B = N_L$, and since the target particles and source particles correspond to the same set (the $N_m$ quadrature points), the target batches are equivalent to the leaf source clusters in the tree.

### 3.6.3 Particle-Cluster Approximation by Barycentric Lagrange Interpolation

We now describe the particle-cluster approximations for the Coulomb and Yukawa singularity subtraction kernels. Due to the differences in the singularity subtractions schemes, the form of the particle-cluster approximation differs for the two kernels. We begin with the Yukawa kernel, as it requires less modification from the work presented in Chapter 2.

**Particle-cluster approximation of singularity subtraction Yukawa kernel**. The sum in the right hand side of Eq. (3.59) can be rewritten as

$$\varphi_i \approx \frac{4\pi f_i}{\kappa^2} + \sum_{\substack{j=1 \\ i\neq j}}^{N_m} [f_j - f_i] \frac{e^{-\kappa|\mathbf{r}_i-\mathbf{r}_j|}}{|\mathbf{r}_i-\mathbf{r}_j|} w_j = \frac{4\pi f_i}{\kappa^2} + \sum_{\mathcal{C}} \varphi(\mathbf{r}_i, C), \tag{3.63}$$

where the second sum is taken over a set of source clusters $\mathcal{C}$ according to the interaction lists described in Chapter 2, and

$$\varphi(\mathbf{r}_i, C) = \sum_{\substack{\mathbf{r}_j\in C \\ \mathbf{r}_i\neq\mathbf{r}_j}} [f_j - f_i] \frac{e^{-\kappa|\mathbf{r}_i-\mathbf{r}_j|}}{|\mathbf{r}_i-\mathbf{r}_j|} w_j \tag{3.64}$$

is the interaction between a target particle $\mathbf{r}_i$ and a source cluster $C = \{\mathbf{r}_j\}$. To accommodate the singularity subtraction scheme, we construct two sets of modified charges,

$$\widehat{f}^1_{\mathbf{k}} = \sum_{\mathbf{r}_j\in C} L_{k_1}(x_j)L_{k_2}(y_j)L_{k_3}(z_j) f_j w_j, \quad \widehat{f}^2_{\mathbf{k}} = \sum_{\mathbf{r}_j\in C} L_{k_1}(x_j)L_{k_2}(y_j)L_{k_3}(z_j) w_j, \tag{3.65}$$

where $\mathbf{r}_j = (x_j, y_j, z_j)$ is a source particle, $\mathbf{s}_{\mathbf{k}} = (s_{k_1}, s_{k_2}, s_{k_3})$ is a tensor product grid of interpolation points, and $L_k(t)$ are the 1D Lagrange interpolating polynomials defined in Eq. (2.3). Note that the first set of modified charges correspond to the discrete field $f_j w_j$, or the continuous field $f(\mathbf{r}')d\mathbf{r}'$, while the second set of modified charges, once multiplied by $f_i$, corresponds to the discrete field $f_i w_j$, or the continuous field $f(\mathbf{r})d\mathbf{r}'$. Then, using 3D polynomial interpolation as in Chapter 2, we approximate the particle-cluster interaction in Eq. (3.64) by

$$\varphi(\mathbf{r}_i, C) \approx \sum_{\mathbf{k}} \frac{e^{-\kappa|\mathbf{r}_i-\mathbf{s}_{\mathbf{k}}|}}{|\mathbf{r}_i-\mathbf{s}_{\mathbf{k}}|} (\widehat{f}^1_{\mathbf{k}} - f_i \cdot \widehat{f}^2_{\mathbf{k}}). \tag{3.66}$$

Hence, to accommodate the Yukawa singularity subtraction scheme, all that is required is to compute an additional set of weights $\widehat{f}^2_{\mathbf{k}}$ and to replace $\widehat{f^{\mathbf{k}}}$ in Eq. (2.11) with $(\widehat{f}^1_{\mathbf{k}} - f_i \cdot \widehat{f}^2_{\mathbf{k}})$.

**Particle-cluster approximation of singularity subtraction Coulomb kernel**. The sum in the right hand side of Eq. (3.62) can be rewritten as

$$\varphi_i \approx 2\pi f_i \alpha^2 + \sum_{\substack{j=1 \\ i\neq j}}^{N_m} \left[ f_j - f_i e^{-|\mathbf{r}_i-\mathbf{r}_j|^2/\alpha^2} \right] \frac{1}{|\mathbf{r}_i-\mathbf{r}_j|} w_j = 2\pi f_i \alpha^2 + \sum_{\mathcal{C}} \varphi(\mathbf{r}_i, C), \tag{3.67}$$

where the second sum is taken over a set of source clusters $\mathcal{C}$, and

$$\varphi(\mathbf{r}_i, C) = \sum_{\substack{\mathbf{r}_j \in C \\ \mathbf{r}_i \neq \mathbf{r}_j}} \left[ f_j - f_i e^{-|\mathbf{r}_i - \mathbf{r}_j|^2/\alpha^2} \right] \frac{1}{|\mathbf{r}_i - \mathbf{r}_j|} w_j \tag{3.68}$$

is the interaction between a target particle $\mathbf{r}_i$ and a source cluster $C = \{\mathbf{r}_j\}$. Again, we construct the two sets of modified charges in Eq. (3.65) and then approximate the particle-cluster interaction by

$$\varphi(\mathbf{r}_i, C) \approx \sum_{\mathbf{k}} (\widehat{f}_k^1 G^1(\mathbf{r}_i, \mathbf{s}_\mathbf{k}) - f_i \cdot \widehat{f}_k^2 G^2(\mathbf{r}_i, \mathbf{s}_\mathbf{k})), \tag{3.69}$$

where

$$G^1(\mathbf{r}_i, \mathbf{s}_\mathbf{k}) = \frac{1}{|\mathbf{r}_i - \mathbf{s}_\mathbf{k}|} \tag{3.70}$$

is the original Coulomb kernel, and

$$G^2(\mathbf{r}_i, \mathbf{s}_\mathbf{k}) = \frac{e^{-|\mathbf{r}_i - \mathbf{s}_\mathbf{k}|^2/\alpha^2}}{|\mathbf{r}_i - \mathbf{s}_\mathbf{k}|} \tag{3.71}$$

is the Gaussian-augmented kernel. Hence, accommodating the singularity subtraction scheme for the Coulomb kernel requires more modification than the scheme for the Yukawa kernel. In particular, in addition to computing the additional set of weights $\widehat{f}_\mathbf{k}^2$, the approximation now uses two different kernel evaluations, $G^1(\mathbf{r}_i, \mathbf{s}_\mathbf{k})$ and $G^2(\mathbf{r}_i, \mathbf{s}_\mathbf{k})$. It should be noted that while the singularity subtraction scheme is essential for efficiently achieving high accuracy with respect to mesh refinement, it does introduce additional floating point operations in the computation of each interaction. Nevertheless, the singularity subtraction schemes reduce the mesh refinement requirements drastically, as shown in Table 3.2, and are well worth the additional operations per interaction.

It is important to note that the approximations in Eq. (3.66) and Eq. (3.69) have the same direct sum structure as their counterpart exact interactions in Eq. (3.64) and Eq. (3.68). In one case the target particle $\mathbf{r}_i$ interacts with the source particles $\mathbf{r}_j$, and in the other it interacts with the interpolation points $\mathbf{s}_\mathbf{k}$; however in both cases the necessary kernel evaluations are independent from one another and can be efficiently computed in parallel on a GPU. As explained in Chapter 2, this is an additional level of parallelism not available to other fast summation schemes based on analytic series expansions, such as the Taylor treecode [49] where the approximations are recursive, and it is this additional level of parallelism that enables efficient GPU calculations using the barycentric treecodes. A further point is that the approximation weights $\widehat{f}_\mathbf{k}^1$ and $\widehat{f}_\mathbf{k}^2$ are independent of the target particle $\mathbf{r}_i$, so they can be precomputed and reused for different targets.

### 3.6.4 Treecode Algorithm

As described in Chapter 2, particle-cluster interactions $\varphi(\mathbf{r}_i, C)$ are organized into batch-cluster interactions, where the decision on whether or not to apply the approximation is given by the Multipole Acceptance Criteria (MAC) in Eq. (2.23) and is diagrammed in Fig. 2.3. The primary difference for Chapter 3 is that instead of the targets being randomly distributed particles, they are structured quadrature points coming from the adaptive mesh refinement scheme and quadrature rule described above. As a reminder, the BLTC uses Chebyshev points of the 2nd kind for interpolation points, given in Eq. (2.5) and (2.6), and the barycentric form of the Lagrange interpolating polynomial [65] given in Eq. (2.3). Figure 3.8 shows a 2D example of a cluster $C$ comprised of seven quadrature cells; Fig. 3.8(a) shows the source particles $\mathbf{r}_j$ in $C$ (these are quadrature points in the adaptive mesh, here defined with order $p = 2$), and Fig. 3.8(b) shows the Chebyshev grid of interpolation points $\mathbf{s_k}$ in $C$ (here defined with degree $n = 3$, these represent the cluster through the particle-cluster approximations in Eq. (3.66) and Eq. (3.69)). The "charge" values on the source particles are $f_j w_j$, and the "modified charge" values on the interpolation points are $\widehat{f}^1_{\mathbf{k}}$ and $\widehat{f}^2_{\mathbf{k}}$ which are computed with Eq. (3.65).

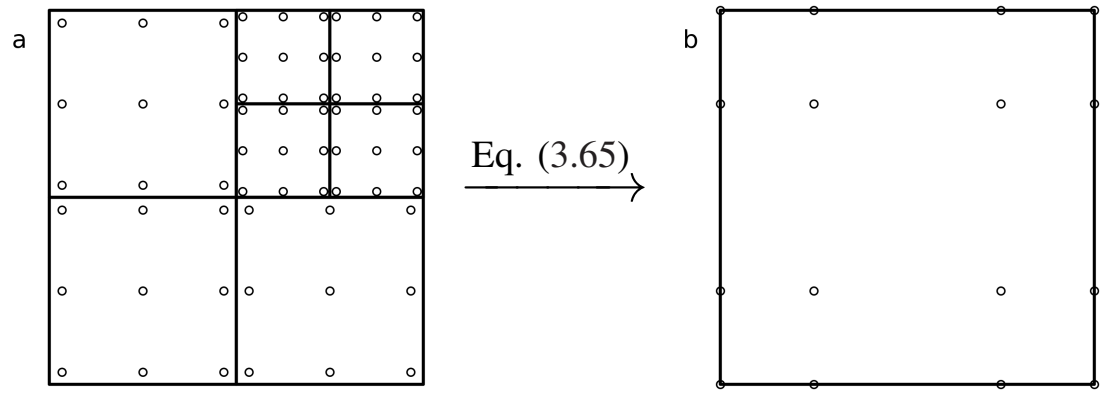


Figure 3.8: Example of a cluster $C$ in 2D, (a) source particles $\mathbf{r}_j$ in $C$ (these are quadrature points in the adaptive mesh, here defined with order $p = 2$), (b) Chebyshev grid of interpolation points $\mathbf{s_k}$ (here defined with degree $n = 3$, the weights at each interpolation point are computed in Eq. (3.65)).

The treecode has three options for a batch-cluster interaction. (1) If the cluster is not well separated from the batch, and has no children, then the batch-cluster interaction is computed by direct summation. Each target particle will interact with each of the quadrature points in Fig. 3.8(a). As this cluster contains 7 cells each containing 9 quadrature points, this requires 63 interactions per target. (2) If the cluster is not well separated from the batch, but has children, then the batch can recursively interact with each of the child clusters. (3) If the cluster is well-separated from the batch, then the batch-cluster interaction is computed by the approximations in Eq. (3.66) and (3.69).

Each target particle interacts with the interpolation points, resulting in 16 interactions per target. Hence, this approximation is cheaper than the direct interaction and should be used whenever the target batch is well separated from the cluster. In general, the approximation is more efficient when the number of quadrature points in a cluster $N_C$ is larger than the number of interpolation points $(n+1)^3$, as in this example.

In general, the treecode algorithm is the same as the algorithm presented in Chapter 2, Alg. 2.1, with the following minor modifications to the procedures and their interpretations:

- randomly distributed particles are replaced with structured quadrature points arising from the adaptive mesh refinement and quadrature scheme described above.
- particle charges $q_i$ are replaced by fields times quadrature weights $f_i w_i$.
- two sets of "modified charges" are computed, $\widehat{f}^1_{\mathbf{k}}$ and $\widehat{f}^2_{\mathbf{k}}$ in Eq. (3.65).
- interactions are computed using the singularity subtraction schemes in Eq. (3.66) and (3.69)

Next we document the treecode accuracy and efficiency for the discrete convolutions arising for the adaptively refined mesh and using the singularity subtraction kernels.

### 3.6.5 Treecode Accuracy

The sums in Eq. (3.59) and Eq. (3.62) are discretizations of convolution integrals given in Eq. (3.57) and (3.60) and introduce discretization error. This discretization error is controlled with the adaptive mesh refinement, the quadrature rule, and the singularity subtraction schemes in order to achieve chemical accuracy. Following discretization, we then compute fast approximations to these discrete sums using the treecode, which introduces another error called the treecode approximation error. It is important to ensure that the treecode approximation error is less than the discretization error so that we achieve treecode-acceleration while maintaining chemical accuracy in the calculations.

We document the accuracy of the treecode for the carbon monoxide molecule with domain $[-20, 20]^3$ a.u., quadrature order $p = 4$, mesh refinement tolerance $tol_m$ =3e-7, SCF tolerance $tol_{scf}$ =1e-5, and Green Iteration tolerance $tol_{gi}$ =1e-6. In this case the number of mesh points is $N_m = 661625$. We compute the ground-state energy with and without the treecode, and record the discretization error $|E_{ref} - E_{ds}|/N_A$, and treecode approximation error $|E_{ds} - E_{tc}|/N_A$, where $E_{ref}$ is the reference energy computed by DFT-FE, and $E_{ds}, E_{tc}$ are computed by the present method using direct summation and the treecode, respectively. Throughout this work the source clusters and target batch size parameters are set to $N_L = N_B = 2000$. Figure 3.9 shows that the discretization error of the present method is $|E_{ref} - E_{ds}|/N_A$ = 6.56e-4 Ha/atom (red dashed line),

while the treecode approximation error is much smaller for a wide range of treecode parameters ($0.35 \leq \theta \leq 0.8$ and interpolation degree $n = 6, 8, 10$). This confirms that in this range of parameter values, the numerical errors introduced by the treecode do not upset the chemical accuracy of the discretization.

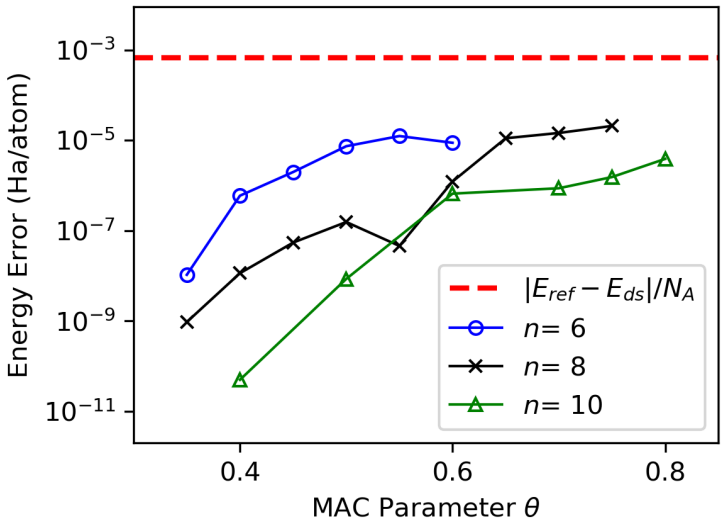


Figure 3.9: Comparison of the treecode approximation error to the underlying discretization error for the carbon monoxide molecule. The mesh contains $N_m = 661625$ points and results in a discretization error of $|E_{ref} - E_{ds}|/N_A =$ 6.56e-4 Ha/atom (red dashed line). Solid curves and symbols show the treecode approximation error $|E_{ds} - E_{tc}|/N_A$ for treecode MAC parameter $0.35 \leq \theta \leq 0.8$ and interpolation degree $n = 6, 8, 10$.

### 3.6.6 Treecode Efficiency on a 6-core CPU and Single GPU

**Relation to Chapter 2**. This section documents the BLTC efficiency on a multicore CPU and a single GPU. This investigation is similar to that of Chapter 2, however there are several key differences, in particular the particle distributions and the interaction kernels. First, in Chapter 2 the particles were randomly located according to a uniform distribution; this results in very well-balanced trees and relatively high speedups over the direct sum. In the present calculations, the particles come from a highly non-uniform adaptively refined mesh; in particular, the tree is significantly deeper near the two atomic nuclei than in the far field. Imbalanced trees tend to afford less speedup from the treecode. In particular, in this example, a large fraction of the particles are either very close to the oxygen atom or very close to the carbon atom. Particles near the same atom are not going to be well-separated at higher levels of the tree, and will have to interact either directly or at a deep level of the tree, reducing the speedup over direct sum. In this case, with only two atoms, each particle near an atom is not well separated from a substantial fraction of the

total set of particles. Nevertheless, we observe a speedup using the treecode which grows as the mesh size grows, and note that for larger molecular systems (more than two atoms), each particle will be well-separated from a larger fraction of the total set of particles and the expected treecode speedup is even larger. Second, the interaction kernels in this chapter are the singularity subtraction kernels described above. These kernel involve more floating point operations than those presented in Chapter 2; we consistently observe the trend that increasing the number of operations increases the computation time more on the CPU than on the GPU, resulting in even larger GPU versus CPU speedups.

**Hardware and parallelization schemes**. The following calculations were performed on a 6-core 2.6 GHz Intel Core i7 processor and a single NVIDIA Titan V GPU. Since this is a single-node calculation, we employ the shared memory parallelization approach described in Section 2.3.1 for the 6-core CPU calculations. As noted above, the GPU implementation takes advantage of the fact that the particle-cluster interactions in Eq. (3.64) and Eq. (3.68) and the approximations in Eq. (3.66) and Eq. (3.69) have the same direct sum structure involving independent kernel evaluations. The GPU processes the particle-particle interactions between the target batch and source cluster in parallel without thread divergence; this is because the MAC applies uniformly to all particles in a given target batch [52]. In practice, interaction lists are precomputed for each target batch to identify the source clusters interacting with the batch.

**Test calculation**. The performance of the BLTC on both platforms is demonstrated by computing the Hartree energy $E_H$ in Eq. (3.14) for the carbon monoxide molecule using the electron density from the first SCF iteration. Results are shown in Table 3.4 using direct summation and the treecode, for quadrature order $p = 4$ and mesh refinement parameter $tol_m$ between 1e-3 and 1e-8 yielding the indicated mesh size $N_m$. The direct sum energy values in the 3rd column converge as the mesh is refined, and the 4th column shows the corresponding discretization error using the value $E_H = 74.88578$ obtained with $tol_m$=1e-8 as the reference. The 5th column records the treecode approximation error for MAC $\theta = 0.7$ and degree $n = 8$; this is the difference between the value of $E_H$ computed by direct summation (column 3) and the value computed by the treecode (not shown). The results show that the treecode approximation error is well below the discretization error and within chemical accuracy.

The remainder of Table 3.4 records computation times on the 6-core CPU and GPU for direct summation (ds) and the treecode (tc). Averaging over the six runs in Table 3.4, direct summation runs 192 times faster on the GPU than on the 6-core CPU, while the treecode runs 70 times faster. On both platforms the treecode is faster than direct summation, and the speedup (ds/tc) increases as the mesh is refined; this is consistent with $O(N_m^2)$ scaling for direct summation and $O(N_m \log N_m)$ scaling for the treecode. In particular, for the largest mesh size with approximately 2.2 million mesh points, the treecode computation time on the GPU is less than 18 s, which is about 4.5 times faster

than direct summation.

| | | | | | 6-core CPU time (s) | | | GPU time (s) | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $tol_m$ | $N_m$ | $E_H$ (Ha) | ds error | tc error | ds | tc | ds/tc | ds | tc | ds/tc |
| 1e-3 | 141000 | 74.88640 | 6.20e-4 | 5.31e-7 | 70.42 | 26.24 | 2.68 | 0.39 | 0.47 | 0.82 |
| 1e-4 | 184750 | 74.87690 | 8.88e-3 | 1.20e-6 | 150.13 | 41.40 | 3.63 | 0.67 | 0.68 | 0.97 |
| 1e-5 | 249500 | 74.88668 | 9.00e-4 | 8.85e-6 | 216.17 | 80.32 | 2.69 | 1.13 | 1.13 | 1.00 |
| 1e-6 | 459500 | 74.88551 | 2.70e-4 | 3.68e-5 | 638.23 | 196.11 | 3.25 | 3.57 | 2.72 | 1.31 |
| 1e-7 | 928500 | 74.88574 | 4.00e-5 | 6.33e-6 | 2509.2 | 486.54 | 5.16 | 13.70 | 6.49 | 2.11 |
| 1e-8 | 2224375 | 74.88578 | na | 2.57e-6 | 15239.6 | 1373.90 | 11.09 | 78.31 | 17.27 | 4.54 |

Table 3.4: Treecode accuracy and acceleration for the Carbon monoxide molecule using quadrature order $p = 4$ and mesh refinement parameter $tol_m$, giving mesh size $N_m$, Hartree energy $E_H$ (Ha) in Eq. (3.14) for electron density in first SCF iteration, ds error (discretization error, computed using $tol_m$ = 1e-8 as reference), tc error (treecode error, $|E_H(\mathrm{ds}) - E_H(\mathrm{tc})|$ using MAC $\theta = 0.7$ and interpolation degree $n = 8$). Run time (s) for the direct sum (ds) and treecode (tc) and treecode speedup (ds/tc) on a 6-core CPU and a single GPU.

### 3.7 Ground State Energy Computations for Atoms and Molecules

The ground-state energy of several atoms (Li, Be, O) and small molecules ($H_2$, CO, $C_6H_6$) was computed using treecode-accelerated Green Iteration (TAGI) with the LDA exchange-correlation functional [113, 114]. For each system the SCF iteration continued until the density residual fell below $tol_{scf}$ =1e-4. $tol_{gi}$ was set to 3e-3 for the first step in the SCF, then gradually reduced to 1e-5 over the next four steps. The TAGI discretization parameters (quadrature order $p$, adaptive mesh parameter $tol_m$) and treecode parameters (degree $n$, MAC $\theta$) were chosen to ensure chemical accuracy of 1 mHa/atom in the computed ground-state energy. The computations were performed on a single node, where the treecode (written in C with OpenMP+OpenACC) was run on the four GPUs and the remainder of the code (written in Python) was run in serial on one CPU core. The calculations were parallelized over the four GPUs using the shared memory scheme described in Section 2.3.1.

Table 3.5 presents the parameters and results for each system. The numerical parameters $(p, tol_m, n, \theta)$ are chosen to ensure chemical accuracy in the energy, and the heavier carbon and oxygen atoms require somewhat higher numerical resolution than the lighter hydrogen, lithium, and beryllium atoms. In particular, a larger mesh size $N_m$ requires slightly tighter treecode parameters (increasing degree from $n = 6$ to $n = 7, 8$, decreasing MAC from $\theta = 0.8$ to $\theta = 0.7, 0.6$). Column 9 records the error in the ground-state energy computed by TAGI with respect to reference energies computed to $0.1$ mHa/atom accuracy with DFT-FE [40], showing that TAGI achieves chemical accuracy. Column 10 records the total wall clock computation time (minutes). The benzene

molecule ($C_6H_6$, $N_e = 42$) is the largest system considered; the computation used approximately 1.5 million mesh points and required less than 4 hours of wall clock time.

| system | | mesh parameters | | | treecode | | results | | |
|---|---|---|---|---|---|---|---|---|---|
| formula | $N_e$ | $p$ | $tol_m$ | $N_m$ | $n$ | $\theta$ | $E_{TAGI}$ | error (Ha/atom) | time (min) |
| Li | 3 | 4 | 7e-6 | 232000 | 6 | 0.8 | -7.334051 | 4.59e-4 | 0.6 |
| Be | 4 | 3 | 1e-5 | 179712 | 6 | 0.8 | -14.44566 | 5.32e-4 | 0.5 |
| O | 8 | 4 | 3e-7 | 421000 | 6 | 0.8 | -74.46967 | -3.38e-4 | 2.4 |
| $H_2$ | 2 | 3 | 1e-3 | 51200 | 6 | 0.8 | -1.13584 | 9.05e-4 | 0.1 |
| CO | 14 | 4 | 3e-7 | 661625 | 7 | 0.7 | -112.47337 | -7.21e-4 | 15.5 |
| $C_6H_6$ | 42 | 4 | 3e-6 | 1464500 | 8 | 0.6 | -230.19316 | -3.64e-4 | 232.5 |

Table 3.5: Ground-state energy computations of atoms and small molecules using TAGI with errors computed with respect to reference values $E_{ref}$ computed using DFT-FE. TAGI discretization parameters (quadrature order $p$, adaptive mesh parameter $tol_m$, mesh size $N_m$), treecode parameters (degree $n$, MAC $\theta$); error (Ha/atom) and total wall clock computation time (minutes) on a single node with 4 GPUs.

Figure 3.10 show 2D slices of the adaptively refined mesh and computed electron density for the carbon monoxide molecule 3.10(a) and benzene molecule 3.10(b). Both molecules are located in the $z = 0$ plane; for the CO molecule, the carbon atom is at (-1.06581,0,0) and the oxygen atom is at (+1.06581,0,0); for the benzene molecule, the six carbon atoms are at $(\mp 0.682781, \pm 2.548170, 0)$, $(\pm 2.548230, \mp 0.682767, 0)$, and $(\pm 1.86544, \pm 1.86541, 0)$, and the six hydrogen atoms are at $(\mp 1.21247, \pm 4.52502, 0)$, $(\pm 4.52548, \mp 1.21333, 0)$, and $(\pm 3.31252, \pm 3.31351, 0)$. The slices are taken in the $z = 0$ plane and show the region $[-15, 15]$ a.u. in the $xy$-plane. The adaptive mesh successfully captures the variation in magnitude of the electron density.

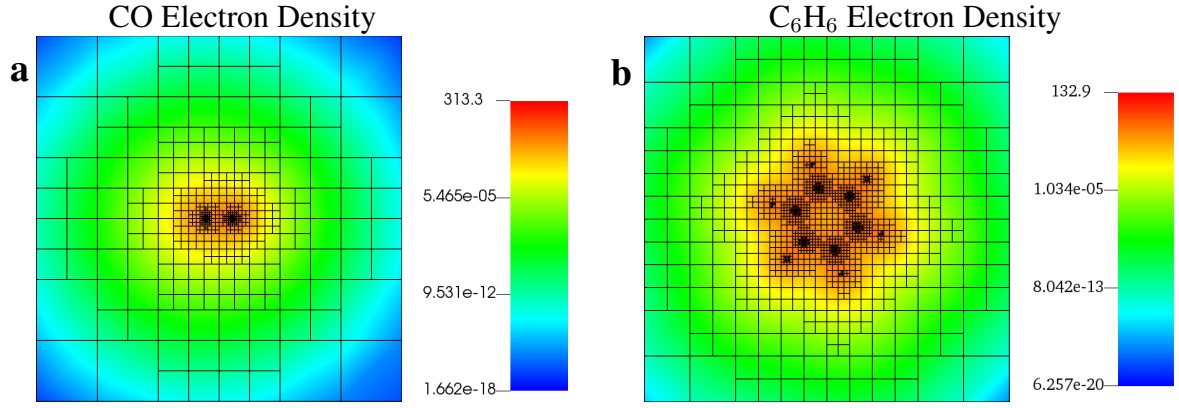


Figure 3.10: 2D slices of the adaptively refined mesh and computed electron density $\rho$ for (a) carbon monoxide molecule (CO), and (b) benzene molecule ($C_6H_6$). The slices are taken at $z = 0$ and show $[-15, 15]^2$ a.u. in $xy$-plane.

## 3.8 Conclusions

We presented a real-space method for all-electron Kohn-Sham DFT computations called Treecode-Accelerated Green Iteration (TAGI). TAGI is based on a reformulation of the Kohn-Sham equations in which the eigenvalue problem for the energies and wavefunctions $(\varepsilon_i, \psi_i)$ in differential form is recast as a fixed-point problem in integral form by convolution with the bound state Helmholtz Green's function [97]. In each SCF iteration the fixed-points are computed by Green Iteration, where the convolution integrals are discretized on an adaptive mesh and the discrete convolution sums are efficiently evaluated using a GPU-accelerated treecode.

TAGI relies on several key techniques to achieve chemical accuracy and computational efficiency. First, the Fejér quadrature rule and adaptive mesh refinement based on integration of a test function are used to compute integrals and represent the fields. Second, singularity subtraction is applied to evaluate convolution integrals having singular kernels; in particular a standard scheme is used for the Yukawa kernel in the Green Iteration convolutions, and we developed a new scheme for the Coulomb kernel in the Hartree potential convolutions since the standard scheme is inapplicable in that case. Third, a gradient-free method is employed to update the eigenvalues in Green Iteration. Fourth, the fixed-point iteration for the wavefunctions and eigenvalues in Green Iteration is accelerated using Anderson mixing. Fifth, the discrete convolution sums are computed efficiently using a barycentric Lagrange treecode (BLTC), which reduces the operation count from $O(N_m^2)$ to $O(N_m \log N_m)$, where $N_m$ is the number of mesh points. The GPU implementation of the BLTC is facilitated by the properties of barycentric Lagrange interpolation including its scale-invariance and the fact that the particle-cluster approximation in Eq. (3.66) and Eq. (3.69) consists of independent kernel evaluations which can be evaluated concurrently [52].

We demonstrated the effect of these techniques on the carbon monoxide molecule. First, we investigated the quadrature rule and adaptive mesh refinement scheme, showing that the ground-state energy of the CO molecule is computed to within chemical accuracy of 1 mHa/atom using roughly 600,000 quadrature points and 4th order quadrature. Second, we demonstrated the effect of the singularity subtraction schemes; the ground-state computation was performed with and without singularity subtraction on a sequence of progressively refined meshes, and we observed a 100-fold reduction in error using singularity subtraction. Third, we compared three methods for updating the eigenvalues in Green Iteration, 1) Laplacian update using the Rayleigh quotient of the Hamiltonian differential operator, 2) gradient update using integration by parts to reduce the order of the operator, and 3) gradient-free update; on a wide range of meshes the gradient-free update yields a 10-fold improvement in accuracy over the gradient update, and a 100-fold improvement over the Laplacian update. Fourth, we investigated the convergence of Green Iteration in the first SCF iteration for the CO molecule, showing that the spectral gap in the Hamiltonian eigenvalues controls the rate

of convergence of the eigenfunctions in Green Iteration; in particular, a small gap $|\varepsilon_i - \varepsilon_{i+1}|$ implies slow convergence of the eigenfunction $\psi_i$; the convergence rates were predicted and then verified computationally; finally we showed in the case of slow convergence, Green Iteration can be accelerated by applying Anderson mixing to the eigenpairs, yielding a 6-fold reduction in the number of iterations in this example. Fifth, we demonstrated the treecode's ability to rapidly compute accurate approximations of the discrete convolution sums; in ground-state computations for the CO molecule, we showed that the treecode approximation error can be driven below the discretization error; we then demonstrated the speedup of the treecode over direct summation on both a 6-core CPU (parallelized with OpenMP) and a GPU (parallelized with OpenACC), achieving an $11\times$ speedup on the CPU and a $4.5\times$ speedup on the GPU; finally we observed a $70\times$ speedup of the treecode running on the GPU in comparison with the CPU.

We then performed TAGI computations for several atoms and small molecules on a single node with 4 GPUs, and verified the accuracy of the ground-state energy with respect to reference values. The results demonstrate the chemical accuracy and computational efficiency afforded by TAGI on these benchmark systems.

# Chapter 4

# Extending Treecode-Accelerated Green Iteration to Pseudopotential Calculations

This chapter extends the Treecode-Accelerated Green Iteration method developed in Chapter 3 in two keys ways. First, it develops techniques for accurate and efficient pseudopotential calculations, enabling TAGI to treat systems in either the all-electron or pseudopotential framework. Second, this chapter develops the distributed memory parallelization required for TAGI to scale to multiple compute nodes. While Chapter 3 presented results limited to a single GPU node containing up to four GPUs, this chapter leverages the distributed memory MPI implementation of BaryTree to perform calculations on up to eight nodes containing thirty-two GPUs.

## 4.1 Introduction

All-electron Kohn-Sham DFT calculations require substantial resolution near the atomic nuclei in order to accurately represent the wavefunctions. In particular, due to the singular nuclear potential, the wavefunctions for the lower energy electrons vary rapidly and may have a sharp cusp at the nucleus, while the wavefunctions for the higher energy wavefunctions oscillate rapidly near the nucleus in order to maintain the orthogonality condition. In the previous chapter, we resolve these sharp cusps and rapid oscillations near the nuclei with the adaptive mesh refinement scheme. This scheme achieves the desired accuracy, but can result in many quadrature points per atom and expensive calculations. In this chapter we introduce an alternative approach to Kohn-Sham DFT calculations that mitigates these effects through the use of non-singular pseudopotentials, and present the techniques developed for TAGI that enable accurate and efficient pseudopotential calculations. We then present various techniques implemented in TAGI related to high-performance computing, in particular techniques that enable distributed memory MPI parallelization and that allow for additional GPU acceleration.

The method of pseudopotentials relies on the observation that within a molecule, an atom's core electrons are chemically inert and remain localized to its nucleus, while its valence electrons participate in bonding. There are a variety of pseudopotentials, each respecting a different set of

constraints [118, 119, 120, 121], and in particular this work extends TAGI to Optimized Norm-Conserving Vanderbilt (ONCV) pseudopotentials [121, 122, 123]. Pseudopotentials reduce the computational cost of the calculations in two ways. First, the electrons are partitioned into core and valence electrons; the core electrons are absorbed into the nucleus and contribute to the pseudopotential, leaving only the valence states to be computed. Second, the pseudopotentials are non-singular and are significantly smoother than the all-electron nuclear potentials. By implicitly accounting for the core electrons and replacing the singular potential, the resulting valence wavefunctions no longer oscillate rapidly near the nuclei and can be resolved on significantly coarser meshes.

This chapter presents the extension of Treecode-Accelerated Green Iteration to pseudopotential calculations and demonstrates the method on several larger systems than were demonstrated for the all-electron case. Section 4.2 presents the ONCV pseudopotential formulation and how it differs from all-electron formulations in Green Iteration. Section 4.3 describes the real-space discretization techniques used to achieve accurate pseudopotential calculations on significantly coarser meshes than all-electron calculations. Section 4.4 presents additional high performance computing optimizations such as GPU porting of several routines and a distributed memory MPI parallelization. Section 4.5 presents pseudopotential TAGI results for two silicon quantum dots and two carbon fullerenes. Section 4.6 provides a summary of our pseudopotential TAGI findings and ideas for future developments to improve its performance for larger systems.

## 4.2 Pseudopotential Formulation

### 4.2.1 Kohn-Sham Hamiltonian Construction

The structure of Kohn-Sham DFT calculations is independent of whether it is an all-electron or pseudopotential calculation; both use the self-consistent field iteration to solve the nonlinear eigenvalue problem for the Kohn-Sham operator,

$$\mathcal{H}[\rho]\psi_i(\mathbf{r}) = \varepsilon_i\psi_i(\mathbf{r}), \quad i = 1, 2, \ldots, \quad \mathcal{H}[\rho] = -\frac{1}{2}\nabla^2 + V_{eff}[\rho], \tag{4.1}$$

where the electron density $\rho(\mathbf{r})$ is defined in Eq. (3.4). The difference between all-electron and pseudopotential calculations is the construction of the Kohn-Sham effective potential,

$$V_{eff}[\rho](\mathbf{r}) = V_H[\rho](\mathbf{r}) + V_{ext}(\mathbf{r}) + V_{xc}[\rho](\mathbf{r}), \tag{4.2}$$

specifically the external potential $V_{ext}(\mathbf{r})$ due to the atomic nuclei. The all-electron external potential is given by

$$V_{ext}(\mathbf{r}) = \sum_{J=1}^{N_A} \frac{-Z_J}{|\mathbf{r} - \mathbf{R}_J|}, \tag{4.3}$$

that is, the Coulomb potential due to each of the nuclei of charge $Z_J$ located at $\mathbf{R}_J$. For ONCV pseudopotentials, the external potential is composed of a local term $V^J_{loc}(\mathbf{r})$ and a nonlocal term $V^J_{nonloc}$, defined by its action on a wavefunction $\psi(\mathbf{r})$,

$$V^J_{nonloc}\psi(\mathbf{r}) := \int V^J_{nonloc}(\mathbf{r},\mathbf{r}',\mathbf{R})\psi(\mathbf{r}')d\mathbf{r}' = \sum_{\ell pm} C^J_{\ell pm} h^J_{\ell p} \chi^J_{\ell pm}(\mathbf{r},\mathbf{R}_J), \tag{4.4}$$

where $\chi^J_{\ell pm}(\mathbf{r},\mathbf{R}_J)$ is the pseudopotential projector corresponding to quantum numbers $(\ell, m)$ for atom $J$ located at $\mathbf{R}_J$, $p$ denotes the projector index, and $C^J_{\ell pm}$ and $h^J_{\ell p}$ are defined by

$$C^J_{\ell pm} = \int \chi^J_{\ell pm}(\mathbf{r},\mathbf{R}_J)\psi(\mathbf{r})d\mathbf{r}, \qquad \frac{1}{h^J_{\ell p}} = \langle \xi^J_{\ell m} | \chi^J_{\ell pm} \rangle, \tag{4.5}$$

where $\xi^J_{\ell m}$ denotes the single atom wavefunction. For the ONCV pseudopotentials, $p = 1, 2$. Then, the action of the external potential is given by

$$V^{PS}_{ext}\psi(\mathbf{r}) = \sum_{J=1}^{N_A} \left( V^J_{loc}(\mathbf{r})\psi(\mathbf{r}) + V^J_{nonloc}\psi(\mathbf{r}) \right). \tag{4.6}$$

Core electrons are absorbed into the nucleus, which is left with net charge $Z^J_{valence}$. The local potential $V^J_{loc}$ is a regularized potential corresponding to the nucleus and core electrons; it asymptotically converges to $\frac{-Z^J_{valence}}{|\mathbf{r}-\mathbf{R}_J|}$ in the far field.

### 4.2.2 Green Iteration

The method of Green's functions for converting the eigenvalue problem for the Kohn-Sham differential operator into a fixed-point problem for the corresponding integral operator is identical to the all-electron case and results in the integral equations,

$$\psi_i(\mathbf{r}) = \mathcal{G}(\varepsilon_i)\psi_i(\mathbf{r}), \quad i = 1, \ldots, N_w, \quad \psi_i \perp \psi_j,\ j < i, \tag{4.7}$$

where

$$\mathcal{G}(\varepsilon)\psi(\mathbf{r}) = \int G_\varepsilon(\mathbf{r},\mathbf{r}')V_{eff}[\rho](\mathbf{r}')\psi(\mathbf{r}')d\mathbf{r}', \tag{4.8}$$

and

$$G_\varepsilon(\mathbf{r},\mathbf{r}') = -\frac{e^{-\sqrt{-2\varepsilon}|\mathbf{r}-\mathbf{r}'|}}{2\pi|\mathbf{r}-\mathbf{r}'|}. \tag{4.9}$$

In the case of pseudopotentials, however, the term $V_{eff}[\rho](\mathbf{r}')\psi(\mathbf{r}')$ is split into the local and nonlocal pieces as described above. The construction of the nonlocal potential in Eq. (4.4) depends on the current estimate of the wavefunction, therefore this nonlocal construction occurs in every step in

Green Iteration. Aside from the construction of the potential, the remainder of Green Iteration remains the same for pseudopotential and all-electron calculations.

## 4.3 Numerical Implementation

### 4.3.1 Pseudopotential Spline Interpolation

In this work we use Optimized Norm-Conserving Vanderbilt pseudopotentials generated with the code ONCVPSP [122, 123]. This database provides the constants $h^J_{\ell p}$ as well as discrete radial data at $\{r_i\} = r_0, r_1, \ldots, r_{cutoff}$ for the following fields: the single atom electron density $\rho^J(r)$, the local external potential $V^J_{loc}(r)$, and the projectors $\chi^J_{\ell p}(r)$. In the procedures described below, these fields are evaluated at the quadrature points of the real-space mesh. This requires an interpolation of the radial data as well as extrapolation for quadrature points outside the cutoff radius $r_{cutoff}$ of the discrete data. The 3-dimensional projectors $\chi^J_{\ell pm}(\mathbf{r})$ are obtained by evaluating the radial interpolant of $\chi^J_{\ell p}(r)$, and then multiplying by the appropriate spherical harmonic. For the interpolation we use cubic splines and enforce the following specified boundary conditions.

**Density**. For the density $\rho^J(r)$ we use natural boundary conditions, i.e. zero second derivative, at both $r = 0$ and $r = r_{cutoff}$. To extrapolate beyond $r_{cutoff}$, we fit a decaying exponential to the final discrete intervals. As the single-atom density is only used to provide an initial guess for the SCF iteration, the convergence and accuracy of the calculation does not depend sensitively on the density spline.

**Local external potential**. For the local external potential $V^J_{loc}(r)$, we enforce first derivative boundary conditions. At the $r = 0$ boundary, we set the derivative $slope_L$ to be equal to the slope between the first and second discrete points,

$$slope_L = \frac{V^J_{loc}(r_1) - V^J_{loc}(r_0)}{r_1 - r_0}, \tag{4.10}$$

in order to avoid introducing any spurious numerical oscillations near the nucleus that would affect accuracy. At the $r = r_{cutoff}$ boundary, we use the fact that $V^J_{loc}(r)$ asymptotes to the nuclear Coulomb potential $-Z^J_{valence}/r$ in the far field. We set the derivative $slope_R$ at the $r = r_{cutoff}$ boundary to $-V^J_{loc}(r_{cutoff})/r_{cutoff}$,

$$slope_R = \frac{d}{dr}\frac{-Z_{valence}}{r_{cutoff}} = \frac{Z_{valence}}{r^2_{cutoff}} = -\frac{V^J_{loc}(r_{cutoff})}{r_{cutoff}}. \tag{4.11}$$

To extrapolate to quadrature points beyond $r_{cutoff}$ we simply evaluate $-Z_{valence}/r$. Figure 4.1 shows the local potential spline for the silicon atom; Fig. 4.1(a) shows the spline and how it differs from $-Z_{valence}/r$ near the nucleus but approaches it at the cutoff radius, and Fig. 4.1(b) shows the

difference $|-Z_{valence}/r - V_{loc}(r)|$, which decays towards zero as $r \to r_{cutoff}$.

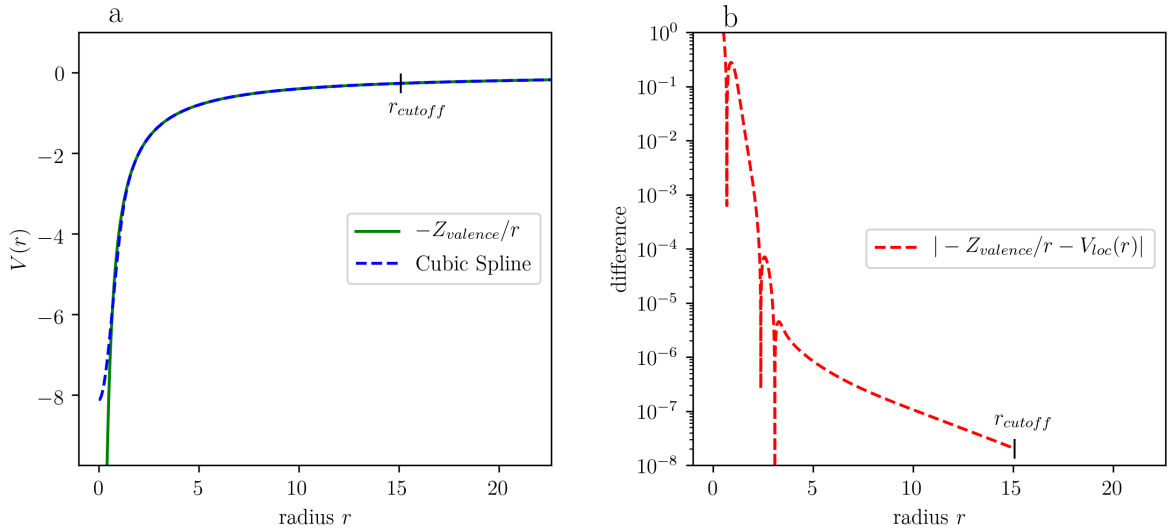


Figure 4.1: Silicon atom, local external potential spline interpolation and asymptotic extrapolation, (a) the cubic spline interpolant and the asymptotic potential $-Z_{valence}/r$, and (b) the difference between $V_{loc}(r)$ and $-Z_{valence}/r$, which decays as $r \to r_{cutoff}$.

**Projectors**. For the projectors we use natural boundary conditions. ONCVPSP sets $r_{cutoff}$ large enough that the projectors have decayed sufficiently close to zero by the boundary, hence we extrapolate with zeros.

### 4.3.2 Mesh Generation

**Motivation for two-mesh scheme**. Pseudopotentials reduce computational costs in two ways. First, they reduce the number of electrons per atom, thereby reducing the number of wavefunctions that must be computed. Second, the resulting wavefunctions are not highly oscillatory near the nucleus, as is the case with the all-electron potential, which alleviates the need for highly refined meshes. Figure 4.2(a) shows the radial component of the all-electron wavefunctions $\psi^{AE}_{n\ell}(r)$ for a silicon atom, and Fig. 4.2(b) compares the all-electron wavefunctions to the pseudopotential valence wavefunctions $\psi^{PSP}_{n\ell}(r)$. The pseudopotential wavefunctions do not oscillate near the nucleus like the all-electron wavefunctions, but converge to the all-electron wavefunctions away from the nucleus. Due to having both fewer and smaller oscillations than the all-electron wavefunctions, the pseudopotential wavefunctions can be accurately represented on significantly coarser meshes.

Figure 4.2 suggests TAGI can use a relatively coarse mesh for pseudopotential calculations and still achieve chemical accuracy. While it is true that the pseudopotential wavefunctions can be represented with coarse meshes, the limiting factor of accuracy is not the resolution of the wavefunctions; instead, it is the resolution of the ONCV pseudopotential projectors from Eq. (4.4).

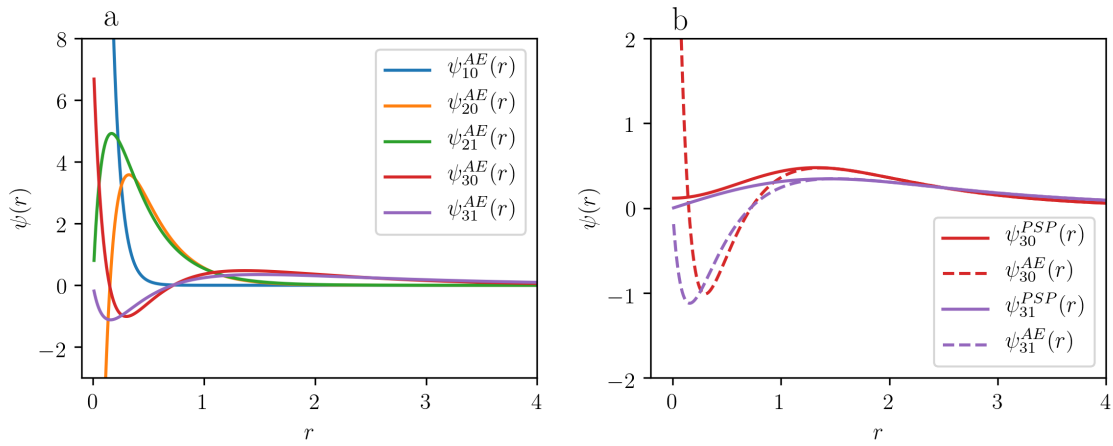


Figure 4.2: Silicon atom, radial components of the (a) all-electron wavefunctions, (b) valence wavefunctions for ONCV pseudopotential and all-electron. The pseudopotential absorbs $\psi_{10}(r)$, $\psi_{20}(r)$, and $\psi_{21}(r)$ into the nucleus and eliminates the oscillation near the nucleus from $\psi_{30}(r)$ and $\psi_{31}(r)$. The pseudopotential wavefunctions converge to the all-electron wavefunctions away from the nucleus.

Figure 4.3 shows the radial components of (a) the wavefunctions and the (b) projectors for the silicon atom, demonstrating the relative smoothness of the wavefunctions compared to the projectors. The projectors oscillate more rapidly than the wavefunctions and require more refinement than the wavefunctions. To address this we employ a two-mesh scheme that represents the wavefunctions on a coarse mesh and the projectors on a fine mesh. Next we describe how each mesh is generated as well as the interpolation between them. Then, below, we present the two-mesh solution procedure that achieves both accurate and efficient calculations.

**Coarse Mesh Generation**. As shown in Fig. 4.2, the relative smoothness of the pseudopotential wavefunctions suggests that they can be represented accurately on significantly coarser meshes than the all-electron wavefunctions. In this work we generate coarse meshes based on three parameters: (1) a ball radius $r_{coarse}$, (2) an inner mesh spacing inside the ball $\Delta h_{inner}$, and (3) a far-field mesh size $\Delta h_{outer}$. Except where specified otherwise, when cells are refined they are bisected in each of the three dimensions, resulting in eight children cells of equal size. First, the domain is refined uniformly so that no cells exceed the far-field mesh size parameter. In particular, none of a cell's three side lengths may exceed the size parameter. Second, any cells lying within the ball radius of an atom are refined until no cells in the ball exceed the inner mesh spacing parameter. Third, the cells outside the ball radius $r_{coarse}$ are refined so that they coarsen into the far-field mesh, i.e. cells immediately beyond $r_{coarse}$ are refined until no larger than $2\Delta h_{inner}$, the next layer of cells are refined until no larger than $4\Delta h_{inner}$, and so on until the far-field spacing $\Delta h_{outer}$ is reached. Figure 4.4 shows an example of the all-electron and pseudopotential meshing schemes for a silicon

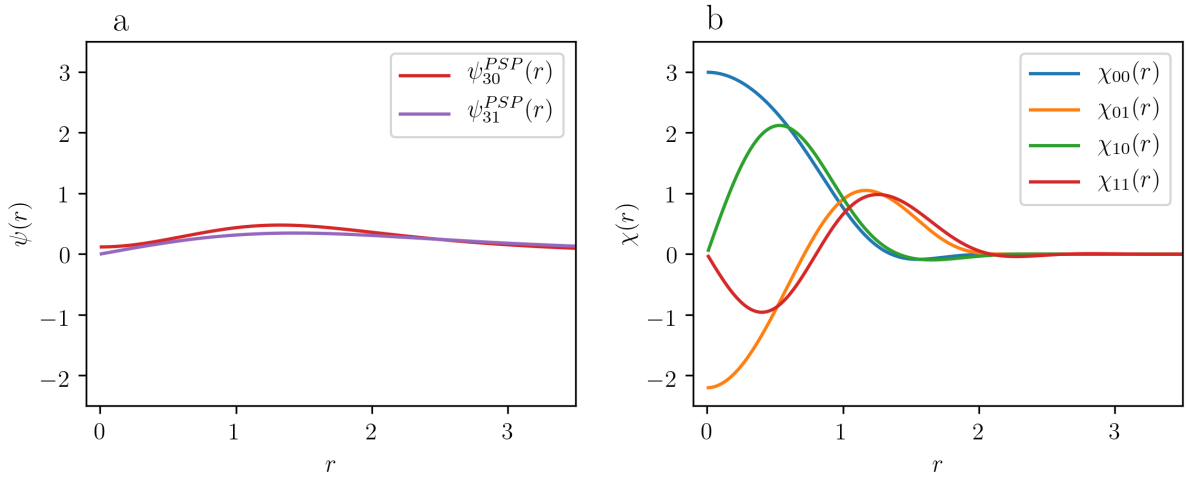


Figure 4.3: Silicon atom, radial components (suppressed magnetic quantum number $m$) of (a) the single-atom wavefunctions $\psi_{n\ell}(r)$, and (b) the ONCV projectors $\chi_{\ell p}(r)$. While the wavefunctions are accurately represented on a coarse mesh, the projectors are not due to the frequency and magnitude of their oscillation.

atom at the origin. The all-electron mesh is constructed with $tol_m$ =1e-6, while the pseudopotential mesh is constructed with ball radius $r_{coarse} = 2.0$, far-field mesh spacing $\Delta h_{outer} = 8.0$, and inner mesh spacing $\Delta h_{inner} = 0.5$. The all-electron mesh contains many levels of local refinement near the atom, as required to capture the peaks and oscillations of the all-electron wavefunctions shown in Fig. 4.2, whereas the pseudopotential mesh merely refines to cells of size 0.5.

**Fine Mesh Generation**. We construct the fine mesh as a further refinement of the coarse mesh described above. This allows for fields to be transferred between the two meshes with local cell-wise interpolation, as described below. We construct the fine mesh from the coarse mesh by prescribing two additional parameters: (1) an additional refinement ball radius $r_{fine}$, and (2) a refinement ratio $d_{cf}$ between the coarse and fine mesh spacings. Any cell belonging to the coarse mesh that lies within the refinement ball radius $r_{fine}$ is further refined to generate the fine mesh. A ratio $d_{cf} = 2$ causes each cell to be bisected in each dimension and results in 8 children, a ratio $d_{cf} = 3$ results in 27 children, and so on. Figure 4.5 shows a slice of the refined mesh using a refinement ball radius $r_{fine} = 3$ and a ratio $d_{cf} = 2$. The black lines correspond to the original coarse mesh, and the red lines indicate the subsequent refinement. The additional refinement near the nucleus improves the resolution of the pseudopotential projectors.

**Mesh-to-Mesh Interpolation**. By construction, the cells in the refined mesh are nested in the cells of the coarse mesh; if the coarse cell lies within the refinement ball radius then it was subdivided into a set of children cells according to the refinement ratio. During the two-mesh solution scheme described below, some fields that are represented on the coarse mesh must be interpolated to the fine mesh. This interpolation is done cell-wise; the field represented on the coarse

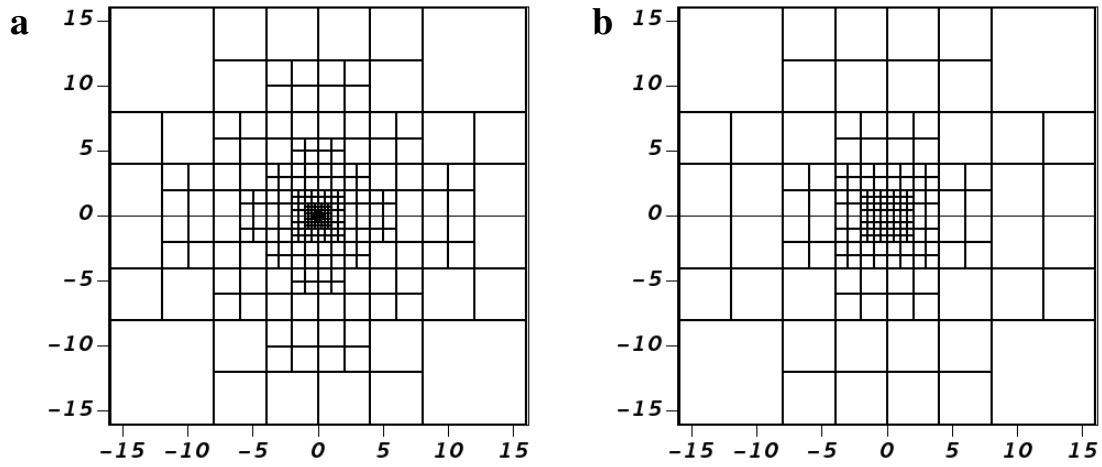


Figure 4.4: Slice of the mesh for a Silicon atom at the origin, (a) all-electron mesh constructed with quadrature order 4 and $tol_m = 1e-6$ (1,016,000 points), and (b) pseudopotential mesh constructed with far-field mesh spacing $\Delta h_{outer} = 8.0$, ball radius $r_{coarse} = 2.0$, and inner mesh spacing $\Delta h_{inner} = 0.5$ (169,000 points). The wavefunctions that must be resolved by each mesh are shown in Fig. 4.2.

mesh cell must be interpolated to the cell's children in the fine mesh. By construction, the fields are represented on Chebyshev points of the first kind for the Fejér quadrature rule as described in Chapter 3. In addition to quadrature, these points serve as interpolation points for the mesh-to-mesh interpolation.

The Chebyshev points of the first kind on the interval $[-1, 1]$ are given by

$$t_k = \cos\theta_k, \quad \theta_k = \frac{(j+1/2)\pi}{p+1}, \quad j = 0:p. \tag{4.12}$$

The 1-dimensional barycentric form of the Lagrange interpolating polynomial of degree $p$ is

$$L_k(x) = \frac{\dfrac{w_k}{x-t_k}}{\displaystyle\sum_{k'=0}^{p} \frac{w_{k'}}{x-t_{k'}}}, \quad w_k = \frac{1}{\prod_{j=0,j\neq k}^{n}(t_k - t_j)}. \tag{4.13}$$

where the interpolation weights for the Chebyshev points of the first kind are given by

$$w_k = (-1)^k \sin\left(\frac{(2k+1)\pi}{2p+2}\right), \tag{4.14}$$

and removable singularity in the barycentric form is discussed in Section 2.2.3.

Consider interpolating from the coarse mesh cell to its children cells. Denote the quadrature

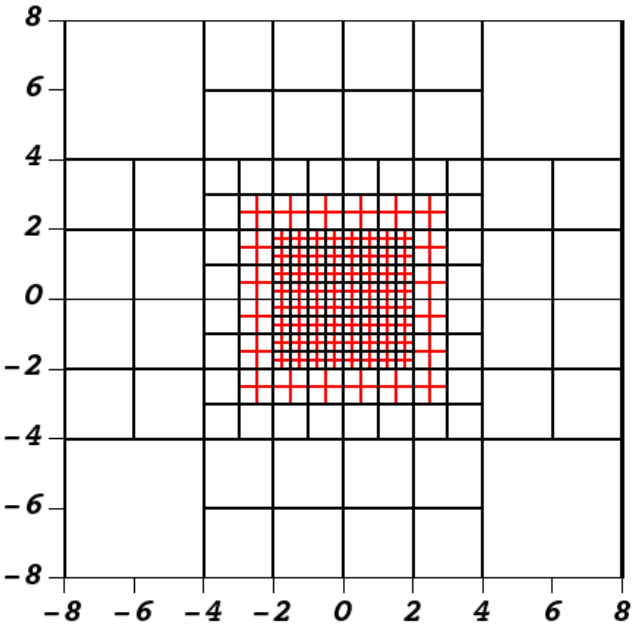


Figure 4.5: Slice of the two-mesh scheme for $x, y \in [8, 8]$ and $z = 0$. The black lines indicate the coarse mesh, which was constructed with far-field mesh spacing $\Delta h_{outer} = 8.0$, ball radius $r_{coarse} = 2.0$, and inner mesh spacing $\Delta h_{inner} = 0.5$. The red lines indicate the additional refinement for the projectors using a refinement ball radius $r_{fine} = 3.0$ and refinement ratio $d_{cf} = 2$. The additional refinement is required to resolve the projectors shown in Fig. 4.3.

points in the coarse cell by $\mathbf{r} = (x, y, z) = (t_{k_1}, t_{k_2}, t_{k_3})$, where the second equality comes from the fact that the quadrature points are Chebyshev points. Similarly, denote the quadrature points in the fine mesh by $\mathbf{r}' = (x', y', z')$, noting that the prime indicates fine mesh points. Interpolating the wavefunction $\psi(\mathbf{r}) = \psi(x, y, z)$ from the coarse mesh to $\psi(\mathbf{r}') = \psi(x', y', z')$ in its refined child cell requires evaluating

$$\psi(x', y', z') \approx \sum_{(k_1,k_2,k_3)} L_{k_1}(x')L_{k_2}(y')L_{k_3}(z')\psi(t_{k_1}, t_{k_2}, t_{k_3}) \tag{4.15}$$

at each point $(x', y', z')$ in the refined cell, where the sum runs over each point $(x, y, z) = (t_{k_1}, t_{k_2}, t_{k_3})$ in the coarse cell. The other fields such as the electron density and potential are similarly interpolated from coarse mesh cells to their fine mesh child cells.

### 4.3.3 Two-Mesh Solution Scheme

We employ a two-mesh scheme that leverages the efficiency of representing the wavefunctions on the coarse mesh and the accuracy of resolving the projectors and local nuclear potential on the

fine mesh. This scheme relies on several key properties:

1. The fields that must be computed are relatively smooth (the wavefunctions and density).
2. The projectors, which require finer resolution, are available *a-priori* as radial data.
3. The dominant cost in the particle-cluster treecode is $O(N \log M)$, where $N$ is the number of target points and $M$ is the number of source points. The scheme described below uses the coarse mesh for the $N$ target points and the fine mesh for the $M$ source points.
4. High accuracy is not required until the end of the calculation, when the density and energies are converged.

The strategy is to construct a coarse mesh suitable for the wavefunctions and a fine mesh suitable for the projectors and local nuclear potential using the generation procedures described above. The wavefunctions are then computed directly on the coarse mesh, and when high accuracy is required, the potential is constructed on the fine mesh. Next we describe how the two-mesh calculation is performed, and below we describe when the two-mesh scheme is used.

Recall that each step in Green Iteration involves a convolution of the form

$$\psi^{(n+1)}(\mathbf{r}) = \int G_\varepsilon(\mathbf{r}, \mathbf{r}') V_{eff}[\rho](\mathbf{r}') \psi^{(n)}(\mathbf{r}') d\mathbf{r}'. \tag{4.16}$$

When using the two-mesh scheme we differentiate between the target mesh and source mesh in the convolution; the set of points $\{\mathbf{r}\}$ correspond to the coarse mesh of $N$ target points and the set of points $\{\mathbf{r}'\}$ correspond to the fine mesh of $M$ source points. In order to compute $\psi^{(n+1)}(\mathbf{r})$, we must obtain $\psi^{(n)}(\mathbf{r}')$ and $V_{eff}[\rho](\mathbf{r}')$ on the fine mesh. For clarity we will separate $V_{eff}[\rho](\mathbf{r}')\psi^{(n)}(\mathbf{r}')$ into its four individual terms,

$$V_{eff}[\rho](\mathbf{r})\psi^{(n)}(\mathbf{r}) = V_H[\rho](\mathbf{r})\psi^{(n)}(\mathbf{r}) + V_{xc}[\rho](\mathbf{r})\psi^{(n)}(\mathbf{r}) + \left(\sum_{J=1}^{N_A} V_{loc}^J(\mathbf{r})\right)\psi^{(n)}(\mathbf{r}) + V_{nonloc}\psi^{(n)}(\mathbf{r}), \tag{4.17}$$

and describe how each is obtained on the fine mesh. The first term is the product of the wavefunction and the Hartree potential which depends on the density. $\psi^{(n)}(\mathbf{r})$ was computed on the coarse mesh during the previous step of Green Iteration, and $V_H[\rho](\mathbf{r})$ was computed on the coarse mesh during the previous SCF iteration. Both of these fields are interpolated onto the fine mesh with the mesh-to-mesh interpolation scheme that is described above. Similarly, the second term is the product of the wavefunction and the exchange-correlation potential, both of which were computed on the coarse mesh and are interpolated onto the fine mesh. The third term is the product of the local nuclear potential and the wavefunction; again the wavefunction is interpolated from the coarse mesh

onto the fine mesh, while the local nuclear potential is available *a-priori* as radial data and can be interpolated directly onto the fine mesh as described above in Section 4.3.1. The fourth term is the nonlocal term of the pseudopotential, given in Eq. (4.4). The wavefunction is interpolated onto the fine mesh. The projectors $\chi^J_{\ell pm}(\mathbf{r}, \mathbf{R}_J)$ are available *a-priori* and evaluated directly on the fine mesh quadrature points. The coefficients $C^J_{\ell pm}$ are computed on the fine mesh, and the subsequent construction $\sum_{J=1}^{N_A} \sum_{\ell pm} C^J_{\ell pm} h^J_{\ell p} \chi^J_{\ell pm}(\mathbf{r}, \mathbf{R}_J)$ is performed on the fine mesh.

Once each of the four terms in the right hand side of Eq. (4.17) is represented on the fine mesh, the next wavefunction $\psi^{(n+1)}(\mathbf{r})$ can be computed on the coarse mesh. We recall key property 3 from above and note that the scaling of the treecode plays an important role in the two-mesh scheme. In particular, we use a particle-cluster treecode where the dominant cost of the computation scales like $O(N \log M)$, where $N$ is the number of target points and $M$ is the number of source points. Crucially, the coarse mesh is used for the targets and the fine mesh is used for the sources. Therefore, $N$ remains relatively small and the cost of the convolution only grows logarithmically with the size of the fine mesh $M$ and is substantially cheaper than evaluating the convolutions directly on the fine mesh.

This two-mesh convolution is significantly more accurate than the coarse-mesh convolution due to the additional resolution of the projectors, while being only moderately more expensive due to the logarithmic scaling of the particle-cluster treecode with respect to the source mesh. As a result, the calculation achieves accuracy commensurate with fine mesh convolutions in time commensurate with coarse mesh convolutions.

**Two-mesh energy correction**. We recall key property 4 from above, that high accuracy is not required throughout the entire calculation but rather only at the end once the density has converged. Previously, we took advantage of this property by using an adaptive convergence tolerance for Green Iteration; we gradually tighten the Green Iteration tolerance as the SCF iteration progresses, reducing the cost of the iterations when the density is far from converged. Now, we take advantage of this property again when applying the two-mesh scheme. We find that rather than using the two-mesh scheme throughout the calculation, we can use it only at the end of the calculation to compute corrections to the eigenvalues and the energy. This approach still achieves the improved accuracy due to representing the projectors on the fine mesh while allowing a majority of convolutions to be performed exclusively on the coarse mesh.

Algorithm 4.1 describes the two-mesh correction scheme which takes the converged coarse mesh density and eigenpairs as input and computes two-mesh corrected energies. This is an efficient approach in which the fine mesh is only used at the very end of the calculation after the SCF converges, as compared to less greedy approaches where the fine mesh is used throughout some or all of the SCF iterations. Lines 1-3 interpolate the density to the fine mesh $\{\mathbf{r}'\}$, and then compute a

corrected Hartree potential on the coarse mesh $\{\mathbf{r}\}$ and subsequent Hartree energy,

$$V_H^{corr}[\rho](\mathbf{r}) = \int \frac{\rho(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|} d\mathbf{r}', \quad E_H^{corr} = \frac{1}{2}\int V_H^{corr}[\rho](\mathbf{r})\rho(\mathbf{r})d\mathbf{r}. \tag{4.18}$$

Line 4 interpolates the corrected Hartree potential and the exchange-correlation potential onto the fine mesh. Line 5 evaluates the local part of the external potential directly on the fine mesh, as this is available *a-priori* for each atomic species as radial data. Line 6 loops over each of the eigenpairs in order to compute a two-mesh corrected eigenvalue. Line 7 interpolates the wavefunction onto the fine mesh, then line 8 uses the interpolated wavefunction and the projectors evaluated directly on the fine mesh to compute the nonlocal potential on the fine mesh. Following line 8 all pieces of the potential are either computed on or interpolated onto the fine mesh. Line 9 performs one iteration of the fixed-point equation, resulting in a corrected wavefunction $\psi_i^{corr}(\mathbf{r})$. Line 10 computes corrected eigenvalues using the corrected wavefunction in the gradient-free eigenvalue update from Eq. (3.52), repeated here in terms of the corrected wavefunction $\psi_i^{corr}$,

$$\varepsilon_i^{corr} = \varepsilon_i - \frac{\langle V_{eff}\psi_i, \psi_i - \psi_i^{corr}\rangle}{\langle \psi_i^{corr}, \psi_i^{corr}\rangle}. \tag{4.19}$$

Line 11 computes the corrected band energy from the corrected eigenvalues,

$$E_{band}^{corr} = 2\sum_{i=1}^{N_w} f_i(\varepsilon_i^{corr}, \mu)\varepsilon_i^{corr}, \tag{4.20}$$

and finally line 12 computes the corrected total energy from the corrected band and Hartree energies,

$$E = E_{band}^{corr} - E_H^{corr} + E_{xc} - \int \rho(\mathbf{r})V_{xc}[\rho](\mathbf{r})d\mathbf{r} + E_{ZZ}. \tag{4.21}$$

Importantly, this greedy final energy correction scheme achieves the same level of accuracy as the less greedy approaches of using the two-mesh scheme during the SCF iterations, with energy differences on the order of $O(10)$ $\mu$Ha/atom for the examples demonstrated in this work, well below the target accuracy.

### 4.4 High Performance Computing Optimizations

This section describes our efforts to further improve the performance of TAGI on high performance computing architectures. First, in addition to the BaryTree convolutions, we port two more computations onto GPUs, the mesh-to-mesh interpolation and the wavefunction orthogonalization. Second, we implement a distributed memory MPI parallelization of TAGI that involves domain

**Algorithm 4.1** Two-Mesh Energy Correction

**input**: converged coarse mesh density $\rho(\mathbf{r})$ and eigenpairs $(\psi_i(\mathbf{r}), \varepsilon_i)$ for $i = 1, \ldots, N_w$
**output**: two-mesh corrected eigenvalues and total energy

interpolate the density $\rho(\mathbf{r})$ from the coarse mesh $\mathbf{r}$ to the fine mesh $\mathbf{r}'$
compute the corrected Hartree potential $V_H^{corr}[\rho(\mathbf{r}')](\mathbf{r})$ in Eq. (4.18) from the fine-mesh density
compute the corrected Hartree energy in Eq. (4.18) from $V_H^{corr}[\rho(\mathbf{r}')](\mathbf{r})$
interpolate $V_H^{corr}[\rho(\mathbf{r}')](\mathbf{r})$ and $V_{xc}(\mathbf{r})$ from the coarse mesh to the fine mesh
evaluate the local external potential $\sum_{J=1}^{N_A} V_{loc}^J(\mathbf{r})$ directly on the fine mesh
for $i = 1, 2, \ldots, N_w$
    interpolate the coarse mesh wavefunction $\psi_i$ onto the fine mesh
    compute the nonlocal potential with the fine mesh projectors $\chi_{\ell pm}^J$ and interpolated wavefunction, completing the construction of $V_{eff}(\mathbf{r}')$ on the fine mesh
    compute the corrected wavefunction $\psi_i^{corr}(\mathbf{r}) = \int G_\varepsilon(\mathbf{r}, \mathbf{r}') V_{eff}(\mathbf{r}') \psi_i(\mathbf{r}') d\mathbf{r}'$
    compute corrected eigenvalue $\varepsilon_i^{corr}$ with the gradient-free update in Eq. (4.19)
correct the band energy with the corrected eigenvalues
correct the total energy with the corrected band energy and Hartree energy

decomposition, parallel mesh refinement, and load balancing. These developments enable TAGI to make use of the distributed memory implementation of BaryTree and scale to multiple compute nodes.

### 4.4.1 GPU porting

The discrete convolutions dominate the cost of the calculation in TAGI, therefore these were the first calculation to be accelerated on GPUs which is discussed in Chapters 2 and 3. However, for larger systems, there are other calculations occurring on the CPU that begin to take a non-negligible amount of time. In particular, for large meshes the mesh-to-mesh interpolation becomes expensive during the two-mesh correction, and for systems with many wavefunctions the orthogonalization during each Green Iteration becomes expensive. We port both of these calculations to GPUs using OpenACC to increase the efficiency of TAGI.

**Mesh-to-mesh Interpolation**. The mesh-to-mesh interpolation is performed on GPUs. Each GPU kernel launch is responsible for interpolating from one parent cell to each of its child cells. For refinement ratio $d_{cf} = 2$ and quadrature order $p$, this amounts to interpolating fields from $(p+1)^3$ coarse mesh points to $8(p+1)^3$ fine mesh points. The parent-to-children interpolation is structured as nested loops; the outer loop runs over the $8(p+1)^3$ points in the fine children cells, while the inner loop runs over the $(p+1)^3$ points in the coarse parent cell. The outer loop is naturally parallelizable; the interpolation at each fine mesh point $\mathbf{r}'$ is independent and can be computed simultaneously. Therefore, the outer loop is parallelized over the thread-blocks, one block per $(x', y', z')$. The inner loop is parallelizable with a reduction; the interpolation at a given

point $\mathbf{r}'$ is the sum over the contributions from each coarse mesh point $\mathbf{r}$. These contributions can be computed in parallel, followed by a reduction. Therefore, we parallelize the inner loop with the threads, one thread per $(x, y, z)$. Each thread computes one term in the sum in Eq. (4.15), followed by a reduction over the thread-block.

**Orthogonalization**. In Green Iteration, after the $j$th wavefunction $\psi_j$ is updated with a convolution, it must be orthogonalized against all previous wavefunctions $\psi_i$, $i < j$. The cost of this orthgonalization on the CPU begins to become non-negligible compared to the GPU convolution for systems containing $\sim O(100)$ electrons. We reduce the cost of the orthogonalization considerably by porting it to also run on the GPUs. The Gram-Schmidt calculation itself maps to GPUs in a straightforward way; the dot products between $\psi_j$ and each $\psi_i$ are independent and are assigned to different thread-blocks, then each individual dot product is computed in parallel by the threads using a reduction. However, an efficient GPU implementation requires careful treatment of the memory. In particular, it is not efficient to copy the set of wavefunctions onto the GPU for each call to the orthogonalization routine; instead the wavefunctions must be copied to the GPU once and allowed to persist in the GPU memory throughout the calculation. We accomplish this using OpenACC *unstructured data regions*; the wavefunctions are copied onto the GPU at the beginning of the calculation, then updated in-place with the convolutions, then subsequently used in the orthogonalization routines. Following orthogonalization, the $j$th wavefunction (and only the $j$th wavefunction) is copied back to the CPU where it is needed for various other calculations that occur on the CPU, such as the gradient-free eigenvalue update and the Anderson mixing. In future work we will consider eliminating the need to copy back to the CPU by also performing these calculations on the GPU, even though they are not performance bottlenecks. This procedure reduces the cost of the orthogonalization to a negligible amount for the system sizes investigated in this work.

### 4.4.2 Distributed Memory MPI Parallelization

Recall that the first implementation of TAGI, described in Chapter 3, used shared memory parallelization with OpenMP and was restricted to a single compute node. To further reduce the computation time and make full use of BaryTree we parallelize TAGI with MPI and the mpi4py python module. The dominant cost of TAGI calculations is the discrete convolutions, hence taking advantage of the parallel efficiency demonstrated in Section 2.4 will significantly improve the performance of TAGI.

**Parallel Mesh Refinement, Domain Decomposition, and Load Balancing**. To achieve high parallel efficiency in the treecode we require each MPI rank to own a localized region of the domain, ensuring each rank's Locally Essential Tree (LET) is minimal, while also maintaining a balanced number of quadrature points in each partition. The construction of a localized and balanced decomposition is subject to the following constraints. First, the mesh should be generated

in parallel, both to achieve efficiency and to avoid memory limitations of the single rank. Second, load balancing must occur after the adaptive refinement is complete. The density of quadrature points varies considerably throughout the domain, hence partitioning the domain prior to mesh generation leads to significant load imbalances. Third, the decomposition of the coarse and fine meshes must coincide so that the local cell-wise interpolation between the meshes is possible without the need for communication. Algorithm 4.2 presents our approach to the mesh generation that satisfies these constraints and Fig. 4.6 diagrams a 2-D example using four MPI ranks.

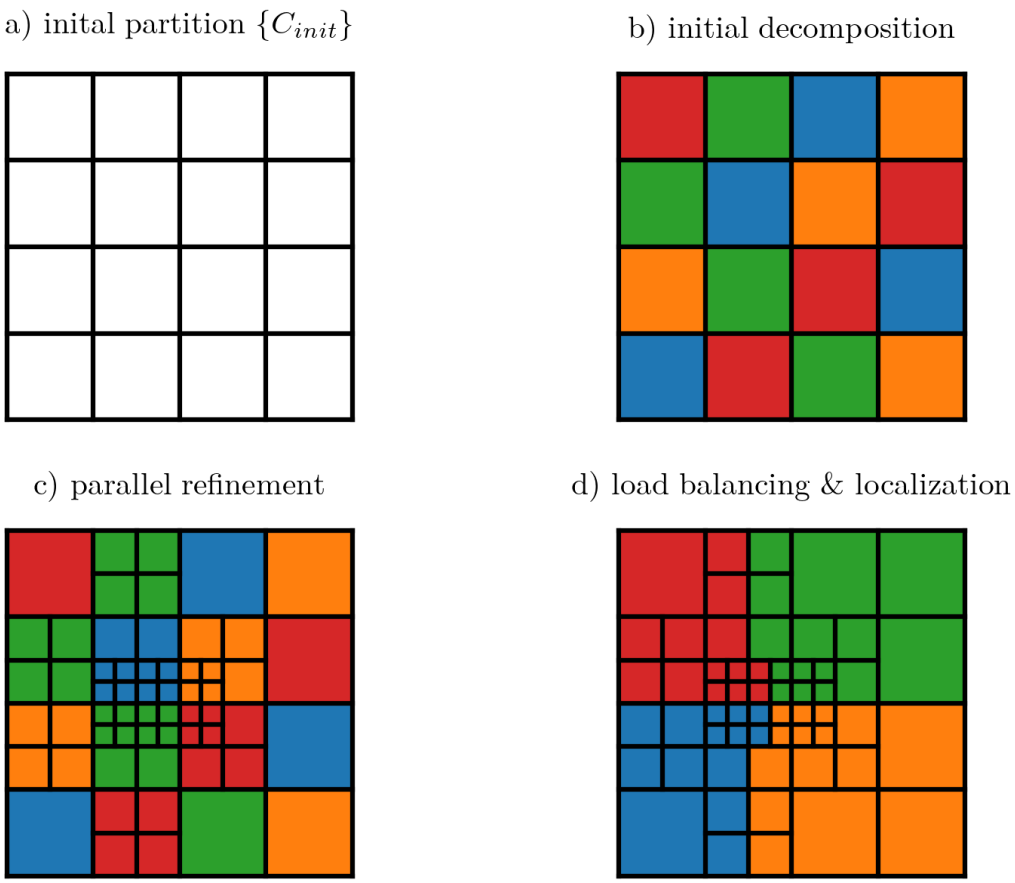


Figure 4.6: A 2-d diagram showing the four steps of the parallel mesh generation and load balancing for a four-rank decomposition (each color is one rank); a) step 1, coarse partitioning of the computational domain, b) step 2, distribution of coarse cells across MPI ranks, c) step 3, parallel local adaptive refinement of the coarse cells, and d) step 4, load balancing and localization with recursive coordinate bisection. In this example, the left side of the domain is more refined than the right side. As a result, after load balancing, the red and blue ranks on the left side represent less volume than the green and the orange on the right, but importantly, the number of cells per rank is balanced (14 or 15 for each rank).

The input to Alg. 4.2 are the domain bounds $L_x$, $L_y$, $L_z$, and initial refinement parameter $\ell_{init}$. The first step in the algorithm is for each processor to divide the domain $[-L_x, L_x] \times [-L_y, L_y] \times [-L_z, L_z]$ into a set of uniform coarse cells $\{C_{init}\}$ with side lengths not exceeding $\ell_{init}$. This is depicted in Fig. 4.6(a). The choice of $\ell_{init}$ is bounded above and below by the following criteria. On the one hand, $\ell_{init}$ should be small enough that the number of cells in the resulting set $\{C_{init}\}$ exceeds the number of MPI ranks. The task of refining each cell is distributed across the ranks, and this condition ensures that each rank participates in the refinement process. On the other hand, if $\ell_{init}$ is too small, the set $\{C_{init}\}$ may become very large and strain the memory limitations, since each rank generates this initial set. In practice, for the examples demonstrated in this work, choosing $\ell_{init}$ so that the number of cells in $\{C_{init}\}$ is 2×-10× the number of MPI ranks works well.

Following this initial coarse partitioning in line 1, the MPI ranks distribute the task of refining each cell in $\{C_{init}\}$ in line 2. This partitioning is shown in Fig. 4.6(b). As shown, we do not attempt to localize each rank before performing the adaptive refinement. While this localization would reduce the communication cost of the post-refinement localization, it can also lead to load-imbalance during the refinement process. Both options are viable, and in this work we choose to localize only after the local refinement is complete.

Following the partitioning of the coarse mesh $\{C_{init}\}$, in line 3, each rank performs the local refinement scheme for each cell it has been assigned, which is shown in Fig. 4.6(c). This generates a set $\{C_{rank,refined}\}$ which contains the cells belonging to the refined mesh. Note that this set is not necessarily balanced among the ranks, nor is it localized. It is not balanced because not all cells in $\{C_{init}\}$ undergo equal refinement; those containing atoms will receive significantly more refinement than those owning the far-field vacuum. Furthermore, the set $\{C_{rank,refined}\}$ is not necessarily localized because the rank may have refined cells from different regions of the domain. Hence, line 4 calls the load balancing routines which use recursive coordinate bisection (RCB) to both localize and balance the refined cells. In particular, we represent each cell by its midpoint, then load balance the set of midpoints. Each rank inherits the entire quadrature cell for each midpoint it is assigned by the load balancing. This load balancing and localization is shown in Fig. 4.6(d). Considering this domain to be $[0, 1] \times [0, 1]$, all cell midpoints below $y = 0.5$ are assigned to either blue or orange, while all cell midpoints above $y = 0.5$ are assigned to either red or green. Similarly, all midpoint below $x = 0.425$ are assigned to either red or blue, while all midpoints above $x = 0.425$ are assigned to either green or orange. The non-symmetric partitioning in the $x$-dimension accounts for the non-symmetric refinement observed in Fig. 4.6(c). Following this load balancing, each rank owns either 14 or 15 quadrature cells.

Following the load balancing, each rank owns a set of cells $\{C_{rank,balanced}\}$, which contain roughly the same number of cells and are localized. Finally, after completing these four steps, each rank generates the quadrature points for its cells in $\{C_{rank,balanced}\}$, generating the corresponding

fine mesh points where necessary. The result is each rank owning a set of quadrature points for the coarse mesh $\{\mathbf{r}_c\}$ and fine mesh $\{\mathbf{r}_f\}$. These sets are balanced and localized which enables efficient treecode parallel scaling, and they admit the mesh-to-mesh interpolation without communication between ranks.

**Algorithm 4.2** parallel mesh generation and load balancing

**input**: domain bounds $L_x$, $L_y$, $L_z$, and initial refinement parameter $\ell_{init}$
**output**: localized set of quadrature points of the coarse mesh $\{\mathbf{r}_c\}$ and the fine mesh $\{\mathbf{r}_f\}$

1: Each rank partitions the domain into a set of coarse cells $\{C_{init}\}$ of size no greater than $\ell_{init}$.
2: The set $\{C_{init}\}$ is divided evenly among the ranks, without regard for locality.
3: Each rank performs the mesh refinement scheme on its coarse cells and generates its refined cells $\{C_{rank,refined}\}$. The refined cells are not necessarily balanced or localized.
4: Localize and load balance the sets of refined cells using recursive coordinate bisection (RCB) with Zoltan library [124]. Each rank now owns a set of cells $\{C_{rank,balanced}\}$, which contain roughly the same number of cells and are localized.

**MPI Communication** The calculation begins after the parallel mesh building, domain decomposition, and load balancing. Throughout the remainder of the calculation communication is only required in two instances, 1) to compute inner products and 2) to compute the convolutions. Computing inner products with distributed memory fields requires a trivial all-to-all communication in which each processor performs its local inner product, and a global reduction is performed on the local values. In practice, this communication is handled with `MPI_Allreduce`. The majority of inner products are performed during orthogonalization and when computing the action of the non-local pseudopotential. The more complex communication occurs within the treecode, when the convolution integrals are approximated. Each process constructs its local source tree and computes its interpolation weights. Next, each processor determines its Locally Essential Tree (LET) [72], which is the union of source clusters throughout the entire domain that its targets interact with. Importantly, to construct the LET, each processor must only obtain $O(\log N)$ remotely owned clusters due to two facts: (1) each target interacts with $O(\log N)$ clusters, and (2) nearby targets interact with a similar subset of the global source tree. Therefore, while the convolution necessarily involves all-to-all communication, the amount of data that each processor needs to obtain only grows logarithmically with the problem size. Further details about the implementation and parallel efficiency of BaryTree can be found in Chapter 2 and [52].

## 4.5 Results

The ground-state energy of several silicon quantum dots and carbon fullerenes were computed using Treecode-Accelerated Green Iteration (TAGI) with ONCV pseudopotentials [121, 122, 123] and the LDA exchange-correlation functional [113, 114]. The ball and stick diagrams of the

molecules are shown in Fig. 4.7. The silicon quantum dots are constructed in Materials Studio [125] by first generating a silicon diamond crystal structure with Si-Si bond length of 4.42 a.u., then removing silicon atoms beyond a cutoff radius. The resulting silicon ball contains dangling electrons (unsatisfied valence shells) and is passivated with hydrogen atoms with Si-H bond length of 2.91 a.u. We generate two examples, one containing 10 silicon atoms passivated with 16 hydrogen atoms, and one containing 30 silicon atoms passivated with 40 hydrogen atoms. The $C_{20}$ fullerene consists of 20 carbon atoms arranged in a dodecahedron centered at the origin, with nuclei at a radius 3.86 a.u. and nearest atoms separated by distance of 2.75 a.u. The $C_{60}$ fullerene consists of 60 carbon atoms arranged in a truncated icosahedron centered at the origin, with nuclei at a radius between 6.47 and 6.56 a.u. and nearest atoms separated by a distance of 2.62 a.u. Figure 4.8 shows the

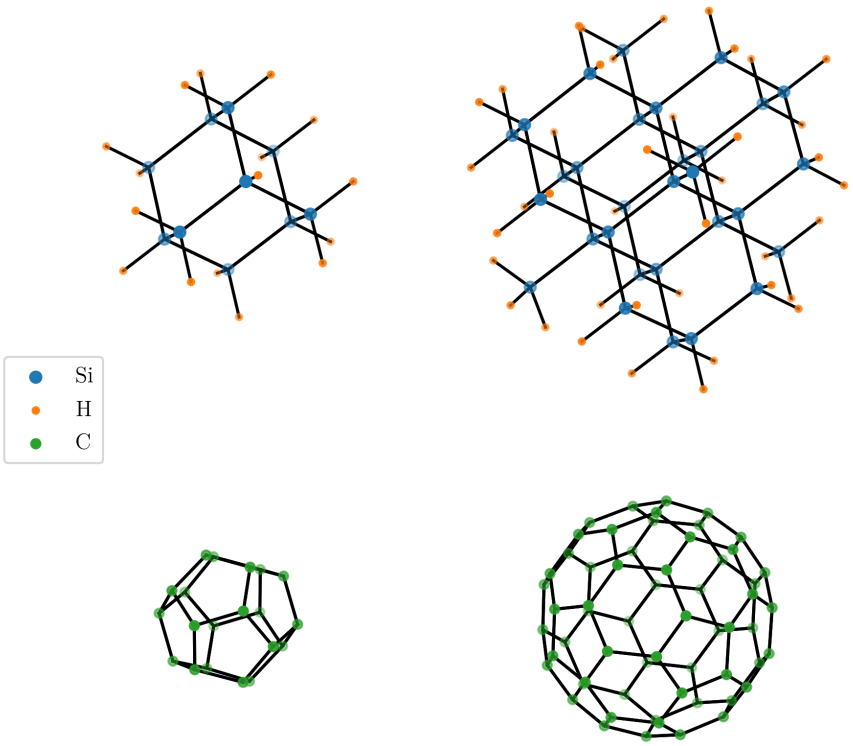


Figure 4.7: Molecules demonstrated in pseudopotential TAGI. Top row are the $Si_{10}H_{16}$ and $Si_{30}H_{40}$ quantum dots, bottom row are the $C_{20}$ and $C_{60}$ fullerenes.

local potentials for the three species used in these calculations. Important to note is that the local potentials for hydrogen and silicon are "softer" than that of carbon, evident by the slopes and depths near $r = 0$. The softness of the local potential is one factor that contributes to the mesh refinement criteria; in particular, we show below that we achieve chemical accuracy with coarser meshes for the silicon quantum dots than for the carbon fullerenes, $\Delta h_{inner} = 1.0$ versus $\Delta h_{inner} = 0.5$.

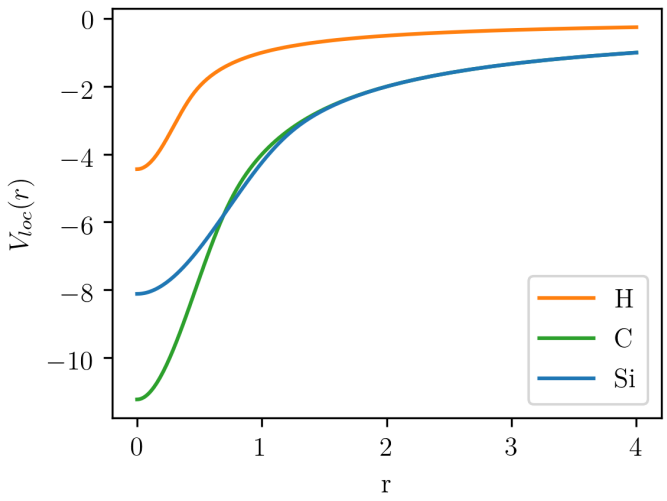


Figure 4.8: Local pseudopotentials for the three atomic species used in the quantum dots and fullerenes. The potentials for hydrogen and silicon are softer than carbon's, enabling chemically accurate calculations of the quantum dots using inner mesh spacing $\Delta h_{inner} = 1.0$, whereas chemical accuracy for the carbon fullerenes requires $\Delta h_{inner} = 0.5$.

Table 4.1 presents the parameters and results for each system. Columns 1-3 provide the system information: 1) chemical formula, 2) number of atoms, and 3) number of valence electrons. Columns 4-5 provide the TAGI discretization parameters: 4) inner mesh spacing $\Delta h_{inner}$ and 5) number of mesh points $N_m$. Columns 6-8 provide the results: 6) ground state energy $E_{TAGI}$, 7) error with respect to reference values computed with DFT-FE [40], and 8) wall clock time in minutes. All calculations use common values for all other parameters, specified below. The iteration tolerances were set as follows. The SCF convergence tolerance is set to $tol_{scf}$ =1e$-$4 per atom. The Green Iteration tolerance is set to $tol_{gi}$ =1e$-$2 for the first SCF iteration, then gradually tightened to 1e$-$5 over the first 10 SCF iterations. The mesh was generated as follows. The computational domain was set to $[-64, 64] \times [-64, 64] \times [-64, 64]$ a.u. The far-field mesh spacing parameter was $\Delta h_{outer} = 8.0$, the coarse-mesh ball radius $r_{coarse} = 2.0$, the fine-mesh ball radius $r_{fine} = 3.0$, the refinement ratio $d_{cf} = 2$. Each cell in the mesh was discretized with quadrature order $p = 3$. The barycentric Lagrange treecode was used for all convolutions with interpolation degree $n = 8$ and multipole acceptance criterion $\theta = 0.85$. The computations were performed on the XSEDE cluster Comet at the San Diego Supercomputer Center on eight nodes, using 32 NVIDIA P100 GPUs. These parameters achieve chemical accuracy with errors below 0.1 mHa/atom for each system.

| system | | | mesh details | | results | | |
|---|---|---|---|---|---|---|---|
| formula | $N_A$ | $N_{valence}$ | $\Delta h_{inner}$ | $N_m$ | $E_{TAGI}$ | error (Ha/atom) | time (min) |
| $Si_{10}H_{16}$ | 26 | 56 | 1.0 | 511232 | -48.73862 | -3.52e-5 | 15.3 |
| $Si_{30}H_{40}$ | 70 | 160 | 1.0 | 1683625 | -141.72169 | -2.18e-5 | 165.8 |
| $C_{20}$ | 20 | 80 | 0.5 | 642048 | -113.46990 | -3.87e-5 | 57.5 |
| $C_{60}$ | 60 | 240 | 0.5 | 1778176 | -342.15098 | -8.53e-5 | 483.8 |

Table 4.1: Ground-state energy computations of atoms and small molecules using TAGI with errors computed with respect to reference values $E_{ref}$ computed using DFT-FE. System parameters (formula, number of atoms $N_A$, number of valence electrons $N_{valence}$), discretization parameters (inner mesh spacing $\Delta h_{inner}$, mesh size $N_m$), and results (ground state energy $E_{TAGI}$, error, and wall clock time).

## 4.6 Conclusion

This Chapter extends Treecode-Accelerated Green Iteration to pseudopotential calculations using the Optimized Norm-Conserving Vanderbilt (ONCV) pseudopotentials [121, 122, 123]. We developed new discretization techniques to achieve pseudopotential chemical accuracy of 0.1 mHa/atom using significantly coarser meshes than the all-electron discretization from Chapter 3. Further, we improved TAGI for both all-electron and pseudopotential calculations by extending calculations to multiple compute nodes using an MPI distributed memory parallelization and porting additional computations to GPUs. These developments enabled TAGI to make full use of the distributed memory implementation of BaryTree from Chapter 2 and to perform calculations of systems as large as the $Si_{30}H_{40}$ quantum dot and $C_{60}$ fullerene using eight GPU compute nodes.

The discretization demands are considerably different for pseudopotential and all-electron calculations. For all-electron TAGI calculations, we dedicated a large effort to local adaptive refinement around the nuclei in order to treat the Coulomb singularities in the external potential. In pseudopotential calculations, the external potential is non-singular and smooth at the nuclei, and instead the challenge is in accurately capturing the oscillatory pseudopotential projectors. In Section 4.3 we developed a two-mesh scheme that takes advantage of the fact that the relatively oscillatory projectors are available *a-priori* whereas the wavefunctions and density, which must be computed, are relatively smooth. The two-mesh scheme computes the wavefunctions on a relatively coarse mesh for efficiency, while interpolating the projectors onto a more refined mesh for accuracy. Further, we identify that this two-mesh scheme only needs to be applied at the end of the calculation, once the density has converged, enabling the coarse mesh to be used exclusively throughout most of the calculation and further improving the efficiency.

In Section 4.4 we developed several high performance computing aspects of TAGI that apply directly to both all-electron and pseudopotential calculations. First, we ported two calculations that were performed on CPUs to GPUs, in particular the mesh-to-mesh interpolation that is used

to interpolate the wavefunctions and potential from the coarse mesh to the fine mesh during the two-mesh scheme, and the wavefunction orthogonalization used in each step of Green Iteration. Second, we developed a distributed memory MPI parallelization that extends TAGI to multiple compute nodes. This involves a domain decomposition, parallel mesh refinement, and load balancing scheme. After these initialization procedures, a vast majority of all communication requirements occur during the convolution integrals, which are already handled via the Locally Essential Trees described in Chapter 2. Throughout the remainder of the TAGI calculations, communication only occurs during inner products which are handled efficiently with `MPI_Allreduce`. This enabled TAGI calculations on Comet using eight nodes and using 32 NVIDIA P100 GPUs, the limit of our available resources.

In Section 4.5 we demonstrated TAGI's new capabilities on several pseudopotential calculations, including the $Si_{30}H_{40}$ quantum dot containing 160 valence electrons and the $C_{60}$ fullerene containing 240 valence electrons, considerably larger test cases than were demonstrated for all-electron calculations. The pseudopotential calculations require many fewer quadrature points per atom to achieve chemical accuracy than all-electron calculations, and therefore we were able to increase the system size and the number of electrons significantly while still keeping the cost of the convolutions reasonably low. However, this introduced new bottlenecks to the calculation. In particular, while we drive the cost of each convolution down to a fraction of a second using the MPI parallelized and GPU accelerated BaryTree, the total number of iterations required increases significantly as the number of electrons and wavefunctions increases. Despite using the wavefunction mixing technique described in Chapter 3, in some instances Green Iteration still requires in excess of 50 iterations to converge a single wavefunction. There are several paths forward that could reduce the number of convolutions required and significantly reduce TAGI's computation time. First, we showed in Chapter 3 that the convergence rate of Green Iteration is tied to the spectral gaps in the differential operator. Therefore, it may be possible to filter the spectrum before performing the fixed-point iteration in order to improve the convergence rate and reduce the number of iterations. Another potential path forward is to use an alternative integral equation formulation of the Kohn-Sham equations, such as those based on inverting the Laplacian instead of the modified Helmholtz operator. In this case, the integral equation would be a generalized eigenvalue problem, rather than a fixed-point problem, which lends itself to a variety of other numerical methods such as Krylov subspace methods [117]. In summary, the techniques developed in this chapter extended the capability of TAGI to much larger system sizes than were previously demonstrated for all-electron calculations, and methods that reduce the number of convolutions required per SCF iteration would be beneficial for further scaling to even larger systems.

# Chapter 5

# Conclusion

This work developed a new numerical method for Kohn-Sham Density Functional Theory based on an integral equation formulation of the Kohn-Sham equations. We accomplished this with a combination of two projects; first, a GPU accelerated and MPI parallelized treecode library, called BaryTree, and second, a Green's function method for KS-DFT called Treecode-Accelerated Green Iteration (TAGI). TAGI is unique in that it is a real-space method without a basis set, it solves fixed-point problems for integral operators rather than eigenvalue problems for differential operators, and it uses BaryTree to compute fast summations of discretized convolutions. TAGI achieves systematic convergence to chemical accuracy for both all-electron and ONCV pseudopotential calculations through adaptive refinement of the real-space domain. TAGI is accelerated with GPUs and parallelized with MPI, suitable for modern high performance computing architectures. Next we discuss the current state of both BaryTree and TAGI and describe several ongoing and future developments that will further improve their performance.

## 5.1 BaryTree

### 5.1.1 Present Work

Chapter 2 presented the development of BaryTree, a library of treecodes based on barycentric interpolation, which are implemented for modern high performance computing environments. Treecodes are fast summation methods for computing N-body interactions, or equivalently for evaluating discretized convolution sums arising in integral equations. The algorithm replaces particle-particle interactions with particle-cluster approximations, and reduces the computational complexity from $O(N^2)$ to $O(N \log N)$. In the case of BaryTree, the particle-cluster approximations are derived from barycentric interpolation at Chebyshev points and are well suited for GPU acceleration. BaryTree contains treecodes based on both barycentric Lagrange and barycentric Hermite interpolation; the Lagrange form is kernel-independent, while the Hermite form requires first order partial derivatives of the kernel.

**HPC Performance**. We developed BaryTree for modern high performance computing environments. GPUs account for a majority of the compute capability on modern clusters (95% of flop/s for Summit at Oak Ridge National Lab [126]). Therefore, we implemented all of the expensive steps of the algorithm for GPUs, taking advantage of nested parallelism within the barycentric interpolation formulas. Furthermore, we parallelized BaryTree with MPI, enabling it to run on multiple compute nodes. We demonstrated BaryTree on up to eight nodes of Comet, containing four GPUs each, for a total of 32 GPUs. We achieved large speedups from the GPU porting, and good parallel scaling across multiple GPU nodes [52].

**Code Availability**. BaryTree is publicly available at github.com/Treecodes/BaryTree under the MIT License. In addition to the treecode library itself, the repository includes several features to aid in usability. These include standalone example calculations on both CPUs and GPUs, demonstrations of interfacing with BaryTree from C or Python applications, and a procedure for creating user-defined convolution kernels.

### 5.1.2 Future developments for BaryTree

**Mixed precision**. BaryTree is implemented with double precision arithmetic in order to ensure high accuracy approximations. However, to accommodate applications that do not demand such high accuracy, it will be useful to implement a mixed precision scheme that uses single precision arithmetic for far-field approximations while still using double precision for the near-field direct interactions. This will accelerate the far-field particle-cluster approximations in cases where full double precision is not required. This may become increasingly important if the GPU hardware continues to support and optimize for single (and even half) precision arithmetic.

**Virtual treetop for fine-grain MPI parallelization**. This version of BaryTree was developed and optimized for GPUs using one MPI rank per GPU. Typical calculations use relatively few MPI ranks (up to 32 on Comet), with each rank performing a large amount of work on its assigned GPU. Some of the design decisions in BaryTree reflect the optimization for GPUs and are not optimal for fine-grain parallel CPU calculations. In particular, the current MPI implementation does not build the virtual top of the global source tree; instead, each rank creates its own tree over its local source particles. This is sufficient for the GPU version given the compute resources we have available (maximum 32 GPUs), however it is not well suited for a fine-grain parallelization using a large number of CPUs or GPUs. In a fine-grain parallelization, well-separated MPI ranks may prefer to interact with clusters that are larger than the root of a remote cluster. In the present implementation, this is not possible, and is a limiting factor in the fine-grain parallel scalability. Developing the virtual treetop will improve the parallel scaling for a large number of MPI ranks by enabling larger cluster approximations, unrestricted by the decomposition.

**Overlapping communication and computation**. Next, we will investigate the advantages of overlapping communication and computation in BaryTree. In particular, one approach is to overlap the near-field computations involving local data with the communication that constructs the Locally Essential Tree for the remote interactions. However, for fine-grain parallel calculations, the local compute becomes a smaller fraction of the total work. Therefore, it may also be advantageous to overlap the computation with one portion of the Locally Essential Tree while performing to communication for another portion. This overlapping will be more impactful in a fine-grain parallelization where the LET construction accounts for a larger fraction of the overall time.

**Cluster-cluster BaryTree**. We are developing a cluster-cluster variant of BaryTree in which both the source particles and the target particles are organized into hierarchical trees of clusters, each cluster represented by a grid of Chebyshev points. The particle-particle interactions are replaced with cluster-cluster interactions in the far-field, determined by a dual-tree traversal [127]. Preliminary results have shown compute phase of the cluster-cluster treecode to be more efficient that the compute phases of the already developed particle-cluster and cluster-particle treecodes for a range of test cases, reducing the time required to achieve a desired accuracy. There remains work to be done in optimizing the downward pass, as well as extending the capability to Hermite interpolation and the singularity subtraction kernels used in TAGI.

**OpenMP Offloading**. NVIDIA GPUs used to be the default for high performance computing clusters, however recently the GPU landscape has started evolving. Other vendors are providing accelerators for some of the upcoming clusters; for example AMD is providing the accelerators for Frontier at Oak Ridge National Laboratory and El Capitan at Lawrence Livermore National Laboratory, and Intel is providing the accelerators for Aurora at Argonne National Laboratory. BaryTree was initially ported to GPUs using OpenACC directives compiled with the PGI compiler. The Portland Group Inc. (PGI) is now owned by NVIDIA, and the future of OpenACC and PGI isn't clear in terms of supporting a variety of accelerators. In the expanding accelerator landscape, we want BaryTree to remain flexible and adaptable to a wide range of accelerators. To accomplish this we plan to also perform the GPU porting with OpenMP offloading, introduced in OpenMP 4.0. The goal is for OpenMP offloading is to make BaryTree agnostic to the exact type of accelerator, whether it is an NVIDIA, AMD, Intel, or other brand. Between the OpenACC and OpenMP offloading, we expect BaryTree to be flexible enough to adapt to the evolving accelerator landscape and achieve good accelerator performance on any HPC clusters.

## 5.2 Treecode-Accelerated Green Iteration

### 5.2.1 Present Work

Chapters 3 and 4 present the development of Treecode-Accelerated Green Iteration (TAGI), an integral equation based numerical method for Kohn-Sham Density Functional Theory. TAGI relies on BaryTree for fast summation of discrete convolution sums. First, in Chapter 3, we developed TAGI for all-electron calculations, which uses the true nuclear potential of each atom and explicitly accounts for each electron. Second, in Chapter 4, we extended TAGI to pseudopotential calculations, which use a regularized nuclear pseudopotential that implicitly accounts for some of the "core" electrons. The challenges, developments, and demonstrations of both types of calculation are discussed below.

**All-Electron calculations**. Achieving sufficiently accurate all-electron calculations required the development of the techniques as described in Chapter 3. In particular, the singular nuclear potentials led to the adaptive mesh refinement scheme based on accurate integration of appropriate test functions combined with a higher order Fejér quadrature rule. The singular convolution kernels led to the development of the singularity subtraction schemes, where we used a standard scheme for the Yukawa kernel and developed a new scheme for the Coulomb kernel [128]. Analysis of the spectra of the integral and differential operators demonstrated potentially slow convergence of the fixed-point iteration for the wavefunctions and eigenvalues, which led to using mixing schemes to accelerate the convergence. We demonstrated all-electron TAGI on a handful of atoms and small molecules as large as the benzene molecule $C_6H_6$, consisting of 42 electrons.

**Pseudopotential calculations**. The focus of Chapter 4 was the development of TAGI for ONCV pseudopotential calculations in order to efficiently extend the method to larger systems. We achieved efficient discretization through the development of a two-mesh scheme that relies on the relative smoothness of the wavefunctions compared to the pseudopotential projectors. For efficiency, the smooth wavefunctions are computed on a relatively coarse mesh, while for accuracy, the relatively oscillatory projectors, which are available *a-priori* as radial data, are evaluated on a relatively fine mesh. This scheme results in significantly fewer quadrature points per atom than the corresponding all-electron mesh, while still achieving chemical accuracy. Furthermore, we ported additional pieces of TAGI to GPUs to improve performance, focusing on two calculations that become non-negligible for larger system sizes, the mesh-to-mesh interpolation and the wavefunction orthogonalization. Additionally, we parallelized TAGI with MPI, including a domain decomposition, parallel mesh refinement scheme, and load balancing. This allowed us to perform TAGI calculations on up to eight GPU nodes on Comet, containing a total of 32 GPUs. We demonstrated pseudopotential TAGI on several larger systems including the $C_{20}$ and $C_{60}$ fullerenes and the $Si_{10}H_{16}$ and $Si_{30}H_{40}$ quantum dots.

### 5.2.2 Future developments for TAGI

The convolutions remain the dominant cost in TAGI calculations, therefore TAGI will benefit directly from many of the previously described future developments of BaryTree. Additionally, there are a number of ideas for modifying TAGI to improve efficiency such as using an alternative integral formulation, using a more efficient real-space discretization, and improving the parallel scaling and GPU efficiency through wavefunction batching.

**Alternative integral forms**. One of the significant challenges faced by TAGI is the sequential nature of the wavefunction calculations. Green iteration begins by computing $\psi_0$, then $\psi_1$ subject to $\psi_1 \perp \psi_0$, and so on. Notably, as described in Chapter 3, each wavefunction is a fixed-point of a different integral operator. This is in contrast to a standard eigenvalue problem, where the task is to compute many eigenfunctions of a single operator. There are techniques for efficiently computing multiple eigenfunctions, such as Krylov subspace methods [117], that cannot be directly applied to Green Iteration. However, the integral formulation used in TAGI is not the only integral formulation of the Kohn-Sham equations. One alternative is to rearrange the Kohn-Sham equations in order to invert the Laplace operator, rather than the modified Helmholtz operator. Instead of a fixed-point equation for a set of integral operators, this inversion results in a generalized eigenvalue problem for a single integral operator. Computing the generalized eigenfunctions and eigenvalues presents its own challenges, however it could be more efficient than sequentially computing the fixed-points in the current formulation.

**Curvilinear mesh**. Another significant challenge faced by TAGI in the case of all-electron calculations is the construction of the adaptively refined mesh. Constructing a mesh which achieves suitable accuracy in spite of the singular nuclear potential and the sharp cusps in the wavefunctions and density, while also maintaining an efficiently low number of quadrature points, was a major challenge. We have presented our approach, where we discretize the domain into cuboid cells according to a refinement criterion, then discretize each cell with a Cartesian tensor product of Chebyshev points. However alternative approaches may be able to also achieve chemical accuracy while requiring significantly fewer quadrature points per atom. One potential alternative meshing scheme is based on curvilinear coordinates. In particular, this scheme could take advantage of radial symmetry very close to the atomic nuclei while efficiently handling the singularity in the nuclear potential. Reducing the number of quadrature points per atom would directly reduce the cost of each convolution and increase the efficiency of TAGI.

**Batch processing wavefunctions**. The BaryTree convolutions are able to efficiently use the GPUs when there are many quadrature points per MPI rank. However, to reduce computation time we increase the number of MPI ranks, thereby reducing the target points per rank. In its current form, BaryTree only supports computing one discrete sum at a time. One possibility for improving efficiency is to investigate a batching scheme where the discrete sums for several wavefunctions are

computed simultaneously, more efficiently saturating the GPU with work, similar to the batched linear algebra routines in BLAS and cuBLAS [129, 130]. Wavefunction batching will involve both modifying the iterations in TAGI to enable simultaneous computation of multiple fixed-points, and generalizing BaryTree to accommodate computing multiple sets of modified weights at each interpolation point, one for each discrete sum in the batch. The GPU compute kernels would then be responsible for computing multiple interactions per target point, thereby increasing the GPU occupation in the event that there are too few target points. Many of the other aspects of BaryTree are identical for all sums in the batch, such as the interaction lists and the MPI communication patterns. With this feature TAGI will scale more efficiently in the regime where a single convolution does not fully saturate the GPUs with work.

### 5.3 Concluding Remarks

We developed a numerical method for Kohn-Sham Density Functional Theory based on an integral equation formulation of the Kohn-Sham equations called Treecode-Accelerated Green Iteration. Compared to the preexisting methods based on the Kohn-Sham differential operator, TAGI enjoys several advantages and faces several unique challenges. In particular, the integral operator presents significant challenges in both accuracy and efficiency. Considering accuracy, the integral operators in TAGI contain singular Green's functions and singular nuclear potentials (for all-electron calculations). Discretizing these integrals accurately was difficult and required the development of several numerical techniques, such as the adaptive mesh refinement scheme for the nuclear singularities and the singularity subtraction schemes for the Green's function singularities. The combination of these techniques enable TAGI to systematically convergence to chemical accuracy. Considering efficiency, evaluating the discretized integral operators is much more expensive than evaluating corresponding differential operators for two reasons. First, the discrete Kohn-Sham differential operator is often sparse, as is the case with finite-difference and finite-element discretizations, for which there are very efficient algorithms for sparse matrix-vector multiplications. Second, the discrete differential operators are local, allowing for distributed memory parallelization of the matrix-vector products typically only requiring neighbor-to-neighbor communication. The integral operator does not enjoy sparsity or locality, it is inherently dense and global because every point in the domain interacts with every other point via the convolution, and the kernels are slowly decaying (compared to applications using a Gaussian kernel, for example, where local truncation is possible). Hence, evaluating the integral operator is expensive and does not parallelize trivially. We overcame these challenges by developing BaryTree, a treecode library based on barycentric interpolation. BaryTree reduces the cost of computation and communication for evaluating the integral operator. At the expense of losing locality and sparsity, TAGI gains

several advantages. First, TAGI avoids numerical differentiation, which tends to amplify high frequency errors. Ultimately, this may enable TAGI to achieve chemical accuracy on coarser meshes than the differential equation based alternatives, however this is reliant on accurately handling the Green's function singularities and our progress so far does not demonstrate a significant reduction in mesh size. Second, TAGI is well suited for GPU calculations, as the convolutions computed with BaryTree enjoy significant GPU speedups, which is important as GPUs dominate more of the available floating point operations in HPC environments. The discrete convolution sums have high arithmetic intensity, that is, each piece of data is used in many arithmetic operations, an important characteristic in exploiting the increased number of floating point operations per second available on GPUs.

The techniques developed in this work have led to a viable numerical method for KS-DFT based on treecode-accelerated integral equations. We demonstrated the method's ability to achieve chemical accuracy for both all-electron and pseudopotential calculations on systems as large as a few hundred electrons. While TAGI is not yet as efficient as the more mature methods such as those based on plane-waves, Gaussian type orbitals, or finite elements, this work demonstrates the feasibility of the integral equation approach to KS-DFT using treecodes. This work overcomes many of the initial challenges faced by the method, and we see many opportunities for further improvement. With continued effort, due to its systematic convergence and efficient GPU acceleration, TAGI has potential to mature into a competitive method for real-space KS-DFT.

# BIBLIOGRAPHY


[1] K. Burke and L. O. Wagner, "DFT in a nutshell," *Int. J. Quantum Chem.*, vol. 113, pp. 96–101, 1 2013.

[2] A. Jain, Y. Shin, and K. A. Persson, "Computational predictions of energy materials using density functional theory," *Nat. Rev. Mater.*, vol. 1, pp. 1–13, 1 2016.

[3] M. Aydinol, A. Kohan, G. Ceder, K. Cho, and J. Joannopoulos, "Ab initio study of lithium intercalation in metal oxides and metal dichalcogenides," *Phys. Rev. B - Condens. Matter Mater. Phys.*, vol. 56, pp. 1354–1365, 7 1997.

[4] F. Zhou, M. Cococcioni, K. Kang, and G. Ceder, "The Li intercalation potential of LiMPO4 and LiMSiO4 olivines with M = Fe, Mn, Co, Ni," *Electrochem. commun.*, vol. 6, pp. 1144–1148, 11 2004.

[5] S. Das, P. Motamarri, V. Gavini, B. Turcksin, Y. W. Li, and B. Leback, "Fast, scalable and accurate finite-element based ab initio calculations using mixed precision computing: 46 PFLOPS simulation of a metallic dislocation system," in *Int. Conf. High Perform. Comput. Networking, Storage Anal. SC*, (New York, NY, USA), pp. 1–11, IEEE Computer Society, 11 2019.

[6] M. J. Frisch, G. W. Trucks, H. B. Schlegel, G. E. Scuseria, M. A. Robb, J. R. Cheeseman, G. Scalmani, V. Barone, B. Mennucci, G. A. Petersson, H. Nakatsuji, M. Caricato, X. Li, H. P. Hratchian, A. F. Izmaylov, J. Bloino, G. Zheng, J. L. Sonnenberg, M. Hada, M. Ehara, K. Toyota, R. Fukuda, J. Hasegawa, M. Ishida, T. Nakajima, Y. Honda, O. Kitao, H. Nakai, T. Vreven, J. A. Montgomery Jr., J. E. Peralta, F. Ogliaro, M. Bearpark, J. J. Heyd, E. Brothers, K. N. Kudin, V. N. Staroverov, R. Kobayashi, J. Normand, K. Raghavachari, A. Rendell, J. C. Burant, S. S. Iyengar, J. Tomasi, M. Cossi, N. Rega, J. M. Millam, M. Klene, J. E. Knox, J. B. Cross, V. Bakken, C. Adamo, J. Jaramillo, R. Gomperts, R. E. Stratmann, O. Yazyev, A. J. Austin, R. Cammi, C. Pomelli, J. W. Ochterski, R. L. Martin, K. Morokuma, V. G. Zakrzewski, G. A. Voth, P. Salvador, J. J. Dannenberg, S. Dapprich, A. D. Daniels, O. Farkas, J. B. Foresman, J. V. Ortiz, J. Cioslowski, and D. J. Fox, "Gaussian 09 Revision E.01."

[7] P. Giannozzi, O. Andreussi, T. Brumme, O. Bunau, M. Buongiorno Nardelli, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, M. Cococcioni, N. Colonna, I. Carnimeo, A. Dal Corso, S. de Gironcoli, P. Delugas, R. A. DiStasio, A. Ferretti, A. Floris, G. Fratesi, G. Fugallo, R. Gebauer, U. Gerstmann, F. Giustino, T. Gorni, J. Jia, M. Kawamura, H.-Y. Ko, A. Kokalj, E. Küçükbenli, M. Lazzeri, M. Marsili, N. Marzari, F. Mauri, N. L. Nguyen, H.-V. Nguyen,

A. Otero-de-la Roza, L. Paulatto, S. Poncé, D. Rocca, R. Sabatini, B. Santra, M. Schlipf, A. P. Seitsonen, A. Smogunov, I. Timrov, T. Thonhauser, P. Umari, N. Vast, X. Wu, and S. Baroni, "Advanced capabilities for materials modelling with Quantum ESPRESSO," *J. Phys. Condens. Matter*, vol. 29, 11 2017.

[8] J. Zhang, Y. Cheng, W. Lu, E. Briggs, A. J. Ramirez-Cuesta, and J. Bernholc, "Large-Scale Phonon Calculations Using the Real-Space Multigrid Method," *J. Chem. Theory Comput.*, vol. 15, no. 12, pp. 6859–6864, 2019.

[9] P. Hohenberg and W. Kohn, "Inhomogeneous Electron Gas," *Phys. Rev.*, vol. 136, pp. B864–B871, 11 1964.

[10] W. Kohn and L. J. Sham, "Self-Consistent Equations Including Exchange and Correlation Effects," *Phys. Rev.*, vol. 140, pp. A1133–A1138, 11 1965.

[11] K. Burke, "Perspective on density functional theory," *J. Chem. Phys.*, vol. 136, no. 15, p. 150901, 2012.

[12] R. O. Jones, "Density functional theory: Its origins, rise to prominence, and future," *Rev. Mod. Phys.*, vol. 87, pp. 897–923, 8 2015.

[13] N. Mardirossian and M. Head-Gordon, "Thirty years of density functional theory in computational chemistry: An overview and extensive assessment of 200 density functionals," *Mol. Phys.*, vol. 115, pp. 2315–2372, 10 2017.

[14] B. Anasori, Y. Xie, M. Beidaghi, J. Lu, B. C. Hosler, L. Hultman, P. R. Kent, Y. Gogotsi, and M. W. Barsoum, "Two-Dimensional, Ordered, Double Transition Metals Carbides (MXenes)," *ACS Nano*, vol. 9, pp. 9507–9516, 10 2015.

[15] X. Chen, Z. Kong, N. Li, X. Zhao, and C. Sun, "Proposing the prospects of Ti3CN transition metal carbides (MXenes) as anodes of Li-ion batteries: A DFT study," *Phys. Chem. Chem. Phys.*, vol. 18, pp. 32937–32943, 12 2016.

[16] N. Pour, Y. Gofer, D. T. Major, and D. Aurbach, "Structural analysis of electrolyte solutions for rechargeable Mg batteries by stereoscopic means and DFT calculations," *J. Am. Chem. Soc.*, vol. 133, pp. 6270–6278, 4 2011.

[17] L. Yu and A. Zunger, "Identification of potential photovoltaic absorbers based on first-principles spectroscopic screening of materials," *Phys. Rev. Lett.*, vol. 108, p. 068701, 2 2012.

[18] L. Yu, R. S. Kokenyesi, D. A. Keszler, and A. Zunger, "Inverse design of high absorption thin-film photovoltaic materials," *Adv. Energy Mater.*, vol. 3, pp. 43–48, 1 2013.

[19] M. P. Balanay and D. H. Kim, "DFT/TD-DFT molecular design of porphyrin analogues for use in dye-sensitized solar cells," *Phys. Chem. Chem. Phys.*, vol. 10, pp. 5121–5127, 8 2008.

[20] F. D. Angelis, S. Fantacci, and A. Selloni, "Alignment of the dye's molecular levels with the TiO2 band edges in dye-sensitized solar cells: a DFT-TDDFT study," *Nanotechnology*, vol. 19, p. 424002, 9 2008.

[21] N. Santhanamoorthi, C. M. Lo, and J. C. Jiang, "Molecular design of porphyrins for dye-sensitized solar cells: A DFT/TDDFT study," *J. Phys. Chem. Lett.*, vol. 4, pp. 524–530, 2 2013.

[22] H. A. Rahnamaye Aliabad, Z. Barzanuni, S. R. Sani, I. Ahmad, S. Jalali-Asadabadi, H. Vaezi, and M. Dastras, "Thermoelectric and phononic properties of (Gd, Tb) MnO3 compounds: DFT calculations," *J. Alloys Compd.*, vol. 690, pp. 942–952, 1 2017.

[23] A. A. Khan, I. Khan, I. Ahmad, and Z. Ali, "Thermoelectric studies of IV-VI semiconductors for renewable energy resources," *Mater. Sci. Semicond. Process.*, vol. 48, pp. 85–94, 6 2016.

[24] D. Zhang, J. Yang, Q. Jiang, Z. Zhou, X. Li, J. Xin, A. Basit, Y. Ren, and X. He, "Multi-cations compound Cu2CoSnS4: DFT calculating, band engineering and thermoelectric performance regulation," *Nano Energy*, vol. 36, pp. 156–165, 6 2017.

[25] S. Goedecker, "Linear scaling electronic structure methods," *Rev. Mod. Phys.*, vol. 71, pp. 1085–1123, 7 1999.

[26] D. R. Bowler and T. Miyazaki, "O(N) methods in electronic structure calculations," *Reports Prog. Phys.*, vol. 75, p. 036503, 2 2012.

[27] J. Aarons, M. Sarwar, D. Thompsett, and C.-K. Skylaris, "Perspective: Methods for large-scale density functional calculations on metallic systems," *J. Chem. Phys.*, vol. 145, p. 220901, 12 2016.

[28] 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, 10 1996.

[29] F. Gygi, "Architecture of Qbox: A scalable first-principles molecular dynamics code," *IBM J. Res. Dev.*, vol. 52, no. 1-2, pp. 137–144, 2008.

[30] J. R. Chelikowsky, N. Troullier, and Y. Saad, "Finite-difference-pseudopotential method: Electronic structure calculations without a basis," *Phys. Rev. Lett.*, vol. 72, pp. 1240–1243, 2 1994.

[31] L. Kronik, A. Makmal, M. L. Tiago, M. M. Alemany, M. Jain, X. Huang, Y. Saad, and J. R. Chelikowsky, "PARSEC – The pseudopotential algorithm for real-space electronic structure calculations: Recent advances and novel applications to nano-structures," *Phys. Status Solidi Basic Res.*, vol. 243, pp. 1063–1079, 4 2006.

[32] J. Bernholc, M. Hodak, and W. Lu, "Recent developments and applications of the real-space multigrid method," *J. Phys. Condens. Matter*, vol. 20, 7 2008.

[33] Y. Saad, J. R. Chelikowsky, and S. M. Shontz, "Numerical Methods for Electronic Structure Calculations of Materials," *SIAM Rev.*, vol. 52, pp. 3–54, 1 2010.

[34] E. Tsuchida and M. Tsukada, "Large-Scale Electronic-Structure Calculations Based on the Adaptive Finite-Element Method," *J. Phys. Soc. Japan*, vol. 67, pp. 3844–3858, 11 1998.

[35] J. E. Pask, B. M. Klein, C. Y. Fong, and P. A. Sterne, "Real-space local polynomial basis for solid-state electronic-structure calculations: A finite-element approach," *Phys. Rev. B*, vol. 59, pp. 12352–12358, 5 1999.

[36] L. Lehtovaara, V. Havu, and M. Puska, "All-electron density functional theory and time-dependent density functional theory with high-order finite elements," *J. Chem. Phys.*, vol. 131, no. 5, pp. 1–10, 2009.

[37] P. Motamarri, M. Nowak, K. Leiter, J. Knap, and V. Gavini, "Higher-order adaptive finite-element methods for Kohn–Sham density functional theory," *J. Comput. Phys.*, vol. 253, pp. 308–343, 11 2013.

[38] P. Motamarri and V. Gavini, "Subquadratic-scaling subspace projection method for large-scale Kohn-Sham density functional theory calculations using spectral finite-element discretization," *Phys. Rev. B*, vol. 90, p. 115127, 9 2014.

[39] B. Kanungo and V. Gavini, "Large-scale all-electron density functional theory calculations using an enriched finite-element basis," *Phys. Rev. B*, vol. 95, p. 035112, 1 2017.

[40] P. Motamarri, S. Das, S. Rudraraju, K. Ghosh, D. Davydov, and V. Gavini, "DFT-FE – A massively parallel adaptive finite-element code for large-scale density functional theory calculations," *Comput. Phys. Commun.*, vol. 246, p. 106853, 1 2020.

[41] M. D. Greenberg, *Applications of Green's Functions in Science and Engineering*. Dover Publications, 2015.

[42] P. Dirac, *The Principles of Quantum Mechanics (International Series of Monographs on Physics)*. Oxford University Press, 4th ed., 1982.

[43] J. Barnes and P. Hut, "A hierarchical O(N log N) force-calculation algorithm," *Nature*, vol. 324, pp. 446–449, 12 1986.

[44] L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations," *J. Comput. Phys.*, vol. 73, no. 2, pp. 325–348, 1987.

[45] V. Rokhlin, "Rapid solution of integral equations of scattering theory in two dimensions," *J. Comput. Phys.*, vol. 86, pp. 414–439, 2 1990.

[46] Z.-H. Duan and R. Krasny, "An adaptive treecode for computing nonbonded potential energy in classical molecular systems," *J. Comput. Chem.*, vol. 22, no. 2, pp. 184–195, 2001.

[47] K. Lindsay and R. Krasny, "A particle method and adaptive treecode for vortex sheet motion in three-dimensional flow," *J. Comput. Phys.*, vol. 172, pp. 879–907, 9 2001.

[48] K. Srinivasan, H. Mahawar, and V. Sarin, "A Multipole Based Treecode Using Spherical Harmonics for Potentials of the Form R-$\lambda$," in *Proc. 5th Int. Conf. Comput. Sci. - Vol. Part I*, ICCS'05, (Berlin, Heidelberg), pp. 107–114, Springer-Verlag, 2005.

[49] P. Li, H. Johnston, and R. Krasny, "A Cartesian treecode for screened coulomb interactions," *J. Comput. Phys.*, vol. 228, pp. 3858–3868, 6 2009.

[50] L. Wang, R. Krasny, and S. Tlupova, "A kernel-independent treecode based on barycentric Lagrange interpolation." arXiv:1902.02250, 2 2019.

[51] R. Krasny and L. Wang, "A treecode based on barycentric Hermite interpolation for electrostatic particle interactions," *Commun. in Comput. Phys. (to appear)*, 2020. arXiv:1902.02250v2.

[52] N. Vaughn, L. Wilson, and R. Krasny, "A GPU-Accelerated Barycentric Lagrange Treecode," in *IEEE Int. Work. Parallel Distrib. Sci. Eng. Comput. (to appear)*, 3 2020. http://arxiv.org/abs/2003.01836.

[53] E. Elsen, M. Houston, V. Vishal, E. Darve, P. Hanrahan, and V. Pande, "N-body simulation on GPUs," *Proc. 2006 ACM/IEEE Conf. Supercomput.*, 2006.

[54] L. Nyland, M. Harris, and J. Prins, "Fast N-body simulation with CUDA," *GPU Gems, Vol. 3*, pp. 677–695, 2009.

[55] I. Lashuk, A. Chandramowlishwaran, M. H. Langston, T.-A. Nguyen, R. Sampath, A. Shringarpure, R. Vuduc, L. Ying, D. Zorin, and G. Biros, "A Massively Parallel Adaptive Fast Multipole Method on Heterogeneous Architectures," *Commun. ACM*, vol. 55, pp. 101–109, 2012.

[56] D. Malhotra and G. Biros, "PVFMM: A Parallel Kernel Independent FMM for Particle and Volume Potentials," *Commun. Comput. Phys.*, vol. 18, no. 3, pp. 808–830, 2015.

[57] D. Malhotra and G. Biros, "Algorithm 967: A Distributed-Memory Fast Multipole Method for Volume Potentials," *ACM Trans. Math. Softw.*, vol. 43, pp. 1–27, 2016.

[58] T. Takahashi, C. Cecka, and E. Darve, "Optimization of the parallel black-box fast multipole method on CUDA," in *2012 Innov. Parallel Comput.*, pp. 1–14, IEEE, 5 2012.

[59] P. Fortin and M. Touche, "Dual tree traversal on integrated GPUs for astrophysical N-body simulations," *Int. J. High Perform. Comput. Appl.*, vol. 33, no. 5, pp. 960–972, 2019.

[60] T. Hamada, T. Narumi, R. Yokota, K. Yasuoka, K. Nitadori, and M. Taiji, "42 TFlops hierarchical N-body simulations on GPUs with applications in both astrophysics and turbulence," in *Proc. Conf. High Perform. Comput. Networking, Storage Anal.*, pp. 1–12, 11 2009.

[61] J. Bédorf, E. Gaburov, and S. P. Zwart, "A sparse octree gravitational N-body code that runs entirely on the GPU processor," *J. Comput. Phys.*, vol. 231, no. 7, pp. 2825–2839, 2012.

[62] J. Bédorf, E. Gaburov, M. S. Fujii, K. Nitadori, T. Ishiyama, and S. P. Zwart, "24.77 Pflops on a gravitational tree-code to simulate the Milky Way Galaxy with 1860 GPUs," in *SC '14 Proc. Int. Conf. High Perform. Comput. Networking, Storage Anal.*, pp. 54–65, IEEE, IEEE Press, 2014.

[63] M. Burtscher and K. Pingali, "An Efficient CUDA Implementation of the Tree-Based Barnes Hut n-Body Algorithm," *GPU Comput. Gems Emerald Ed.*, 2011.

[64] R. Yokota and L. A. Barba, "Comparing the treecode with FMM on GPUs for vortex particle simulations of a leapfrogging vortex ring," *Comput. Fluids*, vol. 45, no. 1, pp. 155–161, 2011.

[65] J.-P. Berrut and L. N. Trefethen, "Barycentric Lagrange Interpolation," *SIAM Rev.*, vol. 46, pp. 501–517, 1 2004.

[66] L. Ying, G. Biros, D. Zorin, and H. Langston, "A New Parallel Kernel-Independent Fast Multipole Method," in *Proc. 2003 ACM/IEEE Conf. Supercomput.*, SC '03, (New York, NY, USA), pp. 14—-, ACM, 2003.

[67] L. Ying, G. Biros, and D. Zorin, "A kernel-independent adaptive fast multipole algorithm in two and three dimensions," *J. Comput. Phys.*, vol. 196, no. 2, pp. 591–626, 2004.

[68] W. Fong and E. Darve, "The black-box fast multipole method," *J. Comput. Phys.*, vol. 228, no. 23, pp. 8712–8725, 2009.

[69] W. B. March, B. Xiao, S. Tharakan, C. D. Yu, and G. Biros, "A Kernel-independent FMM in General Dimensions," in *Proc. Int. Conf. High Perform. Comput. Networking, Storage Anal.*, SC '15, (New York, NY, USA), pp. 24:1—-24:12, ACM, 2015.

[70] R. Wang, C. Chen, J. Lee, and E. Darve, "PBBFMM3D: a Parallel Black-Box Fast Multipole Method for Non-oscillatory Kernels," *arXiv e-prints*, p. arXiv:1903.02153, 3 2019.

[71] H. E. Salzer, "Lagrangian interpolation at the Chebyshev points $x_{n,\nu} \equiv \cos(\nu\pi/n), \nu = 0(1)n$; some unnoted advantages," *Comput. J.*, vol. 15, pp. 156–159, 5 1972.

[72] M. S. Warren and J. K. Salmon, "Astrophysical N-body simulations using hierarchical tree data structures," in *Supercomput. '92 Proc. 1992 ACM/IEEE Conf. Supercomput.*, pp. 570–576, 1992.

[73] L. Lin, J. Lu, and L. Ying, "Numerical methods for Kohn–Sham density functional theory," *Acta Numer.*, vol. 28, pp. 405–539, 5 2019.

[74] X. Gonze, J.-M. Beuken, R. Caracas, F. Detraux, M. Fuchs, G.-M. Rignanese, L. Sindic, M. Verstraete, G. Zerah, F. Jollet, M. Torrent, A. Roy, M. Mikami, P. Ghosez, J.-Y. Raty, and D. Allan, "First-principles computation of material properties: the ABINIT software project," *Comput. Mater. Sci.*, vol. 25, pp. 478–492, 11 2002.

[75] M. D. Segall, P. J. D. Lindan, M. J. Probert, C. J. Pickard, P. J. Hasnip, S. J. Clark, and M. C. Payne, "First-principles simulation: ideas, illustrations and the CASTEP code," *J. Phys. Condens. Matter*, vol. 14, pp. 2717–2744, 3 2002.

[76] P. Giannozzi, S. Baroni, N. Bonini, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, G. L. Chiarotti, M. Cococcioni, I. Dabo, A. Dal Corso, S. de Gironcoli, S. Fabris, G. Fratesi, R. Gebauer, U. Gerstmann, C. Gougoussis, A. Kokalj, M. Lazzeri, L. Martin-Samos, N. Marzari, F. Mauri, R. Mazzarello, S. Paolini, A. Pasquarello, L. Paulatto, C. Sbraccia, S. Scandolo, G. Sclauzero, A. P. Seitsonen, A. Smogunov, P. Umari, and R. M. Wentzcovitch, "QUANTUM ESPRESSO: a modular and open-source software project for quantum simulations of materials," *J. Phys. Condens. Matter*, vol. 21, 9 2009.

[77] T. Loucks and J. C. Slater, "Augmented Plane Wave Method: A Guide to Performing Electronic Structure Calculations," in *Phys. Today*, vol. 20, pp. 92–93, 11 1967.

[78] O. K. Andersen, "Linear methods in band theory," *Phys. Rev. B*, vol. 12, pp. 3060–3083, 10 1975.

[79] E. Wimmer, H. Krakauer, M. Weinert, and A. J. Freeman, "Full-potential self-consistent linearized-augmented-plane-wave method for calculating the electronic structure of molecules and surfaces: O2 molecule," *Phys. Rev. B*, vol. 24, pp. 864–875, 7 1981.

[80] M. Weinert, E. Wimmer, and A. J. Freeman, "Total-energy all-electron density functional method for bulk solids and surfaces," *Phys. Rev. B*, vol. 26, pp. 4571–4578, 10 1982.

[81] E. Sjöstedt, L. Nordström, and D. Singh, "An alternative way of linearizing the augmented plane-wave method," *Solid State Commun.*, vol. 114, pp. 15–20, 3 2000.

[82] G. K. H. Madsen, P. Blaha, K. Schwarz, E. Sjöstedt, and L. Nordström, "Efficient linearization of the augmented plane-wave method," *Phys. Rev. B*, vol. 64, p. 195134, 10 2001.

[83] M. J. Frisch, G. W. Trucks, H. B. Schlegel, G. E. Scuseria, M. A. Robb, J. R. Cheeseman, G. Scalmani, V. Barone, G. A. Petersson, H. Nakatsuji, X. Li, M. Caricato, A. V. Marenich, J. Bloino, B. G. Janesko, R. Gomperts, B. Mennucci, H. P. Hratchian, J. V. Ortiz, A. F. Izmaylov, J. L. Sonnenberg, D. Williams-Young, F. Ding, F. Lipparini, F. Egidi, J. Goings, B. Peng, A. Petrone, T. Henderson, D. Ranasinghe, V. G. Zakrzewski, J. Gao, N. Rega, G. Zheng, W. Liang, M. Hada, M. Ehara, K. Toyota, R. Fukuda, J. Hasegawa, M. Ishida, T. Nakajima, Y. Honda, O. Kitao, H. Nakai, T. Vreven, K. Throssell, J. A. Montgomery Jr., J. E. Peralta, F. Ogliaro, M. J. Bearpark, J. J. Heyd, E. N. Brothers, K. N. Kudin, V. N. Staroverov, T. A. Keith, R. Kobayashi, J. Normand, K. Raghavachari, A. P. Rendell, J. C. Burant, S. S. Iyengar, J. Tomasi, M. Cossi, J. M. Millam, M. Klene, C. Adamo, R. Cammi, J. W. Ochterski, R. L. Martin, K. Morokuma, O. Farkas, J. B. Foresman, and D. J. Fox, "Gaussian˜16 Revision B.01," 2016.

[84] M. Valiev, E. Bylaska, N. Govind, K. Kowalski, T. Straatsma, H. Van Dam, D. Wang, J. Nieplocha, E. Apra, T. Windus, and W. de Jong, "NWChem: A comprehensive and scalable open-source solution for large scale molecular simulations," *Comput. Phys. Commun.*, vol. 181, pp. 1477–1489, 9 2010.

[85] R. Jorgenson and R. Mittra, "Efficient calculation of the free-space periodic Green's function," *IEEE Trans. Antennas Propag.*, vol. 38, pp. 633–642, 5 1990.

[86] R. Coifman, V. Rokhlin, and S. Wandzura, "The fast multipole method for the wave equation: a pedestrian prescription," *IEEE Antennas Propag. Mag.*, vol. 35, pp. 7–12, 6 1993.

[87] L. N. Medgyesi-Mitschang, J. M. Putnam, and M. B. Gedera, "Generalized method of moments for three-dimensional penetrable scatterers," *J. Opt. Soc. Am. A*, vol. 11, p. 1383, 4 1994.

[88] E. Bleszynski, M. Bleszynski, and T. Jaroszewicz, "AIM: Adaptive integral method for solving large-scale electromagnetic scattering and radiation problems," *Radio Sci.*, vol. 31, pp. 1225–1251, 9 1996.

[89] J. L. Volakis and K. Sertel, *Integral equation methods for electromagnetics*. Scitech, 2011.

[90] M. M. Botha, "Solving the volume integral equations of electromagnetic scattering," *J. Comput. Phys.*, vol. 218, no. 1, pp. 141–158, 2006.

[91] L. D. Faddeyev and B. Seckler, "The Inverse Problem in the Quantum Theory of Scattering," *J. Math. Phys.*, vol. 4, pp. 72–104, 1 1963.

[92] B. R. Johnson and D. Secrest, "The Solution of the Nonrelativistic Quantum Scattering Problem without Exchange," *J. Math. Phys.*, vol. 7, pp. 2187–2195, 12 1966.

[93] E. Alt, P. Grassberger, and W. Sandhas, "Reduction of the three-particle collision problem to multi-channel two-particle Lippmann-Schwinger equations," *Nucl. Phys. B*, vol. 2, pp. 167–180, 6 1967.

[94] R. I. Masel, R. P. Merrill, and W. H. Miller, "Quantum scattering from a sinusoidal hard wall: Atomic diffraction from solid surfaces," *Phys. Rev. B*, vol. 12, pp. 5545–5551, 12 1975.

[95] S. K. Adhikari, "Quantum scattering in two dimensions," *Am. J. Phys.*, vol. 54, pp. 362–367, 4 1986.

[96] K. T. Hecht, *Operator Form of Scattering Green's Function and the Integral Equation for the Scattering Problem*, ch. Scattering, pp. 477–480. New York, NY: Springer New York, 2000.

[97] M. H. Kalos, "Monte Carlo Calculations of the Ground State of Three- and Four-Body Nuclei," *Phys. Rev.*, vol. 128, pp. 1791–1795, 11 1962.

[98] Z. Zhao, N. Kovvali, W. Lin, C.-H. Ahn, L. Couchman, and L. Carin, "Volumetric fast multipole method for modeling Schrödinger's equation," *J. Comput. Phys.*, vol. 224, pp. 941–955, 6 2007.

[99] R. J. Harrison, G. I. Fann, T. Yanai, Z. Gan, and G. Beylkin, "Multiresolution quantum chemistry: Basic theory and initial applications," *J. Chem. Phys.*, vol. 121, pp. 11587–11598, 12 2004.

[100] R. J. Harrison, G. Beylkin, F. A. Bischoff, J. A. Calvin, G. I. Fann, J. Fosso-Tande, D. Galindo, J. R. Hammond, R. Hartman-Baker, J. C. Hill, J. Jia, J. S. Kottmann, M. J. Ou, J. Pei, L. E. Ratcliff, M. G. Reuter, A. C. Richie-Halford, N. A. Romero, H. Sekino, W. A. Shelton, B. E. Sundahl, W. S. Thornton, E. F. Valeev, A. Vazquez-Mayagoitia, N. Vence, T. Yanai, and Y. Yokoi, "Madness: A multiresolution, adaptive numerical environment for scientific simulation," *SIAM J. Sci. Comput.*, vol. 38, no. 5, pp. S123–S142, 2016.

[101] M. J. Mohlenkamp and T. Young, "Convergence of Green Iterations for Schrödinger Equations," *Recent Adv. Comput. Sci.*, pp. 201–208, 7 2008.

[102] M. J. Mohlenkamp, "Function space requirements for the single-electron functions within the multiparticle Schrödinger equation," *J. Math. Phys.*, vol. 54, p. 062105, 6 2013.

[103] B. N. Khoromskij, "On tensor approximation of Green iterations for Kohn-Sham equations," *Comput. Vis. Sci.*, vol. 11, pp. 259–271, 9 2008.

[104] M. V. Rakhuba and I. V. Oseledets, "Fast Multidimensional Convolution in Low-Rank Tensor Formats via Cross Approximation," *SIAM J. Sci. Comput.*, vol. 37, pp. A565–A582, 1 2015.

[105] M. V. Rakhuba and I. V. Oseledets, "Grid-based electronic structure calculations: The tensor decomposition approach," *J. Comput. Phys.*, vol. 312, pp. 19–30, 2016.

[106] L. Fejér, "Mechanische Quadraturen mit positiven Cotesschen Zahlen," *Math. Zeitschrift*, vol. 37, pp. 287–309, 12 1933.

[107] L. N. Trefethen, *Spectral Methods in MATLAB*. SIAM, 2008.

[108] P. Rabinowitz, "Approximate Methods of Higher Analysis. L. V. Kantorovich and V. I. Krylov. Translated from the third Russian edition by Curtis D. Benster. Interscience, New York, 1959," *Science (80-. ).*, vol. 134, no. 3487, p. 1358, 1961.

[109] P. M. Anselone, "Singularity subtraction in the numerical solution of integral equations," *J. Aust. Math. Soc. Ser. B. Appl. Math.*, vol. 22, pp. 408–418, 4 1981.

[110] D. Anderson, "Iterative Procedures for Nonlinear Integral Equations," *J. ACM*, vol. 12, pp. 547–560, 10 1965.

[111] H. Fang and Y. Saad, "Two classes of multisecant methods for nonlinear acceleration," *Numer. Linear Algebr. with Appl.*, vol. 16, pp. 197–221, 3 2009.

[112] H. F. Walker and P. Ni, "Anderson Acceleration for Fixed-Point Iterations," *SIAM J. Numer. Anal.*, vol. 49, pp. 1715–1735, 1 2011.

[113] D. M. Ceperley and B. J. Alder, "Ground State of the Electron Gas by a Stochastic Method," *Phys. Rev. Lett.*, vol. 45, pp. 566–569, 8 1980.

[114] J. P. Perdew and A. Zunger, "Self-interaction correction to density-functional approximations for many-electron systems," *Phys. Rev. B*, vol. 23, pp. 5048–5079, 5 1981.

[115] M. A. L. Marques, M. J. T. Oliveira, and T. Burnus, "Libxc: A library of exchange and correlation functionals for density functional theory," *Comput. Phys. Commun.*, vol. 183, no. 10, pp. 2272–2281, 2012.

[116] S. Lehtola, C. Steigemann, M. J. Oliveira, and M. A. Marques, "Recent developments in libxc — A comprehensive library of functionals for density functional theory," *SoftwareX*, vol. 7, pp. 1–5, 1 2018.

[117] L. N. Trefethen and D. I. Bau, *Numerical Linear Algebra*. SIAM Society for Industrial and Applied Mathematics, 2000.

[118] A. M. Rappe, K. M. Rabe, E. Kaxiras, and J. D. Joannopoulos, "Optimized pseudopotentials," *Phys. Rev. B*, vol. 41, pp. 1227–1230, 1 1990.

[119] D. Vanderbilt, "Soft self-consistent pseudopotentials in a generalized eigenvalue formalism," *Phys. Rev. B*, vol. 41, pp. 7892–7895, 4 1990.

[120] N. Troullier and J. L. Martins, "Efficient pseudopotentials for plane-wave calculations," *Phys. Rev. B*, vol. 43, pp. 1993–2006, 1 1991.

[121] G. Kresse and J. Hafner, "Norm-conserving and ultrasoft pseudopotentials for first-row and transition elements," *J. Phys. Condens. Matter*, vol. 6, pp. 8245–8257, 10 1994.

[122] D. R. Hamann, "Optimized norm-conserving Vanderbilt pseudopotentials," *Phys. Rev. B - Condens. Matter Mater. Phys.*, vol. 88, 8 2013.

[123] M. Schlipf and F. Gygi, "Optimization algorithm for the generation of ONCV pseudopotentials," *Comput. Phys. Commun.*, vol. 196, pp. 36–44, 11 2015.

[124] *The Trilinos Project Website*, 2020. https://trilinos.github.io.

[125] "BIOVIA Materials Studio," 2020.

[126] I. Buck, "Reaching the Summit: Accelerated Computing Powering World's Fastest Supercomputer," 2018. https://blogs.nvidia.com/blog/2018/06/08/worlds-fastest-exascale-ai-supercomputer-summit/.

[127] A. W. Appel, "An efficient program for many-body simulation," *SIAM J. Sci. Stat. Comput.*, vol. 6, no. 1, pp. 85–103, 1985.

[128] N. Vaughn, V. Gavini, and R. Krasny, "Treecode-accelerated Green Iteration for Kohn-Sham Density Functional Theory," *arXiv e-prints*, 3 2020. http://arxiv.org/abs/2003.01833v2.

[129] Z. Fiona, "Introducing Batch GEMM Operations," 2017. https://software.intel.com/en-us/articles/introducing-batch-gemm-operations.

[130] C. Cecka, "Pro Tip: cuBLAS Strided Batched Matrix Multiply," 2 2017. https://devblogs.nvidia.com/cublas-strided-batched-matrix-multiply/.