The Dissertation Committee for Kai-Hsin Liou certifies that this is the approved version of the following dissertation:

A PARALLEL EIGENSOLVER FOR REAL-SPACE PSEUDOPOTENTIAL DENSITY FUNCTIONAL THEORY CALCULATIONS

Committee:

James R. Chelikowsky, Supervisor

Alexander A. Demkov

Gyeong S. Hwang

Brian A. Korgel

A PARALLEL EIGENSOLVER FOR REAL-SPACE PSEUDOPOTENTIAL DENSITY FUNCTIONAL THEORY CALCULATIONS

by

Kai-Hsin Liou

Dissertation

Presented to the Faculty of the Graduate School

of the University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

August 2022

This dissertation is dedicated

to our heavenly Father,

for his glory.

And to my parents,

for their endless love.

Acknowledgements

I want to thank my advisor, Jim. He is always patient with listening to my questions and giving his thoughts and suggestions. I have learned a lot from our discussions and chats. I also want to thank my other committee members, Dr. Demkov, Dr. Hwang, and Dr. Korgel, for their insightful advice and comments on my research.

Second, I want to thank my coworkers, Chao, Ariel, and Leeor. Chao showed me how a scientist approaches a problem and constructs solutions. He is friendly and enthusiastic about the progress of our project. It is a pleasure to work with him. Ariel is a creative person. He initialized the space-filling curves project and implemented the first version. Leeor always read the manuscript carefully and gave wonderful suggestions on the work.

I want to thank Ji Yeon and Yang. Both of them encouraged me a lot during my Ph.D. study.

Last but not least, I would like to thank my colleagues, Charles, Masahiro, Yuki, Dingxin, Weiwei, Timothy, Deena, and Mehmet, for teaching me how to use PARSEC and we working together to improve the code. There were also many useful and interesting discussions. I would also like to thank our collaborator, Junjie, for helping us benchmark the code and run large simulations.

A PARALLEL EIGENSOLVER FOR REAL-SPACE PSEUDOPOTENTIAL DENSITY FUNCTIONAL THEORY CALCULATIONS

by

Kai-Hsin Liou, Ph.D.

The University of Texas at Austin, 2022

Supervisor: James R. Chelikowsky

First-principles electronic structure calculations are a popular tool for understanding and predicting properties of materials. Among such methods, the combination of real-space density functional theory and pseudopotentials to solve the Kohn–Sham equation has several advantages. Real-space methods, such as finite differences and finite elements, avoid the global communication needed in fast Fourier transformation and offer better scalability for large calculations on hundreds or thousands of compute nodes. Besides, finite-difference methods with a uniform real-space grid are easy to implement, *e.g.*, the convergence of a Kohn–Sham solution is controlled by a single parameter – the grid spacing.

One promising algorithm for solving the Kohn–Sham eigenvalue problem in real space is the Chebyshev-filtered subspace iteration method (CheFSI). Within this algorithm, the charge density is constructed without regard to a solution for individual eigenvalues. However, for large systems CheFSI may

suffer from super-linear scaling operations such as orthonormalization and the Rayleigh–Ritz procedure.

In the dissertation I will present two improvements in CheFSI to enhance its scalability and accelerate calculation. The first one is a hybrid method that combines a spectrum slicing method and CheFSI. The spectrum slicing method divides a Kohn–Sham eigenvalue problem into subproblems, wherein each subproblem can be solved in parallel using CheFSI. We will show that, by the simulations of confined systems with thousands of atoms, this hybrid method can be faster and possesses better scalability than CheFSI.

The second improvement is a grid partitioning method based on space-filling curves. Space-filling curves based grid partitioning improves the efficiency of the sparse matrix—vector multiplication, which is the key component of CheFSI. We will show that, by computations of confined systems with 50,000 atoms or 200,000 electrons, this method effectively reduces the communication overhead and improves the utilization of the vector processing capabilities provided by most modern parallel computers.

Along with the improvements, I will also present three applications. One is the study of the evolution of density of states of silicon nanocrystals from small ones to their bulk limit. The simulations can hardly be performed without the improvement in sparse matrix–vector multiplication enhanced by space-filling curves based grid partitioning. The other two applications are the studies of proton transfer in liquid water and the adsorption of water on titanium dioxide surfaces.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

Material properties are related to their electronic structure. For example, electrical conductivity of most solids can be understood by their electronic band structure, and mechanical strength depends on how tightly atoms bind together by the "electron glues." The effects of defects on materials, such as the formation of semiconductors, can be interpreted and predicted through electronic structure calculations. Furthermore, the interaction between electrons and phonons plays an important role when we consider superconductors and finite-temperature phenomena in our daily lives. All of these are related to and rely on our understanding of the electronic structure of materials, and whether we are able to calculate it is paramount.

The fundamental equation for electronic structure is the Schrödinger

equation. It has been formidable work to solve it for large systems. A famous quote from P. A. M. Dirac one century ago pointed this out: "The underlying physical laws necessary for the mathematical theory of a large part of physics and the whole of chemistry are thus completely known, and the difficulty is only that the exact application of these laws leads to equations much too complicated to be soluble." [7] This seems still true in our time; however, we have made progress. The increasing richness in theory, the birth of modern massively parallel computers, and the advances in numerical methods make it possible to solve electronic structure of myriad systems of interest by first-principles calculations.

First-principle calculations and simulations have emerged as a third scientific tool on par with experiment and theory. Since the seminal work of Hohenberg and Kohn in establishing the foundation (proving the existence of a unique functional that gives the lowest value when a system is at its ground state) of density functional theory (DFT) [8], and the creation of a practical calculation scheme (on finding the ground-state energy and corresponding electronic charge density) by Kohn and Sham [9], DFT has become an indispensable tool for understanding and predicting material properties. The idea that the ground-state energy is a functional of the electronic charge density leads to a great simplification of the corresponding many-body Schrödinger equation. The minimization of the functional results in the Kohn–Sham equation [9].

Within DFT, real-space methods are increasingly popular as a means for

reduction of the computational load associated with solving the Kohn–Sham equation [10, 11]. Such methods are mathematically robust, highly accurate, and inherently amenable to multiple/hybrid parallelization paradigms (*i.e.*, intra-/inter-node parallelization, GPU acceleration) without the need for global (and often computationally intractable) communication overhead. Furthermore, mixed and customizable boundary conditions can easily be employed. Many implementations of real-space methods exist, including those based on multigrids [12, 13, 14], wavelets [15], numerical atomic orbitals [16, 17], high-order finite differences [18, 19, 20, 21, 22, 23, 24], and finite elements [25, 26].

Among the various real-space methods, the finite-difference method is particularly straightforward and one of the easiest to implement. By using pseudopotentials, which remove the Coulomb singularity [27, 28] and set the energy and length scale to those of the valence states, simple cubic grids can be used. No explicit basis is then needed and convergence is straightforward. In the finite-difference method, Hamiltonian matrices obtained from domain discretization are typically large but sparse [29]. We require an iterative eigensolver to leverage the character of this Hamiltonian matrix to achieve efficiency.

The routinely solvable system size for materials has increased from hundreds of atoms in the 1990s to tens of thousands of atoms, thanks to theoretical developments and advances in computing power. Real-space pseudopotential DFT along with the Chebyshev-filtered subspace iteration (CheFSI)

algorithm (Figure 1.1) has shown the ability to solve the electronic structures of systems containing up to 50,000 atoms [30, 31, 32].

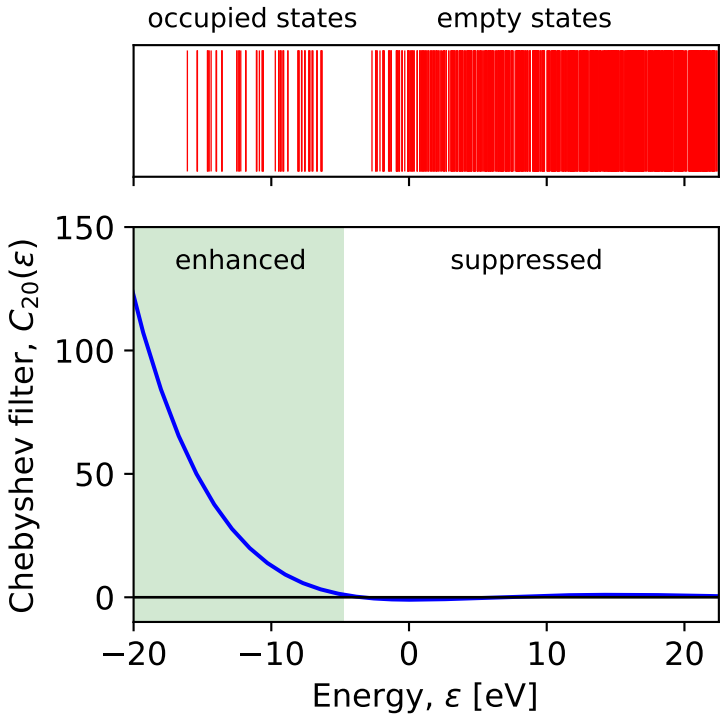

Figure 1.1: Illustration of solving for lowest-energy states from Kohn–Sham equation using Chebyshev-filtered subspace iteration method.

CheFSI is an efficient iterative method for solving the Kohn–Sham equation [17, 30, 31, 33, 34, 35, 36]. CheFSI maintains an evolving subspace approximating the eigenspace. By simultaneously improving the subspace and the potentials, CheFSI reduces the computational load for determining highly accurate eigenstates as the charge density is converged. Once a converged charge density is achieved, the subspace is also the required

eigenspace. In practice, we do not store the Hamiltonian matrix, but we have access to it for sparse matrix–vector multiplication (SpMV). As a result, the performance of CheFSI relies heavily on the efficiency of SpMV of the Hamiltonian matrix with the wave functions. Consequently, an efficient SpMV operation is desired.

The CheFSI algorithm has been adopted in several other large-scale DFT based electronic structure analysis software such as DGDFT [37], SPARC [23], and FE-DFT [38]. A library that implements a Chebyshev Accelerated Subspace Iteration has recently become available [35]. In 2016 Michaud-Rioux *et al.* also demonstrated that a combination of CheFSI and their partial Rayleigh–Ritz method enables the routine calculation of thousands of atoms with only moderate computational resources [17]. The CheFSI algorithm is applied by Banerjee *et al.* [34] to not only the original Kohn–Sham Hamiltonian, but also the projected subspace problem defined in Rayleigh–Ritz method. It is even employed in introductory textbooks [39].

Here, we use PARSEC, a DFT software package featuring real-space calculations and the pseudopotential approximation, to solve the Kohn–Sham equation [19]. The real-space representation of the wave functions has the advantage of making a parallel implementation and software development relatively straightforward [30]. Furthermore, pseudopotential theory and techniques encapsulate the chemically inert core electrons into an ion core potential as part of the external electric field felt by valence electrons. This enables us to solve the Kohn–Sham equation only for the valence electrons, greatly

reducing the computational demands and fixing the energy and length scales by the valence states.

For small to medium sized problems, the time cost of CheFSI is dominated by the multiplication of Hamiltonian matrices with vectors. These SpMV operations can be efficiently parallelized. However, for large systems the Rayleigh–Ritz procedure used to extract the desired eigenvalue approximation and the other dense linear algebra operations required to construct and analyze the projected problem become a bottleneck, as shown in Figure 1.2. As a result, in the dissertation I will propose two methods to address this issue.

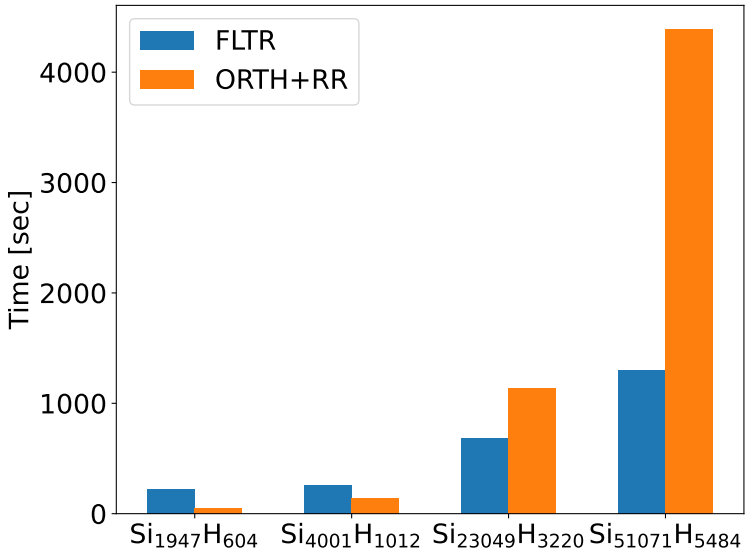

Figure 1.2: The cubic-scaling operations (orthonormalization (ORTH) and Rayleigh–Ritz procedure (RR)) in CheFSI will gradually dominate simulation time cost as the problem size increases. This limits the scalability and hamper larger-scale DFT calculations.

1.2 Overview of Dissertation

In the section on Methods I will start from the time-dependent Schrödinger equation, and derive the time-independent Schrödinger equation for electrons. Then I will discuss the theoretical basis of DFT. By the Hohenberg–Kohn theorems we can solve electronic structure using the Kohn–Sham equation instead of the time-independent Schrödinger equation. Next I will introduce pseudopotential theory and finite-difference discretization, and finally we will obtain the real-space pseudopotential Kohn–Sham equation, which we will use to solve electronic structure problems.

Next, I will introduce the CheFSI method and explain how to apply it to solving the Kohn–Sham equation, followed by the theoretical part of my proposed methods: the spectrum slicing (SS) method and space-filling curves (SFCs) based grid partitioning method.

For SS I will describe how to perform polynomial filtering and review the major computational components of both CheFSI and SS algorithms. Then I will discuss how to construct both Cheybshev and bandpass filter polynomials for computing eigenpairs at the low end and within a spectral slice respectively. Furthermore, I will examine how to partition the spectrum into spectral slices using an estimated density of states obtained from a Lanczos iteration, and refine the partition using Ritz values obtained from previous SCF iterations. Then I will discuss a number of issues related to an efficient implementation of polynomial filtering algorithms. These include

the use of an appropriate data layout for carrying out the computation on a two-dimensional process grid, how to achieve good load balance, and the type of communications involved in the parallel implementation. At the end of Methods, I will introduce first-principles molecular dynamics (FPMD) as we will use it in the studies of proton transfer and water adsorption.

The following part is Results and Discussion. I will present the benchmark results of SS.[1] A number of computational examples are presented in Section 3.1. In particular, we show the performance characteristics of both CheFSI and SS, and note the relative cost of Hamiltonian–vector multiplication and dense matrix computations performed within a subspace. We demonstrate that SS has much better strong parallel scalability compared with CheFSI for a sufficiently large problem when it is performed on an adequately large number of compute cores. In fact, for a large test problem, we show the crossover point (in terms of computational resources) beyond which the time cost used by SS is lower than that used by CheFSI.

I will also present the results of SFCs based grid partitioning[2] and analyze the performance. Afterwards, I will present three applications based on the improvements in the algorithms: evolution of the density of states of silicon nanocrystals, proton transfer in liquid water, and water adsorption on titanium dioxide surfaces.

[1]The work of spectrum slicing has been published on Computer Physics Communications [36].

[2]The work of space-filling curves based grid partitioning has been published on Journal of Chemical Theory and Computation [32].

Finally, I will conclude my dissertation with the key points and findings in my research, and outlook that points to opportunities for future studies.

Chapter 2

Methods

2.1 Description of Electrons – Schrödinger Equation

For a collection of particles (N electrons and M nuclei), these particles can be described by the time-dependent Schrödinger equation:

$$i\hbar\frac{d\Theta(\{\mathbf{r}_i\},\{\mathbf{R}_I\};t)}{dt} = \hat{H}\Theta(\{\mathbf{r}_i\},\{\mathbf{R}_I\};t), \tag{2.1}$$

where $\{\mathbf{r}_i\} = \{\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_N\}$ are the positions of the electrons, $\{\mathbf{R}_I\} = \{\mathbf{R}_1, \mathbf{R}_2, ..., \mathbf{R}_M\}$ are the positions of the nuclei, t is time, the i in front of $\hbar$ is the imaginary unit, $\hbar = \frac{h}{2\pi}$ is the reduced Planck constant, $\Theta(\{\mathbf{r}_i\},\{\mathbf{R}_I\};t)$ is the wave function of the system, and $\hat{H}$ is the Hamiltonian operator which includes kinetic and potential energy of the particles.

If the Hamiltonian does not depend on time, a possible solution of the wave function could be $\Theta(\{\mathbf{r}_i\},\{\mathbf{R}_I\};t) = \Psi(\{\mathbf{r}_i\},\{\mathbf{R}_I\})\exp(-iE_{\text{tot}}t/\hbar)$, where E_{tot} is the total energy of the system. Equation (2.1) becomes

$$\begin{aligned} i\hbar\frac{d\Psi\exp(-iE_{\text{tot}}t/\hbar)}{dt} &= i\hbar\Psi\frac{d\exp(-iE_{\text{tot}}t/\hbar)}{dt} \\ &= E_{\text{tot}}\Psi\exp(-iE_{\text{tot}}t/\hbar) \\ &= \hat{H}\Psi\exp(-iE_{\text{tot}}t/\hbar), \end{aligned} \tag{2.2}$$

and we obtain the time-independent Schrödinger equation for many particles:

$$\hat{H}\Psi(\{\mathbf{r}_i\},\{\mathbf{R}_I\}) = E_{\text{tot}}\Psi(\{\mathbf{r}_i\},\{\mathbf{R}_I\}). \tag{2.3}$$

An Hamiltonian consists of the kinetic energy and potential energy of the underlying particles. For a collection of electrons and nuclei, the Hamiltonian is

$$\begin{aligned} \hat{H} &= \text{KE}_{\text{n}} + \text{KE}_{\text{e}} + \text{PE}_{\text{nn}} + \text{PE}_{\text{ne}} + \text{PE}_{\text{ee}} \\ &= \sum_I \frac{\hat{\mathbf{p}}_I^2}{2M_I} + \sum_i \frac{\hat{\mathbf{p}}_i^2}{2m_{\text{e}}} \\ &+ \frac{1}{2}\frac{e^2}{4\pi\epsilon_0}\sum_{\substack{I,J \\ I\neq J}} \frac{Z_I Z_J}{|\mathbf{R}_I - \mathbf{R}_J|} - \frac{e^2}{4\pi\epsilon_0}\sum_{I,i}\frac{Z_I}{|\mathbf{R}_I - \mathbf{r}_i|} + \frac{1}{2}\frac{e^2}{4\pi\epsilon_0}\sum_{\substack{i,j \\ i\neq j}}\frac{1}{|\mathbf{r}_i - \mathbf{r}_j|}, \end{aligned}$$

where KE_{n} is the kinetic energy of the nuclei, KE_{e} is the kinetic energy of the electrons, PE_{nn} is the potential energy between the nuclei, PE_{ne} is the potential energy between the nuclei and electrons, PE_{ee} is the potential

energy between the electrons, $\hat{\mathbf{p}}_I$ and M_I are the momentum and mass of nucleus I, $\hat{\mathbf{p}}_i$ and m_e are the momentum and mass of electron i, e is the elementary charge, ϵ_0 is the permittivity of free space, Z_I and Z_J are the atomic numbers of nuclei I and J.

Substituting the quantum-mechanical form for the momenta, $\hat{\mathbf{p}}_I = -i\hbar\nabla_I$ and $\hat{\mathbf{p}}_i = -i\hbar\nabla_i$, and adopting the Hartree atomic units (see Appendix A), the Hamiltonian becomes

$$\begin{aligned}\hat{H} = &-\sum_I \frac{\nabla_I^2}{2M_I} - \sum_i \frac{\nabla_i^2}{2} \\ &+ \frac{1}{2}\sum_{\substack{I,J \\ I\neq J}} \frac{Z_I Z_J}{|\mathbf{R}_I - \mathbf{R}_J|} - \sum_{I,i} \frac{Z_I}{|\mathbf{R}_I - \mathbf{r}_i|} + \frac{1}{2}\sum_{\substack{i,j \\ i\neq j}} \frac{1}{|\mathbf{r}_i - \mathbf{r}_j|}.\end{aligned} \tag{2.4}$$

We apply Born–Oppenheimer approximation, which assumes that the motions of electrons and nuclei are decoupled and the wave function of the system can be separated as a product of an electronic wave function and a nuclear wave function:

$$\Psi(\{\mathbf{r}_i\}, \{\mathbf{R}_I\}) = \Phi_{\{\mathbf{R}_I\}}(\{\mathbf{r}_i\})\chi(\{\mathbf{R}_I\}),$$

where the subscript of Φ denotes that Φ is solved under a specific nuclear configuration. Substituting the wave function ansatz into Equation (2.3) and

writing down explicitly the terms of the Hamiltonian, we have

$$\left[-\sum_I \frac{\nabla_I^2}{2M_I} -\sum_i \frac{\nabla_i^2}{2} +\frac{1}{2}\sum_{\substack{I,J\\I\neq J}} \frac{Z_I Z_J}{|\mathbf{R}_I-\mathbf{R}_J|} -\sum_{I,i} \frac{Z_I}{|\mathbf{R}_I-\mathbf{r}_i|} +\frac{1}{2}\sum_{\substack{i,j\\i\neq j}} \frac{1}{|\mathbf{r}_i-\mathbf{r}_j|}\right] \Phi_{\{\mathbf{R}_I\}}\chi = E_{\text{tot}}\Phi_{\{\mathbf{R}_I\}}\chi. \tag{2.5}$$

Next, we define the electronic Hamiltonian, $\hat{H}_{\text{e}}$, and obtain the Schrödinger equation for the electrons:

$$\begin{aligned}\hat{H}_{\text{e}}\Phi_{\{\mathbf{R}_I\}} &= \left[-\sum_i \frac{\nabla_i^2}{2} -\sum_{I,i} \frac{Z_I}{|\mathbf{R}_I-\mathbf{r}_i|} +\frac{1}{2}\sum_{\substack{i,j\\i\neq j}} \frac{1}{|\mathbf{r}_i-\mathbf{r}_j|}\right] \Phi_{\{\mathbf{R}_I\}} \\ &= E_{\{\mathbf{R}_I\}}\Phi_{\{\mathbf{R}_I\}},\end{aligned} \tag{2.6}$$

where $E_{\{\mathbf{R}_I\}}$ is the total energy of the electrons. Note that the nuclear positions $\{\mathbf{R}_I\}$ enters this equation parametrically. With this definition, Equation (2.5) becomes

$$E_{\{\mathbf{R}_I\}}\Phi_{\{\mathbf{R}_I\}}\chi + \left[-\sum_I \frac{\nabla_I^2}{2M_I} +\frac{1}{2}\sum_{\substack{I,J\\I\neq J}} \frac{Z_I Z_J}{|\mathbf{R}_I-\mathbf{R}_J|}\right] \Phi_{\{\mathbf{R}_I\}}\chi = E_{\text{tot}}\Phi_{\{\mathbf{R}_I\}}\chi. \tag{2.7}$$

After rearranging the equation and eliminating the electronic wave function,

we obtain the nuclear Schrödinger equation:

$$\left[-\sum_I \frac{\nabla_I^2}{2M_I} + \frac{1}{2} \sum_{\substack{I,J \\ I \neq J}} \frac{Z_I Z_J}{|\mathbf{R}_I - \mathbf{R}_J|} + E_{\{\mathbf{R}_I\}} \right] \chi = E_{\text{tot}} \chi. \tag{2.8}$$

Note that the electronic energy, $E_{\{\mathbf{R}_I\}}$, serves as the potential-energy surface that the nuclei move along.

2.2 Density Functional Theory

Density functional theory (DFT) is an alternative approach to solving the electronic Schrödinger equation (Equation (2.6)) and obtaining the total electronic energy. To simplify the derivation, let us rewrite Equation (2.6) in terms of operator symbols:

$$\left(\hat{T} + \hat{V}_{\text{ext}} + \hat{V}_{\text{ee}} \right) \Phi = E\Phi, \tag{2.9}$$

where $\hat{T} = -\sum_i \frac{\nabla_i^2}{2}$ is the kinetic energy operator, $\hat{V}_{\text{ext}} = -\sum_{I,i} \frac{Z_I}{|\mathbf{R}_I - \mathbf{r}_i|}$ is the potential energy the electrons feel from the nuclei, $\hat{V}_{\text{ee}} = \frac{1}{2} \sum_{\substack{i,j \\ i \neq j}} \frac{1}{|\mathbf{r}_i - \mathbf{r}_j|}$ is the potential energy due to the interaction between the electrons, $\Phi = \Phi(\{\mathbf{r}_i\})$ is the electronic wave function, and E is the total energy of the electrons.

2.2.1 First and Second Hohenberg–Kohn Theorem

First Hohenberg–Kohn Theorem

The first Hohenberg–Kohn theorem states that the external potential, $\hat{V}_{\text{ext}}$ is a unique functional of the electronic charge density,

$$n(\mathbf{r}) = N \int d\mathbf{r}_2 d\mathbf{r}_3 ... d\mathbf{r}_N \Phi^{\dagger}(\mathbf{r}, \mathbf{r}_2, ..., \mathbf{r}_N)\Phi(\mathbf{r}, \mathbf{r}_2, ..., \mathbf{r}_N),$$

up to a constant. Note that the wave function, $\Phi(\mathbf{r}, \mathbf{r}_2, ..., \mathbf{r}_N)$, satisfies anti-symmetry and is normalized, such that

$$\int d\mathbf{r}\ n(\mathbf{r}) = N \int d\mathbf{r} d\mathbf{r}_2 d\mathbf{r}_3 ... d\mathbf{r}_N \Phi^{\dagger}(\mathbf{r}, \mathbf{r}_2, ..., \mathbf{r}_N)\Phi(\mathbf{r}, \mathbf{r}_2, ..., \mathbf{r}_N) = N.$$

On the other hand, $\hat{V}_{\text{ext}}$ uniquely determines the ground-state wave function of the system, and hence the total energy. As a result, the total energy, E, is a functional of $n(\mathbf{r})$. This means that it is possible to devise a fictitious system that has the same charge density, and the total energy of the fictitious system is the same as that from the Schrödinger equation.

To prove the first Hohenberg–Kohn theorem, we start from two different Hamiltonians:

$$\hat{H} = \hat{T} + \hat{V}_{\text{ext}} + \hat{V}_{\text{ee}},$$

and

$$\hat{H}' = \hat{T} + \hat{V}'_{\text{ext}} + \hat{V}_{\text{ee}}.$$

The difference is in the external potential (two different sets of nuclei). We denote the ground-state wave function of each system as Φ_0 and $\Phi_0^{'}$, respectively. They should be different because the external potentials are different. We then assume *that the two different external potentials,* $\hat{V}_{\text{ext}}$ *and* $\hat{V}_{\text{ext}}^{'}$*, lead to the same ground-state charge densities,* $n_0(\mathbf{r})$, through the Hamiltonians, $\hat{H}$ and $\hat{H}^{'}$.

For the next step we compute the total energy from $\hat{H}^{'}$ using the ground-state wave function of the first system, Φ_0. We expect to obtain an energy that is greater than the ground-state energy, since Φ_0 is not the same as the ground-state wave function of the second system, $\Phi_0^{'}$. That is,

$$
\begin{aligned}
E = & \langle \Phi_0 | \hat{H}^{'} | \Phi_0 \rangle \\
& = \int \Phi_0^{\dagger} (\hat{T} + \hat{V}_{\text{ext}}^{'} + \hat{V}_{\text{ee}}) \Phi_0 d\mathbf{r} \\
& = \int \Phi_0^{\dagger} (\hat{T} + \hat{V}_{\text{ext}}^{'} + \hat{V}_{\text{ee}} + \hat{V}_{\text{ext}} - \hat{V}_{\text{ext}}) \Phi_0 d\mathbf{r} \\
& = \int \Phi_0^{\dagger} (\hat{T} + \hat{V}_{\text{ext}} + \hat{V}_{\text{ee}}) \Phi_0 d\mathbf{r} + \int \Phi_0^{\dagger} (\hat{V}_{\text{ext}}^{'} - \hat{V}_{\text{ext}}) \Phi_0 d\mathbf{r} \\
& = E_0 + \int (V_{\text{ext}}^{'} - V_{\text{ext}}) n(\mathbf{r}) d\mathbf{r} \\
& > \int (\Phi_0^{'})^{\dagger} \hat{H}^{'} \Phi_0^{'} d\mathbf{r} \\
& = E_0^{'},
\end{aligned}
$$

where E_0 and $E_0^{'}$ are the ground-state energy of the first and second system, respectively. On the other hand, we can perform the same derivation for $\Phi_0^{'}$

through $\hat{H}$ and obtain

$$E_0^{'} + \int (V_{\text{ext}} - V_{\text{ext}}^{'}) n(\mathbf{r}) d\mathbf{r} > E_0.$$

If we add up the above two inequalities on both sides, we find

$$E_0 + E_0^{'} > E_0 + E_0^{'}, \tag{2.10}$$

which is wrong. As a result, *different external potentials must result in different charge densities*. That is, the external potential (and the Hamiltonian and total energy) is uniquely determined by its charge density.

Second Hohenberg–Kohn Theorem

The second Hohenberg–Kohn theorem states *that the universal functional ($F = \hat{T} + \hat{V}_{\text{ee}}$) delivers the lowest energy if and only if the input charge density is the true ground-state density, $n_0(\mathbf{r})$*. In other words, if we know the universal functional, a charge density that gives the lowest energy would be the ground-state energy of the system:

$$F[n] + \int V_{\text{ext}} n(\mathbf{r}) d\mathbf{r} \geq F[n_0] + \int V_{\text{ext}} n_0(\mathbf{r}) d\mathbf{r} = E_0. \tag{2.11}$$

In sum, Hohenberg–Kohn theorems prove that the ground-state energy of a non-degenerate system is a functional of the density of the system. That is, $E_0 = F[n_0(\mathbf{r})]$. This greatly simplifies electronic structure problems. Take

$Si_{29}H_{36}$ as an example. There are $29 \times 14 + 36 \times 1 = 442$ electrons in the system, and each electron has three spatial variables. As a result, the solution of the electronic Schrödinger equation (Equation (2.6)) for the system is a function of $442 \times 3 = 1326$ spatial variables. However, by Hohenberg–Kohn theorems we know that the ground-state energy is a functional of the charge density of the system, which is a function of only three spatial variables.

2.2.2 Kohn–Sham Method

The Hohenberg–Kohn theorems allow a description of the ground-state energy of a system if we know (1) the universal functional, $F[n] = \hat{T} + \hat{V}_{\text{ee}}$, and (2) the charge density of the ground state. If we only know the universal functional, but not the ground-state charge density, we are able to approximate the total energy using trial functions for the charge density, thanks to the variational principle. Unfortunately we do not know the universal functional.

Kohn and Sham proposed a method to compute the total energy approximately based on DFT [9]. Their method assumes (1) A fictitious system of independent electrons that results in the same charge density as the one obtained from the original system of interacting electrons. (2) The kinetic-energy contribution and the electron-electron interaction-energy contribution can be largely captured by the kinetic energy and the Hartree energy of these independent electrons. (3) The difference due to the above approximations and the other many-body effects are lumped into an "exchange-correlation

energy term" and approximated by the local density (local-density approximation). That is,

$$\begin{aligned} E &= F[n] + \langle\Phi|\hat{V}_{\rm ext}|\Phi\rangle \\ &= \langle\Phi|\hat{T} + \hat{V}_{\rm ext} + \hat{V}_{\rm ee}|\Phi\rangle \\ &= \langle\Phi|\hat{T}_{\rm KS} + \hat{V}_{\rm ext} + \hat{V}_{\rm H} + \hat{T} - \hat{T}_{\rm KS} + \hat{V}_{\rm ee} - \hat{V}_{\rm H}|\Phi\rangle \\ &= \langle\Phi|\hat{T}_{\rm KS} + \hat{V}_{\rm ext} + \hat{V}_{\rm H} + \hat{V}_{\rm xc}|\Phi\rangle \\ &= \langle\Phi|\hat{T}_{\rm KS}|\Phi\rangle + \langle\Phi|\hat{V}_{\rm ext}|\Phi\rangle + \langle\Phi|\hat{V}_{\rm H}|\Phi\rangle + \langle\Phi|\hat{V}_{\rm xc}|\Phi\rangle \\ &= -\sum_i \int d\mathbf{r}\phi_i^\dagger \frac{\nabla^2}{2}\phi_i + \int d\mathbf{r} V_{\rm ext}(\mathbf{r}) n(\mathbf{r}) + \frac{1}{2}\int d\mathbf{r}d\mathbf{r}' \frac{n(\mathbf{r})n(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|} + E_{\rm xc}. \end{aligned} \tag{2.12}$$

Then we define a Lagrangian, $\mathcal{L}[n]$, that includes the total energy and the constraints required by the normalization of the single-particle orbitals for the independent electrons through the help of Lagrange multipliers, λ_i:

$$\mathcal{L}[n] = E + \sum_i \lambda_i (1 - \phi_i^\dagger \phi_i).$$

To find a minimum of $\mathcal{L}[n]$, we need to satisfy

$$\frac{\delta\mathcal{L}[n]}{\delta n} = 0, \tag{2.13}$$

and

$$\frac{\delta\mathcal{L}[n]}{\delta\lambda_k} = 0, \text{ for } 1 \le k \le N, \tag{2.14}$$

where $n = \sum_i^N |\phi_i|^2$. The latter ones are automatically satisfied since they are the constraints we impose. For the former, we have

$$\frac{\delta \mathcal{L}[n]}{\delta n} = \sum_k \frac{\delta \phi_k^\dagger}{\delta n} \frac{\delta \mathcal{L}[n]}{\delta \phi_k^\dagger} = 0. \tag{2.15}$$

Since $\frac{d\phi_k^\dagger}{dn}$ is non-zero, we then have

$$\frac{\delta \mathcal{L}[n]}{\delta \phi_k^\dagger} = 0, \tag{2.16}$$

for every k. Expanding $\mathcal{L}[n]$, we obtain

$$\frac{\delta E}{\delta \phi_k^\dagger} + \sum_i \lambda_i \frac{\delta}{\delta \phi_k^\dagger}(1 - \phi_i^\dagger \phi_i) = 0. \tag{2.17}$$

With further simplification, the above equation becomes

$$\left[-\frac{\nabla^2}{2} + V_{\text{ext}}(\mathbf{r}) + \int d\mathbf{r}' \frac{n(\mathbf{r}')}{|\mathbf{r}' - \mathbf{r}|} + V_{\text{xc}}[n] \right] \phi_k - \lambda_k \phi_k = 0, \tag{2.18}$$

where $V_{\text{xc}}[n] = \frac{\delta E_{\text{xc}}}{\delta n}$. Interpreting the Lagrange multipliers as Kohn–Sham energy levels and the functions, ϕ_k, as Kohn–Sham single-particle orbitals, we obtain the Kohn–Sham equation:

$$\left(-\frac{1}{2}\nabla^2 + V_{\text{ext}}(\mathbf{r}) + V_{\text{H}}(\mathbf{r}) + V_{\text{xc}}[n] \right) \phi_i(\mathbf{r}) = \varepsilon_i \phi_i(\mathbf{r}), \tag{2.19}$$

where

$$V_{\text{ext}}(\mathbf{r}) = -\sum_I \frac{Z_I}{|\mathbf{r} - \mathbf{R}_I|}, \tag{2.20}$$

$$V_{\text{xc}}[n] = \frac{\delta E_{\mathbf{xc}}}{\delta n}, \tag{2.21}$$

and

$$V_{\text{H}}(\mathbf{r}) = \int d\mathbf{r}' \frac{n(\mathbf{r}')}{|\mathbf{r}' - \mathbf{r}|}, \tag{2.22}$$

which is often recast into the Poisson equation form (see Appendix B for the derivation):

$$\nabla^2 V_{\text{H}}(\mathbf{r}) = -4\pi n(\mathbf{r}). \tag{2.23}$$

The Kohn–Sham equation is a non-linear eigenvalue problem, since the effective potential, $V_{\text{eff}} = V_{\text{ext}} + V_{\text{H}} + V_{\text{xc}}$, depends on its solution. One approach to solving this equation is to construct a self-consistent field (SCF). A flowchart of the SCF construction is shown in Figure 2.1. The effective potential is calculated from an initial charge density that is constructed from the superposition of atomic charge densities. Using this potential we solve a linear eigenvalue problem and obtain a new charge density and the corresponding effective potential. The resulting effective potential is mixed with those of previous steps to achieve a self-consistent potential, *i.e.*, the procedure is iterated until the difference between the input potential and the output potential is within a specified tolerance.

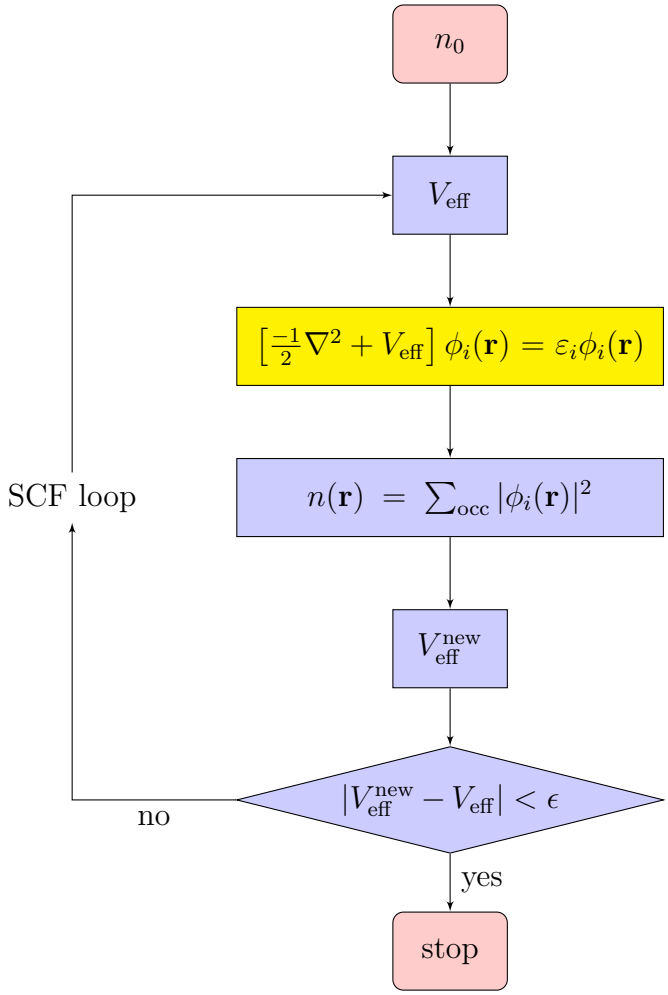

Figure 2.1: Flowchart of an SCF DFT calculation

The single-particle orbital ϕ_i and the corresponding energy ε_i form the ith eigenpair of the Kohn–Sham Hamiltonian H_{KS}, which is a functional of the ground-state electronic charge density $n(\mathbf{r})$ defined by

$$n(\mathbf{r}) = 2\sum_{i=1}^{N_s} f_i|\phi_i(\mathbf{r})|^2, \tag{2.24}$$

where the prefactor 2 accounts for electronic spins (which we do not distinguish in this work since we are not considering magnetic systems), N_s is the number of states being calculated, and f_i is the Fermi–Dirac distribution function:

$$f(t) = \frac{1}{\exp((t - \varepsilon_{\mathrm{F}})/k_{\mathrm{B}}T) + 1} \tag{2.25}$$

evaluated at the single-particle energy ε_i. Here, k_{B} is the Boltzmann constant, and T is the temperature. The eigenvalues ε_i's are ordered in ascending order, *i.e.*, $\varepsilon_1 \leq \varepsilon_2 \leq \cdots$. The first N_{occ} eigenpairs, (ϕ_i, ε_i), $i = 1, 2, ..., N_{\mathrm{occ}}$, are occupied states, and all others are unoccupied states.

If there is a spectral gap between the occupied and unoccupied states, the Fermi level ε_{F} is between $\varepsilon_{N_{\mathrm{occ}}}$ and $\varepsilon_{N_{\mathrm{occ}}+1}$ and ensures the electronic charge density $n(\mathbf{r})$ integrates to $2N_{\mathrm{occ}}$. When the temperature T is relatively low, $f(t)$ decreases from 1 to 0 rapidly near the Fermi level. As a result, the number of effective terms N_s to be summed in (2.24) may include a few partially occupied states.

2.3 Pseudopotential Theory

Pseudopotential theory is the foundation of pseudopotentials which incorporate the effects of core electrons into the external potential and result in an effective ion core potential felt by valence electrons.

We solve the all-electron Kohn–Sham equation for an atom first, which

is computationally easy:

$$\left(-\frac{1}{2}\nabla^2 + V_{\text{ext}} + V_{\text{H}} + V_{\text{xc}}\right)\phi_i^{\text{AE}}(\mathbf{r}) = \varepsilon_i^{\text{AE}}\phi_i^{\text{AE}}(\mathbf{r}), \tag{2.26}$$

where we obtain the all-electron wave functions, $\phi_i^{\text{AE}}(\mathbf{r})$, and energy levels, $\varepsilon_i^{\text{AE}}$. Then we choose nodeless and smooth pseudo wave functions, $\phi_i^{\text{PS}}(\mathbf{r})$, such that

$$\phi_i^{\text{PS}}(\mathbf{r}) = \begin{cases} \text{a smooth function}, & \text{for } |\mathbf{r}| < r_{\text{c}}, \\ \phi_i^{\text{AE}}(\mathbf{r}), & \text{for } |\mathbf{r}| \geq r_{\text{c}}, \end{cases} \tag{2.27}$$

where r_{c} is a cutoff radius. We also choose energy levels to be the same as the all-electron ones.

Substituting the pseudo-wave function of a target state and the corresponding energy level into the Kohn–Sham equation and computing the Hartree and exchange-correlation terms using the charge density from the pseudo-wave function, we can solve for the effective pseudopotential for the target state:

$$V_{\text{ext}}^{\text{PS}}(\mathbf{r}) = \varepsilon_i^{\text{AE}} + \frac{1}{2}\left(\frac{\nabla^2\phi_i^{\text{PS}}(\mathbf{r})}{\phi_i^{\text{PS}}(\mathbf{r})}\right) - V_{\text{H}}(\mathbf{r}) - V_{\text{xc}}(\mathbf{r}). \tag{2.28}$$

In most chemical environments, only valence electrons participate in bonding (*i.e.*, the core electrons almost do not change). However, the orbitals of valence electrons need to be orthogonal to the core states, *i.e.*, possess a nodal structure. This may necessitate a very fine real-space grid or

high-energy Fourier components. To avoid this obstacle, Hellmann [40] and Fermi [41] proposed the concept of pseudopotentials (see the reviews by Chelikowsky [27] and Schwerdtfeger [28]). As mentioned earlier, pseudopotentials are obtained by assuming a nodeless pseudo-wave function that reproduces the same eigenvalue as the original all-electron wave function:

$$\left(-\frac{1}{2}\nabla^2 + V_{\text{ext}}^{\text{PS}} + V_{\text{H}}[n^{\text{PS}}] + V_{\text{xc}}[n^{\text{PS}}]\right)\phi_i^{\text{PS}}(\mathbf{r}) = \varepsilon_i \phi_i^{\text{PS}}(\mathbf{r}), \tag{2.29}$$

where $V_{\text{ext}}^{\text{PS}}$ is the unscreened pseudopotential or ion core pseudopotential, V_{ion}. In general, the ionic potential depends on the angular-momentum quantum number and we can express it as

$$\hat{V}_{\text{ion}} = \sum_l V_l \hat{P}_l, \tag{2.30}$$

where $\hat{P}_l$ is a projector for angular momentum quantum number l to project out the corresponding component from a wave function.

Kleinman and Bylander proposed a fully-separable form for the ionic potential operator [42]:

$$\hat{V}_{\text{ion}} = \sum_l V_l \hat{P}_l = V_{\text{ion,local}} + \hat{V}_{\text{ion,nonlocal}}. \tag{2.31}$$

The ionic potential operator is called separable because the dependence of two spatial points in the nonlocal part is disconnected, *i.e.*, $\hat{V}_{\text{ion,nonlocal}}(\mathbf{r}, \mathbf{r}') =$

$\hat{P}_l(\mathbf{r})\Delta V_l\hat{P}_l(\mathbf{r}')$ [43]. If we apply the operator to a wave function, we will have

$$\hat{V}_{\text{ion}}\phi_i(\mathbf{r}) = V_{\text{ion,local}}(\mathbf{r})\phi_i(\mathbf{r}) + \hat{V}_{\text{ion,nonlocal}}\phi_i, \tag{2.32}$$

where the nonlocal part is written as

$$\hat{V}_{\text{ion,nonlocal}}\phi_i = \sum_{l,m} G_{lm}u_{lm}(\mathbf{r})\Delta V_l(\mathbf{r}), \tag{2.33}$$

where $u_{lm}(\mathbf{r})$ is the pseudo-wave function for angular momentum quantum number l and magnetic quantum number m, $\Delta V_l(\mathbf{r}) = V_l(\mathbf{r}) - V_{\text{ion,local}}(\mathbf{r})$, and

$$G_{lm} = \frac{\int d\mathbf{r}' u_{lm}(\mathbf{r}')\Delta V_l(\mathbf{r}')\phi_i(\mathbf{r}')}{\int d\mathbf{r}'' u_{lm}(\mathbf{r}'')\Delta V_l(\mathbf{r}'')u_{lm}(\mathbf{r}'')}. \tag{2.34}$$

Because ΔV_l is short-ranged, we only need to compute the contribution from the grid points near the nuclear center.

Pseudopotential theory enables us to focus on valence electrons when we solve the Kohn–Sham equation. Consider $Si_{29}H_{36}$ as an example. There are 442 electrons. If we are to solve the Schrödinger equation, the ground-state solution will be a function of 1326 variables. And if we resort to DFT, in theory the solution (the charge density) will be a function of three variables. However, we do not know the universal functional yet. If we use the Kohn–Sham DFT method, we will need to solve for the Kohn–Sham orbitals. For 442 electrons we need the 221 lowest-energy Kohn–Sham orbitals (each being a function of three variables), assuming spin-unpolarized calculations (*i.e.*,

each orbital can accommodate two electrons).

If we further use the pseudopotential theory, we would need to solve for valence states only. That is, states emerging from the 3s and 3p states of silicon atoms and the 1s state of hydrogen atoms. A silicon atom has four valence electrons while a hydrogen atom has one. As a result, for $Si_{29}H_{36}$ there are 152 electrons, and we are solving for the 76 lowest-energy Kohn–Sham orbitals, assuming we are performing spin-unpolarized calculations. We can see in this small case the number of electrons we need to consider is already reduced by a factor three.

Not only the reduced number of electrons to consider, but there are also numerical advantages. The pseudo-wave functions are nodeless. They are smooth and extended states (*i.e.*, they are not highly localized core states). All of these make it less computationally demanding and possible to describe wave functions accurately with a modest basis.

2.4 Finite-Difference Discretization

When discretized on a cubic real-space grid using a higher-order finite-difference method, the Kohn–Sham equation can be expressed as [18]:

$$-\frac{\hbar^2}{2m}\left[\sum_{n_1=-N}^{N} C_{n_1}\phi_l(x_i+n_1h,y_j,z_k)+\sum_{n_2=-N}^{N} C_{n_2}\phi_l(x_i,y_j+n_2h,z_k)+\right.$$
$$\left.\sum_{n_3=-N}^{N} C_{n_3}\phi_l(x_i,y_j,z_k+n_3h)\right]+$$
$$\left[\hat{V}_{\text{ion}}(x_i,y_j,z_k)+V_{\text{H}}(x_i,y_j,z_k)+V_{\text{xc}}(x_i,y_j,z_k)\right]\phi_l(x_i,y_j,z_k)=$$
$$\varepsilon_l\phi_l(x_i,y_j,z_k). \tag{2.35}$$

In the above equation, $\hat{V}_{\text{ion}}(x_i,y_j,z_k)$ is the ionic pseudopotential applied to the valence electrons. $\hat{V}_{\text{ion}}$ can be divided into a local part, $V_{\text{ion,local}}$, plus a nonlocal part, $\hat{V}_{\text{ion,nonlocal}}$, in the Kleimann–Bylander form [42]. $V_{\text{H}}(x_i,y_j,z_k)$ is the Hartree potential, $V_{\text{xc}}(x_i,y_j,z_k)$ is the exchange-correlation potential, h is the grid spacing, and (ϕ_l,ε_l) is the l^{th} eigenpair of this eigenvalue problem. We seek the lowest-energy eigenpairs with $l=1,2,...,N_{\text{occ}}$, where N_{occ} is the highest occupied state.

When H_{KS} is discretized by a real-space method such as finite difference, the discretized H_{KS} is extremely sparse. The dimension of the discretized Hamiltonian is much larger than the number of eigenstates N_s to be computed. Therefore, an iterative method is preferred to compute the desired eigenstates.

2.5 Chebyshev-Filtered Subspace Iteration Method

2.5.1 Polynomial Filtering

There are a number of iterative methods for solving large sparse eigenvalue problems. In the early days, The initial version of PARSEC [19], a real-space pseudopotential DFT code, used the implicitly restarted Lanczos (IRL) algorithm implemented in the ARPACK software [44]. Although the method is robust and accurate, its computational cost increases rapidly with respect to the number of eigenstates computed. This is partly due to the fact that more iterations and restarts are needed to converge eigenvalues deep inside the spectrum, *e.g.*, near the Fermi level. Moreover, IRL can only make use of a good initial guess to a single eigenvector. This feature makes it less effective for subsequent SCF iterations in which a good initial guess to the entire subspace associated with the occupied states is available. Furthermore, IRL is not a block method, *i.e.*, in IRL the Hamiltonian can be multiplied with only one vector at a time. This makes it less scalable on a parallel machine.

Both the Locally Optimal Block Preconditioned Conjugate Gradient (LOBPCG) [45] and Davidson algorithms [46] are block algorithms that can take advantage of a good initial guess of the subspace associated with the occupied states. However, without a good preconditioner, these methods tend to converge slowly. For real-space methods, it is generally difficult to construct an effective preconditioner. Thus these methods are less effective. Furthermore,

these methods are less stable when a large number of eigenpairs are to be computed [47].

The use of polynomial filtering for computing eigenvalues of a large sparse matrix dates back over 50 years to the work of Stiefel [48] and was carefully examined by Yang [49]. The use of Chebyshev filtering for solving the Kohn–Sham eigenvalue problem was first developed by Zhou *et al.* [50], where effective spectral bounds were developed to construct an effective Chebyshev filtering polynomial. The CheFSI was used to compute the invariant subspace associated with the occupied states instead of eigenvectors explicitly. The success of Zhou *et al.* brought a speed-up of five to ten times compared to other eigensolvers, and marked the beginning of the prevalence of the CheFSI method in electronic structure calculations.

We review the CheFSI method in Section 2.5.2, and briefly discuss how to construct a bandpass polynomial filter to amplify eigencomponents associated with interior eigenvalues in Section 2.6.1. We explain why it is a good idea to combine spectrum slicing using bandpass polynomials with CheFSI in a hybrid polynomial filtering algorithm for accelerating the SCF iteration in Section 2.6.2.

2.5.2 Chebyshev-Filtered Subspace Iteration

An mth-degree Chebyshev polynomial of the first kind can be defined recursively as

$$T_m(t) = 2tT_{m-1}(t) - T_{m-2}(t), \tag{2.36}$$

with $T_0(t) = 1$ and $T_1(t) = t$. The magnitude of $T_m(t)$ is bounded by 1 within $[-1, 1]$ and grows rapidly outside of this interval. Figure 2.2 shows Chebyshev polynomials of various degrees.

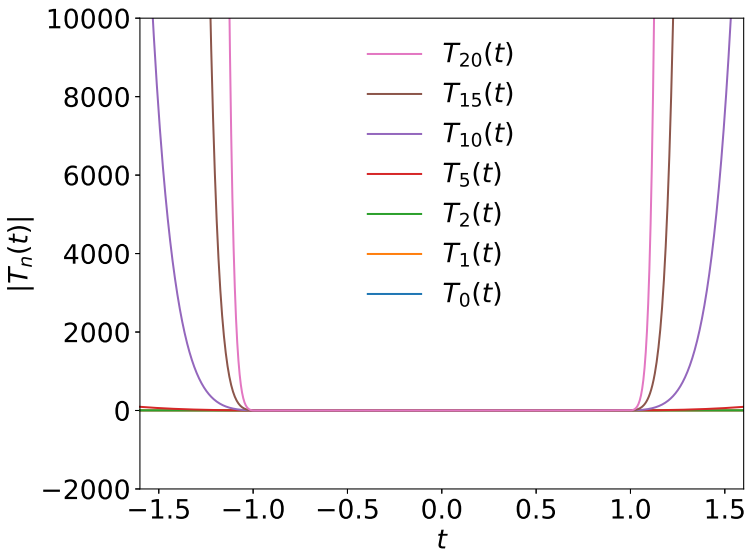

Figure 2.2: Chebyshev polynomials of the first kind.

We define the estimated lower and upper bounds of the spectrum of H, λ_{lb} and λ_{ub}, and an estimated Fermi level, ε_{F}. By mapping the unwanted eigenvalues (*e.g.*, the unoccupied states) of the Kohn–Sham Hamiltonian H enclosed by $[\varepsilon_{\text{F}}, \lambda_{\text{ub}}]$ to $[-1, 1]$ through the linear transformation $(t - c)/e$, where $c = (\varepsilon_{\text{F}} + \lambda_{\text{ub}})/2$ and $e = (\lambda_{\text{ub}} - \varepsilon_{\text{F}})/2$, we can use $\hat{T}_m(H) = T_m((H - cI)/e)v$ to amplify the eigenvector components in v that correspond to eigenvalues outside of $[\varepsilon_{\text{F}}, \lambda_{\text{ub}}]$. Applying $T_m((H - cI)/e)$ repeatedly to a block of vectors V filters out the eigenvectors associated with eigenvalues in $[\varepsilon_{\text{F}}, \lambda_{\text{ub}}]$. The desired eigenpairs can be obtained through the standard

Rayleigh–Ritz procedure [51] described in Algorithm 3.

Owing to the three-term recurrence in (2.36), $W = \hat{T}_m(H)V$ can be computed recursively without forming $\hat{T}_m(H)$ explicitly in advance. Algorithm 1 describes how this step is carried out in detail. To maintain numerical stability, we orthonormalize vectors in W. The orthonormalization can be performed by a (modified) Gram–Schmidt process or by a Householder transformation based QR factorization [52].

Although these algorithms can achieve high accuracy, they are not the most efficient ones, especially when implemented on a distributed, parallel computer. We choose the Cholesky QR algorithm described in Algorithm 2 to perform orthonormalization because Algorithm 2 can make use of highly efficient matrix computation kernels such as matrix–matrix multiplication, Choleksy factorization, and triangular substitution with multiple right-hand sides. We perform a fixed number of Chebyshev polynomial filtering (Algorithm 1) and orthonormalization (Algorithm 2) steps before using Rayleigh–Ritz procedure to extract approximations to the desired eigenvalues and eigenvectors. The overall Chebyshev-filtered subspace iteration procedure is described in Algorithm 4.

In Algorithm 1, the required inputs are λ_{lb}, λ_{ub}, ε_{F}, and a filter degree, m. In the first SCF iteration, λ_{lb} and λ_{ub} can be calculated by running a few Lanczos iterations, and ε_{F} is typically set to $(2\lambda_{\text{lb}}+\lambda_{\text{ub}})/3$. In the subsequent SCF iterations, λ_{lb} can be replaced by the lowest Ritz value while ε_{F} can be computed by solving an equation of the conservation of valence electrons,

i.e., $N_\mathrm{e} = \int n(\mathbf{r})\mathrm{d}\mathbf{r}$, where N_e is the total number of valence electrons. See the work of Saad [53] and Zhou *et al.* [31] for more details on Chebyshev filtering.

We note that for systems with a relatively large gap between the occupied and unoccupied states (*e.g.*, insulators), it may not be necessary to perform the Rayleigh–Ritz procedure because the basis vectors ϕ_i's used in (2.24) can be any orthonormal basis vectors that span the same invariant subspace defined by the occupied states. For gapless systems (metals) or systems with a relatively small energy gap, approximate eigenvalues near the gap are needed to obtain the occupancy defined in (2.25). Moreover, the use of the Rayleigh–Ritz procedure may also allow us to lock eigenvectors that have already converged to reduce the computational cost of subsequent subspace iterations.

Algorithm 1 Chebyshev filtering

procedure W = ChebyFilter(H, V, m, ε_F, λ_ub, λ_lb)
  $e = (\lambda_\mathrm{ub} - \varepsilon_\mathrm{F})/2$
  $c = (\lambda_\mathrm{ub} + \varepsilon_\mathrm{F})/2$
  $\sigma = e/(c - \lambda_\mathrm{lb})$
  $\tau = 2/\sigma$
  $W = (HV - cV)(\sigma/e)$
  for $i = 2 \rightarrow m$ **do**
    $\sigma_{new} = 1/(\tau - \sigma)$
    $W_t = (HV - cV)(2\sigma_\mathrm{new}/e) - (\sigma\sigma_\mathrm{new})V$
    $V = W$
    $W = W_t$
    $\sigma = \sigma_\mathrm{new}$

Algorithm 2 Cholesky QR

1: **procedure** $V = \text{ORTH}(W)$
2: $A = W^T W$ ▷ A is positive definite
3: Find an upper triangular R such that $A = R^T R$ ▷ Cholesky factorization
4: $V = WR^{-1}$

Algorithm 3 Rayleigh–Ritz procedure

1: **procedure** $(V,D) = \text{RAYLEIGHRITZ}(H, V)$
2: $A = V^T HV$
3: Compute Q and D such that $AQ = QD$ and $Q^T Q = I$. ▷ D has the Ritz values
4: $V = VQ$

Algorithm 4 Chebyshev-filtered subspace iteration

1: **procedure** $(V, D) = \text{CHEBYSUBIT}(H, V, m, \varepsilon_{\text{F}}, \lambda_{\text{ub}}, \lambda_{\text{lb}}, \text{maxiter})$
2: **for** iter $= 1 \rightarrow$ maxiter **do**
3: $W = \text{ChebyFilter}(H, V, m, \varepsilon_{\text{F}}, \lambda_{\text{ub}}, \lambda_{\text{lb}})$;
4: $V = \text{Orth}(W)$;
5: $(V,D) = \text{RayleighRitz}(H, V)$;

2.6 Polynomial Filtering Spectrum Slicing Method

Spectrum slicing (SS) methods have been proposed to address the issue from the cubic scaling operations (orthonormalization and Rayleigh–Ritz procedure). SS refers to dividing the spectrum of interest into several spectral slices and computing approximate eigenpairs within each slice simultaneously. Although SS methods can clearly reduce the overhead associated with solving the projected subspace problem (because each slice contains a much smaller subset of the eigenvalues), it needs to compute interior eigenvalues that may be clustered. A different type of polynomial filters needs to be constructed to amplify eigenvalues within each slice, just as a Chebyshev polynomial is used to amplify eigenvalues at the left end of the spectrum in the CheFSI method. The degree of these types of polynomials, which we will refer to as "bandpass filter polynomials," may be much higher than that of the Chebyshev polynomials used in CheFSI when the eigenvalues to be computed are very close to each other.

Previously, a thick-restart Lanczos algorithm was proposed to compute interior eigenvalues filtered by bandpass filter polynomials [54]. The method converges fast, but has limited levels of concurrency. In this work, we propose using a subspace iteration method to compute interior eigenpairs. Although the subspace iteration method tends to have a slower convergence rate, especially at the beginning of a SCF calculation when no good initial guess to the approximate eigenpairs is available, it is a block method that exhibits

multiple levels of concurrency that lead to superior parallel scalability.

To address the potential slow convergence of the subspace iteration method, we propose using CheFSI in the first few SCF iterations and then transitioning to SS based polynomial filtering in the subsequent SCF iterations. We show by numerical examples that this hybrid polynomial filtering scheme is effective.

One of the keys to achieving scalable performance in SS is an optimal partition of the desired part of a spectrum. The partition must take into account the distribution of eigenvalues (which can be obtained from either an estimated density of states or approximated eigenvalues computed in a previous SCF iteration) as well as load balance needs in a parallel implementation. We present a practical way to perform such a partitioning.

2.6.1 Spectrum Slicing

When the number of eigenpairs to be computed (usually the occupied states plus some unoccupied states), N_s, is relatively small (*e.g.*, less than a thousand), the computational cost of CheFSI is dominated by sparse matrix-vector multiplications HV. By making these multiplications scalable through the distribution of vectors in V on a 2D process grid, we can compute the desired eigenpairs efficiently. However, as the system size increases, the number of eigenpairs N_s becomes very large. As a result, orthonormalization and Rayleigh–Ritz procedure, which scale cubically with respect to N_s, start to dominate the computational cost. Furthermore, because these types of

computation involve more collective communication among all processes, the overall computation is difficult to scale to a large number of compute cores.

To address this difficulty, an SS algorithm was proposed by Schofield *et al.* [55] to divide the eigenvalues to be computed into several subintervals (*i.e.*, slices) that can be examined simultaneously. Eigenvalues within each subinterval can still be computed by a polynomial filtered iterative method. However, with the exception of the leftmost subinterval, a different type of filter needs to be constructed to focus on eigenvalues within the other subintervals.

Schofield *et al.* [55] and Li *et al.* [54] implemented methods for constructing polynomial filters $p_m(t)$ for computing eigenvalues within an interval $[a, b]$. A thick-restart Lanczos iteration is used to compute the largest eigenpairs of $p_m(H)$. Although these studies examined several key issues of the SS algorithm, and provided a proof of concept, a few important details such as how to partition the spectrum to achieve scalable performance on a large number of compute cores were not addressed.

Here we examine several implementation details that are key to achieving scalable performance. Instead of using a thick-restarted Lanczos iteration method, which cannot take full advantage of a 2D process grid to compute eigenpairs within each interval, we propose to use a subspace iteration method to compute the largest eigenpairs of $p_m(H)$. However, because the subspace iteration method typically has a slower convergence rate, a high degree polynomial or a large number of iterations may be required.

Spectrum Partition

Although SS is conceptually easy to understand, there are a number of issues that we must address in order to make it a practical computational method. The first pertains to the partition of a spectrum into distinct slices that can be examined in parallel. Because we often do not know how the eigenvalues are distributed in advance of the solution, this is not a trivial task.

One way to estimate the distribution of eigenvalues is to use a Lanczos algorithm [56]. We use an ensemble of ℓ randomly generated vectors $v^{(j)}$, $j = 1, 2, ...\ell$, with Gaussian independent and identically distributed elements to start ℓ k-step Lanczos iterations, which produce ℓ $k \times k$ tridiagonal matrices $T^{(j)}$, $j = 1, 2, ..., \ell$. If the eigenvalues of $T^{(j)}$ are denoted by $\theta_i^{(j)}$, $i = 1, 2, ..., k$, and the first component of the eigenvector associated with $\theta_i^{(j)}$ is denoted by $\tau_i^{(j)}$, an approximation to the density of states can be expressed as

$$d(t) = \sum_{j=1}^{\ell} \sum_{i=1}^{k} (\tau_i^{(j)})^2 \exp\left(-\frac{(t - \theta_i^{(j)})^2}{\sigma_i^{(j)}} \right), \tag{2.37}$$

where $\sigma_i^{(j)}$'s are scaling parameters that should be chosen carefully [56]. Note that the ℓ Lanczos runs can be carried out in parallel on different process groups.

Although the function $d(t)$ describes the distribution of eigenvalues, the value $d(t)$ for a specific t is not so useful. It is more useful to work with the

cumulative density of states defined as

$$g(t) \equiv \int_{-\infty}^{t} d(s)\mathrm{d}s = \frac{\sqrt{\pi}}{2} \sum_{j=1}^{\ell} \sum_{i=1}^{k} (\tau_i^{(j)})^2 \sigma_i^{(j)} \left[\mathrm{erf}\left(\frac{t - \theta_i^{(j)}}{\sigma_i} \right) + 1 \right]. \quad (2.38)$$

In particular, given two real numbers t_1 and t_2 with $t_1 < t_2$, $g(t_2) - g(t_1)$ yields the number of eigenvalues within the interval $[t_1, t_2)$.

The strategy we use to partition the spectrum of interest is to divide the interval $[a, b]$, where a is a lower bound of the eigenvalues of H and b is an (estimated) upper bound of the occupied states, uniformly into p slices of equal size $[l_i, u_i)$, $i = 1, 2, ..., p$, where $l_1 = a$, $u_p = b$ and $l_i = u_{i-1}$ for $i > 1$, $u_i = l_{i+1}$ for $i < p$. We call this partition scheme a *uniform partition*. The optimal number of slices p depends on a number of factors, which we will discuss later in Section 2.6.3. The uniform partition will also need to be modified to achieve good load balance in a parallel implementation.

In order to avoid missing eigenpairs in each slice $[l_i, u_i)$, we construct the polynomial on a slightly larger interval $[l_i - \delta, u_i + \delta]$, where δ is typically chosen to be a fraction of the width of a slice. (We use $\delta = (u_i - l_i)/10$ in our simulations.)

We use $c_i = g(u_i + \delta) - g(l_i - \delta)$ to estimate the number of eigenvalues within that augmented interval. Note that some of the eigenvalues within $[l_i - \delta, u_i + \delta)$ overlap with those in $[l_{i-1} - \delta, u_{i-1} + \delta)$ and $[l_{i+1} - \delta, u_{i+1} + \delta)$, as shown in Figure 2.3. Once the eigenvalues in each slice are computed, we use the slice bounds l_i and u_i to select eigenvalues and eigenvectors to be included

in the occupancy and electronic charge density calculation. An eigenvalue in the overlap regions is possibly captured by both adjacent slices, but will be counted by only one of the two adjacent slices when the SCF calculation approaches convergence. The overlaps between adjacent augmented slices allow us to eliminate the need to orthogonalize approximate eigenvectors computed in different slices, which requires additional data communication.

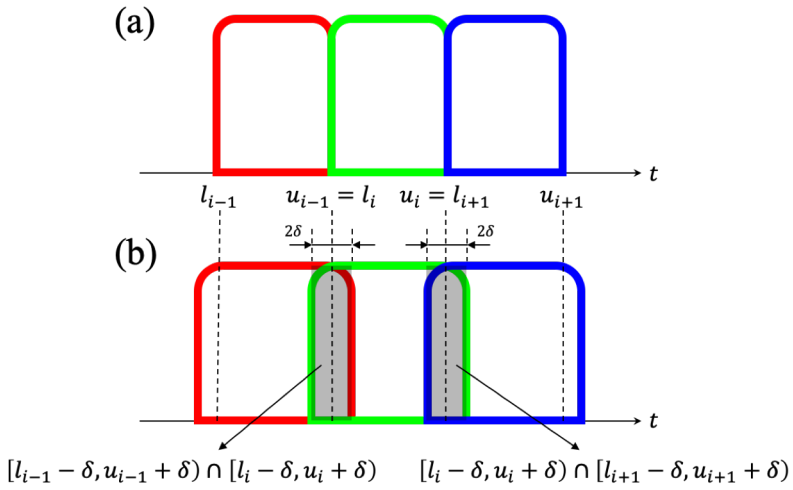

Figure 2.3: (a) An illustration of uniform partitioning. (b) Extension of the bounds of a filter by δ in both directions, and overlaps (shaded regions) of 2δ between adjacent slices.

Based on the eigenvalue count c_i and the degree of the bandpass filter used to amplify eigenvalues within the ith slice, we assign an appropriate number of processes to each slice to balance the computational load among different processes. We will discuss the load balance details in Section 2.6.3.

An alternative way to partition a spectrum is to determine values l_i and u_i such that $g(u_i) - g(l_i)$ are approximately the same for all i's. This approach

will yield slices of different spectral widths even though each slice contains nearly the same number of eigenvalues. The potential difficulty with this approach is that bandpass filters of drastically different degrees may be needed for different slices (see the next section), thereby making load balancing more difficult to achieve.

We note that the use of the Lanczos-based density of states estimation to partition the spectrum is only needed in the first few SCF iterations. When the Ritz values computed from each spectral slice are sufficiently accurate, we can use them to refine the spectrum partition. Such a partition will not change much in subsequent SCF iterations.

Filter Polynomial Construction

The bandpass filter polynomials we construct can have a large impact on the convergence of the SS algorithm. An ideal filter $F(t)$ should map the eigenvalues within a spectral slice $[l, u]$ to a much larger value than the values of F outside of this interval (but within $[\lambda_{\min}, \lambda_{\max}]$. If the interval $[l, u]$ is sufficiently large, the bandpass filters proposed by Schofield *et al.* [55] generally works well. These bandpass filter functions are expanded in terms of Chebyshev polynomials. To simplify the discussion, let us assume the eigenvalues of H are within $[-1, 1]$, and we are interested in the eigenvalues within an interval $[l, u]$ with $-1 < l < u < 1$. With $\alpha_k = \frac{\pi}{k+2}$, the following polynomial filter

$$F(t) \approx p(t) = \sum_{i=0}^{k} \gamma_i g_i^k T_i(t), \tag{2.39}$$

where

$$\gamma_i = \begin{cases} \frac{1}{\pi}(\cos^{-1}(l) - \cos^{-1}(u)) & \text{as } \ i = 0, \\ \frac{2}{\pi}(\frac{\sin\left(i\cos^{-1}(l)\right) - \sin\left(i\cos^{-1}(u)\right)}{i}) & \text{as } \ i > 0, \end{cases} \tag{2.40}$$

and

$$g_i^k = \frac{(1 - \frac{i}{k+2})\sin(\alpha_k)\cos(i\alpha_k) + \frac{1}{k+2}\cos(\alpha_k)\sin(i\alpha_k)}{\sin(\alpha_k)}, \tag{2.41}$$

amplifies $\lambda \in (l, u)$.

Figure 2.4 shows four bandpass filters of different degrees defined on the interval $[-0.8, -0.7]$. Note that in practice we will use $l_i - \delta$ and $u_i + \delta$ as bounds in polynomial filtering but select the converged eigenpairs based on l_i and u_i, as explained in Section 2.6.1.

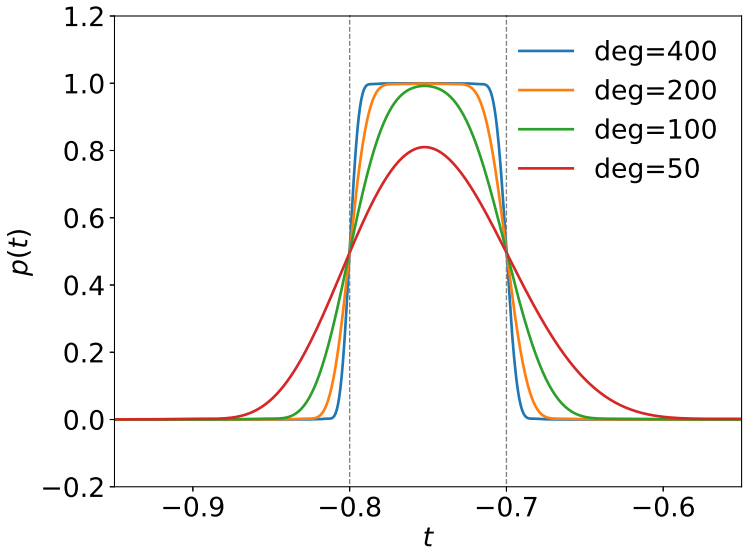

Figure 2.4: Bandpass polynomial filters of different degrees. The dashed lines indicate the lower and upper bounds of the slice.

If $[l, u]$ is relatively small compared with the spectrum width of H, a polynomial filter constructed from a least squares approximation to the Dirac-δ distribution (function), which was proposed by Li *et al.* [54], works well. Such a polynomial can be expressed as

$$F_k(t) = \sum_{j=0}^{k} \mu_j T_j(t), \tag{2.42}$$

with

$$\mu_j = \begin{cases} \frac{1}{2} & \text{as } j = 0, \\ \cos(j \cos^{-1}(\gamma)) & \text{otherwise}, \end{cases} \tag{2.43}$$

where γ is the center of the Dirac-δ function. By specifying the value of $F(t)$ at the bounds, we can determine the degree of the polynomial.

We should note that a Chebyshev polynomial can be used as the filter for the leftmost slice. Because the degree of the Chebyshev polynomial required to amplify the eigenvalues in the leftmost slice is typically much lower than those of the polynomials constructed for other interior slices, we may include more eigenvalues in the leftmost slice in a parallel implementation of the SS algorithm to achieve good load balance. We will discuss this in Section 2.6.3.

Lanczos vs Subspace Iteration

In the previous work by Schofield *et al.* [55] and Li *et al.* [54], polynomial filtering is combined with a thick-restarted Lanczos algorithm to generate a Krylov subspace of the form

$$\mathcal{K}(F(H), v_0) = \{v_0, F(H)v_0, F^2(H)v_0, ...\}, \tag{2.44}$$

where v_0 is typically chosen to be a random starting vector. Rayleigh–Ritz procedure is then performed on the orthonormal basis V of $\mathcal{K}(F(H), v_0)$ produced by the Lanczos algorithm. Because the eigenpairs of interest are those associated with the largest eigenvalues of $F(H)$, which are known to emerge rapidly [57], this approach can produce highly accurate approximations after k iterations, where k is typically a small (*e.g.*, 2 to 3) multiple of the number of eigenvalues within the the interval of interest. This observation was

reported by Schofield *et al.* [55]. However, the main drawback of the Lanczos algorithm is that only one matrix-vector multiplication can be applied at a time. Because $F(H)$ cannot be applied simultaneously to a number of vectors, this procedure prohibits us from using a 2D process grid employed in the parallel implementation of CheFSI.

We propose to use a subspace iteration instead of the Lanczos algorithm to obtain approximations to the desired eigenpairs. The algorithm is nearly identical to Algorithm 4 except the filter used in line 3 of the algorithm is different. The convergence of subspace iteration is linear. The rate of convergence depends on the minimum of the ratio

$$\frac{F(\lambda_{\text{in}})}{F(\lambda_{\text{out}})},$$

where λ_{in} is an eigenvalue of H within the interval of interest $[a, b)$ and λ_{out} is an eigenvalue outside of $[a, b)$. When the ratio is small (close to 1), many iterations may be required to reach convergence. To make the ratio large, a high degree polynomial may be required, thereby increasing the cost per iteration. Thus, there is a tradeoff between the number of subspace iterations and the degree of filter polynomials. The optimal values depend on the size of the interval and the distribution of eigenvalues.

Another way to improve the convergence rate of subspace iteration is to make the subspace slightly larger than the number of eigenvalues within the target interval $[a, b)$. The enlarged subspace can include a fraction of

the states from adjacent slices (see Section 2.6.3). In this case, the effective convergence rate is defined by the minimum of the ratio

$$\frac{F(\lambda_{\text{in}})}{F(\lambda_{\text{out}'})},$$

where $\lambda_{\text{out}'}$ is outside of an interval $[a', b']$, with $a' < a$ and $b' > b$. The overlap intervals $[l_i - \delta, u_i + \delta]$ proposed in Section 2.6.1 allow us to achieve this. The exact values of a' and b' depend on the size of the spectral slice and the distribution of eigenvalues outside of that slice. We only take the Ritz values within $[a, b)$ and the corresponding harmonic Ritz vectors as the approximate eigenpairs of H. Other Ritz pairs obtained from the same space V approximate eigenpairs in other intervals. Therefore, Ritz pairs computed in different slices have some overlap. This overlap allows us to eliminate the need to move vectors across different slices as the Ritz values change throughout the SCF iterations.

As the Hamiltonian H changes over the SCF iterations, a Ritz value $\theta^{(i)} \in [a, b)$ obtained in the ith SCF iteration may move outside of that interval in the next iteration, *i.e.*, $\theta^{(i+1)}$ may move into an adjacent interval. This type of shift tends to occur more often in the first few SCF iterations, when the electronic charge density is far from converged. By making the dimension of the subspace larger than the number of eigenvalues within $[a, b)$, we avoid the need to move the harmonic Ritz vector z associated with $\theta^{(i+1)}$ explicitly to a different slice which is typically mapped to a different process group.

An explicit move would require additional communication. A harmonic Ritz vector in another interval whose corresponding Ritz value was previously outside of that interval but lands in that interval in the $(i+1)$th iteration, may take the place of z. Multiple approximations to the same eigenpair obtained from different slices can be easily sorted out by simply using the bounds of the intervals.

We should note that in most SCF iterations, a subspace iteration is started from previous approximations to the eigenpairs of interest. The subspace iteration method does not need to produce highly accurate approximations to the desired eigenpairs until the SCF calculation is near convergence. Therefore, the number of subspace iterations in a SCF iteration can be set to a modest number (*e.g.*, between 1 and 5).

We note that for interior eigenvalues, the standard Rayleigh–Ritz procedure can produce some spurious eigenpairs as indicated by relatively large residuals [58]. Although including more basis vectors in a subspace can improve the accuracy of the Ritz approximation, in general, the standard Rayleigh–Ritz procedure cannot completely eliminate the presence of spurious Ritz approximations to interior eigenpairs [55]. However, it is well known that the harmonic Rayleigh–Ritz procedure can be used to address this issue [59]. The harmonic Rayleigh–Ritz procedure imposes the condition that

$$(H-\sigma I)^{-1}v - (\tilde{\theta}-\sigma)^{-1}v \perp (H-\sigma I)^2\mathcal{V}, \tag{2.45}$$

where the harmonic Ritz vector v lies in the subspace $\mathcal{V}$, σ is the center of a spectral slice, and $\tilde{\theta}$ is the corresponding harmonic Ritz value. If V forms an orthonormal basis of $\mathcal{V}$, and $v = Vc$ for some c, v and $\tilde{\theta}$ can be obtained by solving the generalized eigenvalue problem

$$V^T(H - \sigma I)Vc = \xi V^T(H - \sigma I)^2 Vc, \tag{2.46}$$

where $\xi = (\tilde{\theta} - \sigma)^{-1}$. For completeness, we outline the harmonic Rayleigh–Ritz procedure in Algorithm 5. The eigenvalue approximations θ_i's are computed as $\theta_i \approx \tilde{\theta}_i = {\xi_i}^{-1} + \sigma$.

Algorithm 5 Harmonic Rayleigh–Ritz

1: **procedure** (V,D) = HarmonicRayleighRitz(H, V, σ)
2: $A = V^T(H - \sigma)V$
3: $B = V^T(H - \sigma)^2 V$
4: Compute Q, $\tilde{D}$ such that $AQ = BQ\tilde{D}$ and $Q^TQ = I$.
5: $V = VQ$

Convergence Monitoring and Termination Criteria

To ensure the convergence of a SCF calculation, which is monitored by examining charge-weighted self-consistent residual error (SRE), defined as

$$\text{SRE} = \sqrt{\frac{1}{N_\text{e}} \int d\mathbf{r} n(\mathbf{r})[V_\text{tot}^{(i)}(\mathbf{r}) - V_\text{tot}^{(i-1)}(\mathbf{r})]^2}, \tag{2.47}$$

where $n(\mathbf{r})$ is the electronic charge density, $V_\text{tot}^{(i)}(\mathbf{r})$ is the total potential after the ith SCF iteration, and N_e is the total number of valence electrons, we

need to make sure that the approximate invariant subspace returned from the polynomial filtered subspace iteration is sufficiently accurate over SCF iterations. Although the accuracy of the approximate invariant subspace can be measured by the residual

$$R = HV - V\Theta,$$

where V contains the Ritz or harmonic Ritz vectors and the diagonal matrix Θ contains the Ritz values on its diagonal, such a measure is likely to be conservative in early SCF iterations where the SRE is likely to be large. In these early SCF iterations, we can also adopt an alternative termination criterion that simply counts the number of Ritz values that falls within a perturbed spectral bounds of each slice, where the size of the perturbation is related to the residual norm of each Ritz pair. We could therefore terminate the subspace iterations when the counts no longer change. Although the question of how to set the convergence tolerance in an optimal fashion remains open, from our experience, an approximate tolerance commensurate with the magnitude of SRE can be used in early SCF iterations. The tolerance can be gradually tightened in subsequent SCF iterations as SRE becomes smaller.

2.6.2 Hybrid Polynomial Filtering

Owing to the relatively slow convergence of the subspace iteration method used in the SS algorithm for computing eigenvalues within each slice, apply-

ing the SS algorithm directly to a random guess of the invariant subspace associated with the desired eigenvalues can be computationally complex. Either an extremely high degree polynomial or many subspace iterations may be required to obtain an approximation to the invariant subspace of interest with sufficient accuracy in the first few SCF iterations. In both cases, the large number of multiplications of H with vectors in subspace iterations may offset the reduction in calculation cost of orthonormalization and Rayleigh–Ritz procedure.

When a good initial guess of the invariant subspace associated with the eigenvalues within a slice is available, a bandpass-filtered subspace iteration can be effectively used to refine the approximation to the desired invariant subspace.

Based on this observation, we combine CheFSI with SS to devise a hybrid polynomial filtering method for SCF calculations. In the first few SCF iterations, CheFSI is used to obtain a reasonably accurate approximation to the desired invariant subspace of a converging sequence of Kohn–Sham Hamiltonians. In the subsequent SCF iterations, the SS method that utilizes bandpass filter polynomials is applied to the approximate subspace produced in previous SCF iterations.

2.6.3 Parallel Implementation

Although the hybrid filtering method that combines CheFSI and SS is conceptually easy to understand, an efficient implementation on a large-scale,

distributed memory, parallel computing platform is nontrivial. The implementation requires a careful data layout on a 2D computational process grid to increase concurrency and reduce communication overhead. To achieve good load balance in SS, the spectrum partition must take into account both the number of eigenvalues to be computed and the degree of the bandpass filter used to amplify eigenvalues within a slice, which may vary from slice to slice.

Data layout

To implement Algorithm 4 efficiently on a distributed memory parallel computer, we need to develop a data distribution scheme to perform both polynomial filtering and orthonormalization in parallel. We employ a 2D process grid $n_{\mathrm{r}} \times n_{\mathrm{c}}$. By partitioning vectors in V into n_{c} groups and mapping each group to a distinct column group in the 2D process grid (Figure 2.5), we can compute several column groups of $\hat{T}(H)V$ independently with essentially no communication between different column groups (shaded regions in Figure 2.5). Within each column group, matrix-vector multiplication $w = Hv$ can be performed in parallel by partitioning and distributing w and v by rows and mapping each row block onto a distinct row process in each column group. The kinetic energy part of H, which can be represented by a finite difference stencil, is not explicitly stored. Both the pseudopotential and the LDA/GGA exchange-correlation potential can be partitioned conformally with the partition of the vectors. Only nearest neighbor communication is

needed to combine local computation.

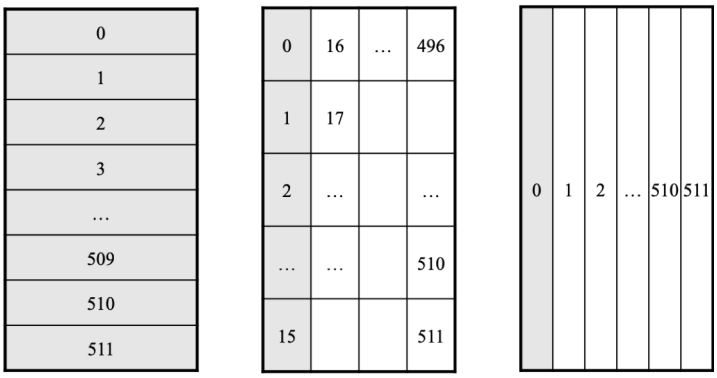

Figure 2.5: Different process grids with 512 MPI processes. The shaded regions denote the first column group.

A 2D process grid can also be used to perform the matrix-matrix multiplication $A = V^T V$ required in the Cholesky QR algorithm by calling the PDGEMM subroutine in PBLAS [60].

Both the Cholesky factorization and the subspace diagonalization used in Rayleigh–Ritz procedure may be performed among a subset of the processes if the total number of processes is large.

The optimal choice of n_r and n_c depends on the size of a problem and the number of available processes. Polynomial filtering favors a large n_c because the multiplication of H with vectors mapped to different column groups can be performed simultaneously with no communication. On the other hand, the dense matrix-matrix multiplication used for inner product calculation $V^T V$ favors a large n_r, especially when the leading dimension of V is large.

Switching from one 2D process grid to another (using the ScaLAPACK

utility PDGEMR2D) can be beneficial when different types of operations are performed. The overhead associated with such a switch is relatively small.

For SS, the column groups are further partitioned into supergroups according to a slice partitioning scheme, *i.e.*, each slice is assigned a set of column groups. For example, Figure 2.6 shows that eight column groups are divided into three supergroups, each associated with a spectral slice. The first spectral slice uses only one column group (highlighted in red), whereas the second and third use four (highlighted in green) and three column groups (highlighted in blue) respectively. Each bandpass-filtered subspace iteration is carried out on a 2D process grid with $n_r \times n_{c_i}$ processes, where n_{c_i} is the number of column groups assigned to the ith slice, and $\sum_i n_{c_i} = n_c$. Unlike polynomial filtering, which is done within each column group, orthonormalization and Rayleigh–Ritz procedure require communication between column groups and are done within each slice independently.

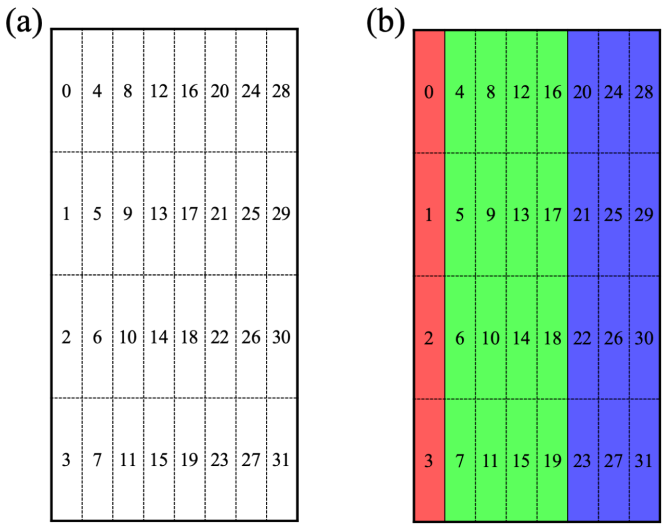

Figure 2.6: (a) A 2D process grid with $n_r \times n_c = 4 \times 8$. (b) A slice partition of the eight column groups into three slices. In this example, slices 1, 2, and 3 have 1, 4, and 3 column groups, respectively.

Load balance

To achieve good load balance in a parallel implementation of CheFSI, we need to divide N_s evenly among different column groups and partition V according to this division.

Balancing computational load for SS is a bit more complicated. The uniform spectrum partition scheme discussed in Section 2.6.1 suggests that different slices may contain a different number of eigenvalues. To achieve good load balance, we proportionally map more processes to slices that have more eigenvalues. This is achieved by partitioning column groups according to the number of eigenvalues in each slice. In addition, because the degrees of the polynomials used for different slices may be different, the partitioning

of column groups needs to take the degrees of the polynomials into account as well. In particular, because the degree of the Chebyshev polynomial used for the leftmost slice is generally much lower than that of a bandpass filter polynomial used for an interior slice, fewer column groups should be assigned to the leftmost slice.

Furthermore, the degrees of bandpass filters associated with different interior slices of equal sizes may be slightly different as well. This is because the amplification effect of a bandpass filter for a particular slice depends on the location of the slice within the spectrum region of interest [55]. Figure 2.7 shows four filters, and the three on the right are bandpass filters of the same degree for different interior slices of the same size. We can see that the filter associated with the rightmost slice is flatter and its maximum within the slice is lower than that associated with the second slice from the left.

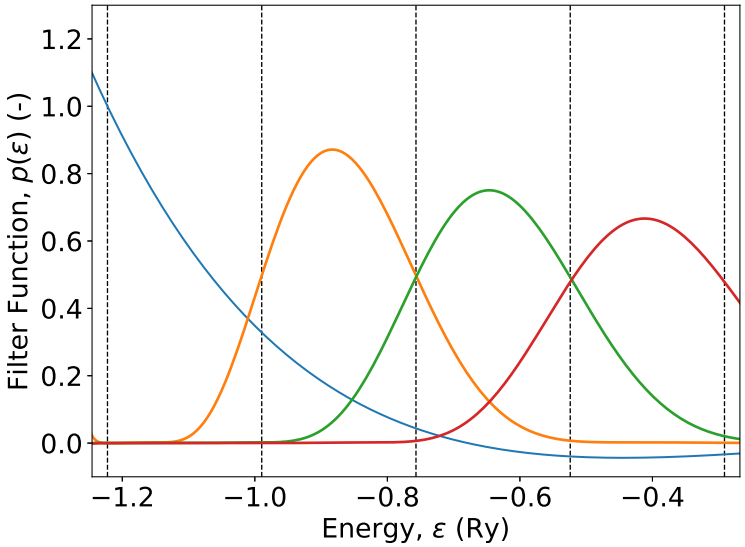

Figure 2.7: Four filters used in the simulation of $Si_{1947}H_{604}$. The degree of the Chebyshev polynomial (for the leftmost slice) is 20, while the degrees of the bandpass filters for the interior slices are 160. The dashed lines visualize the slice bounds.

Figure 2.8 shows that, in order to achieve the same amplification effect, the degrees of the bandpass filters for different slices may be slightly different, with the rightmost slice having a higher degree than the others. We note here that one method to estimate the necessary degree of a bandpass filter has been proposed by Schofield *et al.* [55], which requires a user-specified amplifying ratio, r_{amp}, and demands that both $\frac{p(l_i)}{p(l_i-\delta)} \geq r_{\text{amp}}$ and $\frac{p(u_i)}{p(u_i+\delta)} \geq r_{\text{amp}}$ are satisfied.

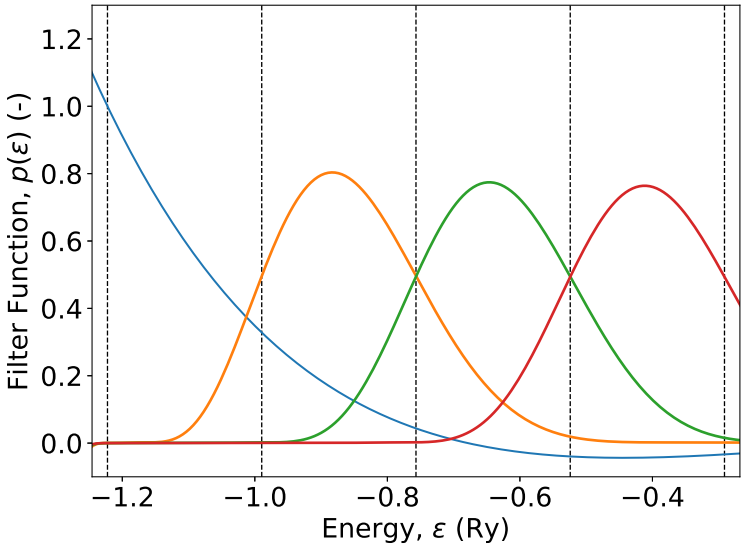

Figure 2.8: Four filters with the same slice bounds as those in Figure 2.7 but of different polynomial degrees. The degree of the Chebyshev polynomial (for the leftmost slice) is 20, while the degrees of the bandpass filters for slices 2, 3, and 4 are 137, 168, and 195, respectively. The amplifying ratio is 1.1. The dashed lines visualize the slice bounds.

Once a partition of the spectrum into multiple slices has been determined, the number of column groups assigned to the ith slice (n_{c_i}) can be set according to the following formula

$$n_{c_i} = \left\lfloor n_{\mathrm{c}} \cdot \frac{p_i m_i}{\sum_{i=1}^{s} p_i m_i} \right\rfloor, \tag{2.48}$$

where p_i is the number of eigenvalues in the ith slice, m_i is the degree of the polynomial constructed for that slice, s is the total number of slices, and n_{c} is the total number of column groups defined in Section 2.6.3. Some adjustment may be needed to ensure that n_{c_i}'s add up to n_c. In practice, we

use a larger p_i, which is the number of eigenvalues in that slice plus some states from adjacent slices. These extra states may originate from as many as two column groups in each of the adjacent slices. This strategy improves the stability of the SS algorithm and prevents the potential necessity of moving overlap states between slices over SCF iterations.

One of the challenges in implementing SS is to choose an optimal partition so that

$$\max_i p_i m_i \tag{2.49}$$

is minimized. Because the degree of the Chebyshev polynomial used for the leftmost slice is generally much lower, it makes sense to make the leftmost slice slightly wider to include more eigenvalues. To minimize (2.49), we can perform an exhaustive search on the number of slices, s, that yields the minimum of (2.49). That is, at first the degree of the Chebyshev polynomial is set to 20. If $20p_1$ is significantly less than (2.49) for any interior slice, we merge n_l leftmost slices with $n_l = 2, 3, .., s-1$ until $20p_1$ is as close to $\max_i p_i m_i$ as possible.

Communication

There are two types of communications in both CheFSI and SS. One is within a column group, and the other is within a slice. The communication required in the multiplication of the Hamiltonian with a distributed vector is limited to within each column group. Processes in the same column group exchange

data via neighborhood collective communication. Communication overhead can be lowered by using non-blocking communication and overlapping the communication and local computation. Another improvement can be done is to use space-filling curves when partitioning the real-space grid. A real-space grid partition based on space-filling curves has good locality of neighboring grid points and thus the communication for stencil traversal can be reduced. This improves the efficiency and scalability of the multiplication of H with vectors. Our preliminary result has shown that, if not communication-bound, a speed-up of 1.5 to 2, compared with a simple real-space grid partition, can be achieved. This will be one of our future studies.

In CheFSI, orthonormalization of basis vectors and other dense matrix-matrix operations such as solving the projected eigenvalue problem involves all processes on the 2D process grid. We rely on the communication layers implemented in PBLAS and ScaLAPACK to perform the necessary data communication required to carry out these operations. The communication cost can be relatively high. In SS, all dense matrix computation is carried out among the column groups mapped to a spectral slice. Consequently, all communication takes place within a subset of processes, which can significantly reduce the communication overhead.

2.7 Space-Filling Curves Based Grid Partitioning Method

Space-filling curves (SFCs) can improve the parallelization efficiency of sparse matrix–vector multiplication (SpMV) operations. A naive domain decomposition method may easily lead to an imbalance in communications when thousands of processors are involved, and prevents one from efficiently parallelizing the eigensolvers and simulating large systems. This problem can be addressed by using SFCs for domain partitioning. SFCs are based on continuous lines that traverse a three-dimensional space or a two-dimensional plane [61]. Using SFCs, data defined on higher-dimensional grids can be represented by a one-dimensional continuous array, which provides an elegant and efficient way to access data [62, 63, 64]. The self-similarity of space-filling curves localizes spatially neighboring grid points in a one-dimensional array [65, 66]. This makes for an efficient computation of the Laplacian acting on wave functions and for effective storing of the nonlocal part of ionic pseudopotentials, such that data communication near subdomain boundaries can be reduced and communication is better balanced over processors. Further improvement can then be obtained by blockwise operations [67, 68].

SFCs based domain partitioning is a kind of graph partitioning that uses only the geometric information (vertex coordinates) and ignores the information of the connectivity between vertices. It is faster and more scalable in the generation of partitions [69], and is more economical in memory usage

compared to the multilevel heuristics based graph partitioning methods [70]. For a regular grid the SFCs based graph partitions are proven "quasi optimal" [70].

Here, we propose a blockwise Hilbert method that partitions a real-space grid based on Hilbert SFCs, the grid-point indices of which are further modified to enable blockwise operations. We implement and test this approach in PARSEC [19], a real-space pseudopotential DFT code. By combining CheFSI, pseudotentials, and high-order finite differences, we achieve highly efficient electronic structure calculations for confined systems. We present results for large silicon nanocrystals up to 12 nm in diameter, which contain over 50,000 atoms.

2.7.1 Grid Partitioning Based on Space-Filling Curves

SFCs can be used to generate efficient grid partitions. As illustrated in Figure 2.9 for a four-processor, two-dimensional case, the stencil corresponding to the Laplacian requires communication between four processors by a simple Cartesian order (SCO) method, while by the Hilbert SFC method the Laplacian would involve only three processors (see the next section for the construction of SFCs). The difference can be more evident when a simulation domain is partitioned into hundreds of processors. By taking advantages of the self-similarity of SFCs, we are able to devise grid partitioning schemes that minimize the boundary communication between processors when performing SpMVs.

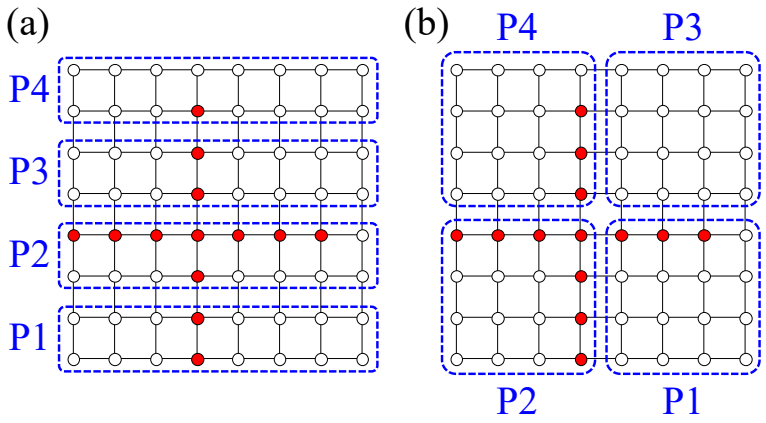

Figure 2.9: Illustration of two-dimensional grid partitioning by (a) the SCO method and (b) the Hilbert SFC method. The red dots constitute 13-point stencils.

Figure 2.10 illustrates the two-dimensional versions of the grid partitioning methods we will examine in this study. The four different grid partitioning methods are (1) SCO, (2) Hilbert, (3) blockwise SCO (the SCO method with grid blocks), and (4) blockwise Hilbert (the Hilbert method with grid blocks). The SCO method simply partitions a real-space grid by running through the Cartesian coordinates in order. "Blockwise" means that, when performing the Laplacian, wave functions of multiple grid points (belonging to the same grid block) are updated simultaneously rather than one grid point at a time.

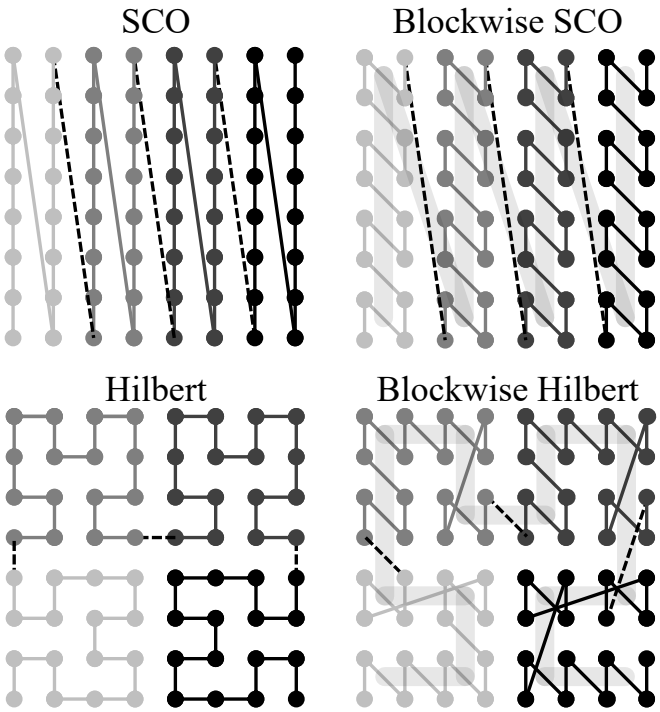

Figure 2.10: Two-dimensional illustration of the four grid partitioning methods. The grid points are partitioned into four processors. The thin solid lines follow the actual memory storage order. The thick gray solid lines in the blockwise methods visualize the order of the grid blocks. The grayscale of the grid points denote the processor to which they belong.

2.7.2 Generation of Blockwise Hilbert Space-Filling Curves

Figure 2.11 shows the first three generations of three-dimensional Hilbert SFCs. The vertices of the virtual boxes are the grid points. Generation 0 has only the central point ($8^0 = 1$ point). The rules to generate the curves can be seen from Generation 1 ($8^1 = 8$ points) to Generation 2 ($8^2 = 64$ points). To obtain the eight virtual boxes in Generation 2, we scale and rotate the

virtual box in Generation 1. For example, to obtain the coordinates of the grid points of the virtual box 5 in Generation 2, we scale the virtual box in Generation 1 by 0.5 in each direction, rotate it around the y-axis by 180 degrees, and translate it to the fifth vertex of the virtual box in Generation 1. Mathematically,

$$\begin{aligned}
\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} &= T_{\text{vertex } 5} + R_{y,180^o} \times 0.5 \times \begin{pmatrix} x \\ y \\ z \end{pmatrix} \\
&= \begin{pmatrix} -0.25 \\ -0.25 \\ 0.25 \end{pmatrix} + \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \times 0.5 \times \begin{pmatrix} x \\ y \\ z \end{pmatrix} \\
&= 0.5 \times \left[\begin{pmatrix} -0.5 \\ -0.5 \\ 0.5 \end{pmatrix} + \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \right],
\end{aligned}$$

where (x, y, z) are the coordinates of the vertices of the virtual box in Generation 1, and (x', y', z') are the coordinates of the vertices of the virtual box 5 in Generation 2. A recursive Hilbert curves generation algorithm is listed in Algorithm 6. The input for the algorithm is the order of the curve. The outputs (`fx`, `fy`, `fz`) are the fractional coordinates of the grid points in the Hilbert order. All of the equations in Algorithm 6 can be derived in the same manner as the above one.

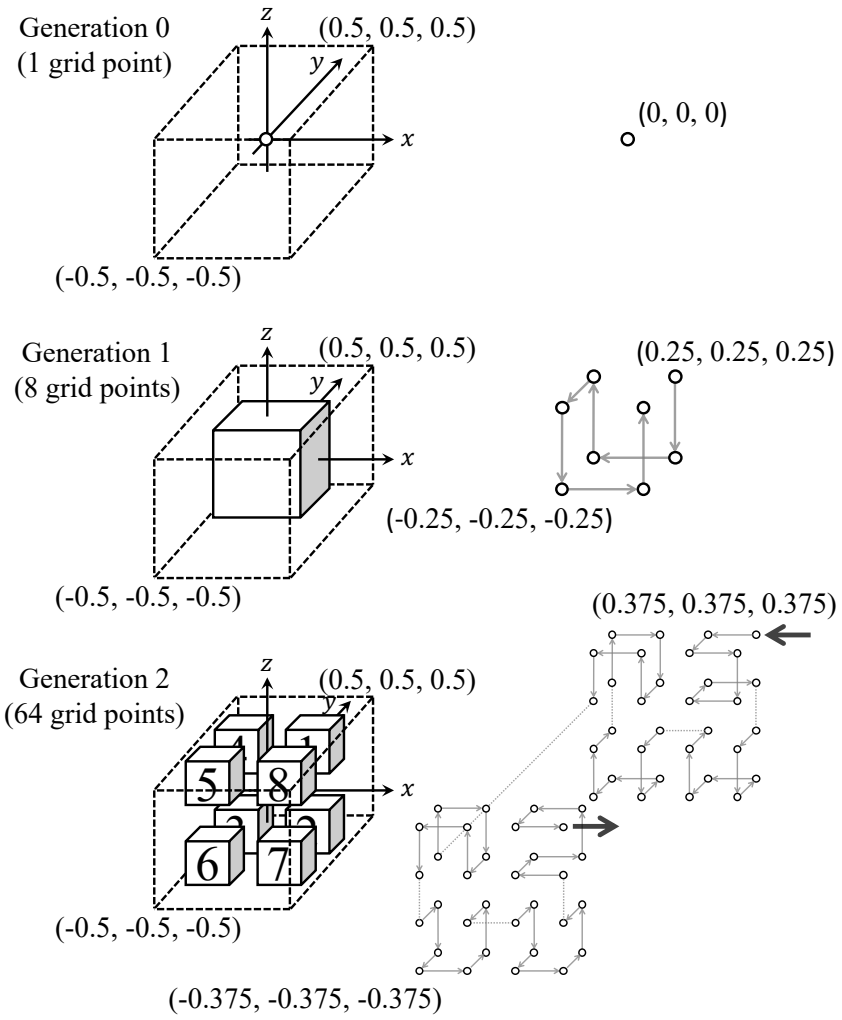

Figure 2.11: The first three generations of three-dimensional Hilbert curves. The generation rules are demonstrated from Generation 1 to Generation 2, where the eight virtual boxes are obtained by scaling, rotating, and translating the one in Generation 1.

Algorithm 6 Generation of grid points based on Hilbert space-filling curves

procedure CreateHilbert3D(`order`, `fx`, `fy`, `fz`)

if `order` is 0 **then**

$\texttt{fx} = \texttt{fy} = \texttt{fz} = 0.0$

else

Create temporary arrays `tx`, `ty`, and `tz`, each with length $8^{\texttt{order}-1}$

call CreateHilbert3D(`order` $-$ 1, `tx`, `ty`, `tz`)

l is the length of the temporary arrays

$j = 1$ ▷ virtual box 1

$\texttt{fx}[j : j + l - 1] = 0.5(0.5 + \texttt{tz})$

$\texttt{fy}[j : j + l - 1] = 0.5(0.5 + \texttt{tx})$

$\texttt{fz}[j : j + l - 1] = 0.5(0.5 + \texttt{ty})$

$j = j + l$ ▷ virtual box 2

$\texttt{fx}[j : j + l - 1] = 0.5(0.5 + \texttt{ty})$

$\texttt{fy}[j : j + l - 1] = 0.5(0.5 + \texttt{tz})$

$\texttt{fz}[j : j + l - 1] = 0.5(-0.5 + \texttt{tx})$

$j = j + l$ ▷ virtual box 3

$\texttt{fx}[j : j + l - 1] = 0.5(-0.5 + \texttt{ty})$

$\texttt{fy}[j : j + l - 1] = 0.5(0.5 + \texttt{tz})$

$\texttt{fz}[j : j + l - 1] = 0.5(-0.5 + \texttt{tx})$

$j = j + l$ ▷ virtual box 4

$\texttt{fx}[j : j + l - 1] = 0.5(-0.5 - \texttt{tx})$

$\texttt{fy}[j : j + l - 1] = 0.5(+0.5 + \texttt{ty})$

$\texttt{fz}[j : j + l - 1] = 0.5(+0.5 - \texttt{tz})$

$j = j + l$ ▷ virtual box 5

$\texttt{fx}[j : j + l - 1] = 0.5(-0.5 - \texttt{tx})$

$\texttt{fy}[j : j + l - 1] = 0.5(-0.5 + \texttt{ty})$

$\texttt{fz}[j : j + l - 1] = 0.5(+0.5 - \texttt{tz})$

$j = j + l$ ▷ virtual box 6

$\texttt{fx}[j : j + l - 1] = 0.5(-0.5 - \texttt{ty})$

$\texttt{fy}[j : j + l - 1] = 0.5(-0.5 - \texttt{tz})$

$\texttt{fz}[j : j + l - 1] = 0.5(-0.5 + \texttt{tx})$

$j = j + l$ ▷ virtual box 7

$\texttt{fx}[j : j + l - 1] = 0.5(+0.5 - \texttt{ty})$

$\texttt{fy}[j : j + l - 1] = 0.5(-0.5 - \texttt{tz})$

$\texttt{fz}[j : j + l - 1] = 0.5(-0.5 + \texttt{tx})$

$j = j + l$ ▷ virtual box 8

$\texttt{fx}[j : j + l - 1] = 0.5(+0.5 + \texttt{tz})$

$\texttt{fy}[j : j + l - 1] = 0.5(-0.5 - \texttt{tx})$

$\texttt{fz}[j : j + l - 1] = 0.5(+0.5 - \texttt{ty})$

Deallocate the temporary arrays `tx`, `ty`, and `tz`

After we obtain the fractional coordinates, we use them to generate the index arrays (integer arrays in the units of grid spacing) that will be used in SpMV. Algorithm 7 shows how to use the outputs of Algorithm 6 to obtain the index arrays. In Algorithm 7 we multiply the fractional coordinates (whose values range in $[-0.5, 0.5]$ since we construct the vertices within a unit cube) by $2^{\texttt{order}+1}$, shift by 1, and divide by 2. Finally we store only the integer part. This results in integer indices ranging from $-2^{\texttt{order}-1} + 1$ to $2^{\texttt{order}-1}$ in each direction.

The indices from Algorithm 7 are in the Hilbert order. We need one more step to reorder the indices of the grid points that belong to the same grid block from the Hilbert order to a regular order, as we will use in the blockwise SCO and the blockwise Hilbert methods. We go through the index arrays blockwise and change the indices of the grid points within each block. That is, for every eight grid points, we modify the indices from a Hilbert order to the regular order: $(0,0,0) \rightarrow (0,0,1) \rightarrow (0,1,0) \rightarrow (0,1,1) \rightarrow (1,0,0) \rightarrow (1,0,1) \rightarrow (1,1,0) \rightarrow (1,1,1)$. Taking the virtual box of Generation 1 in Figure 2.11 as an example, the original order of the eight vertices starts at $(1,1,1)$ (the one with fractional coordinates (0.25, 0.25, 0.25)) and ends at $(1,0,1)$ (the one with fractional coordinates (0.25, -0.25, 0.25)). After rearranging the indices of the grid points, the vertices will start at $(0,0,0)$ (the one with fractional coordinates (-0.25, -0.25, -0.25)) and end at $(1,1,1)$. We perform the same operation for each grid block. Afterwards, we partition the real-space domain evenly into the processors by assigning the indices

arrays evenly among the processors.

Algorithm 7 Generation of grid-point indices in Hilbert order

procedure HILBERT3DINT(`order`, `ix`, `iy`, `iz`)
Create temporary arrays `fx`, `fy`, and `fz`, each with length $8^{\texttt{order}}$

call CREATEHILBERT3D(`order`, `fx`, `fy`, `fz`)

$\texttt{toIndex} = 2^{\texttt{order}+1}$

$\texttt{ix} = \text{INT}((\text{NINT}(\texttt{fx} \times \texttt{toIndex}) + 1)/2)$ ▷ NINT returns the nearest integer of its argument. INT returns the integer part of its argument.
$\texttt{iy} = \text{INT}((\text{NINT}(\texttt{fy} \times \texttt{toIndex}) + 1)/2)$
$\texttt{iz} = \text{INT}((\text{NINT}(\texttt{fz} \times \texttt{toIndex}) + 1)/2)$

Deallocate the temporary arrays `fx`, `fy`, and `fz`

2.7.3 Blockwise Laplacian

Algorithm 8 shows how the blockwise Laplacian is applied to a wave function. A processor may hold many grid blocks. Some of the blocks are internal, which means their updates do not require information from the blocks of other processors. The rest of the blocks are external and require information from the blocks on neighbor processors.

Algorithm 8 Blockwise Laplacian

```
1: procedure LaplacianOperator(wfnIn, wfnOut, coeff)
2:     External blocks of a wave function are sent out to the neighbor processors
3:
4:     for all blocks j on this processor do
5:         rowOffset = RowOffsetOfBlock(j)
6:         call UpdateCentralBlock(
7:               wfnIn[rowOffset+1:rowOffset+8],
8:               wfnOut[rowOffset+1:rowOffset+8], coeff)
9:
10:        for all neighbor blocks k of j do
11:            if k is also on this processor then
12:                rowOffsetNeighbor = RowOffsetOfBlock(k)
13:                call UpdateNeighborBlock(
14:                    wfnIn[rowOffsetNeighbor+1:rowOffsetNeighbor+8],
15:                      wfnOut[rowOffset+1:rowOffset+8],
16:                      NeighborType(j,k), coeff)
17:
18:    Wait until the wave functions from the neighbor processors are ready
19:
20:    for all blocks j on this processor do
21:        rowOffset = RowOffsetOfBlock(j)
22:        for all neighbor blocks k of j do
23:            if k is sent from neighbor processors then
24:                rowOffsetNeighbor = RowOffsetOfBlock(k)
25:                call UpdateNeighborBlock(
26:                    wfnIn[rowOffsetNeighbor+1:rowOffsetNeighbor+8],
27:                      wfnOut[rowOffset+1:rowOffset+8],
28:                      NeighborType(j,k), coeff)
```

At the beginning of the SpMV subroutine, the wave functions are sent to the neighbor processors in a non-blocking manner. After which we perform blockwise wave function update for the internal blocks. Once the wave func-

tions from the neighbor processors are ready, we update the wave functions of the external blocks. This overlap of communication and computation hides the latency of the wave function transfer and improves efficiency.

For 12th-order central differences on a uniform real-space grid, we need 12 points, in addition to the central point, in each dimension. For the blockwise update, since a block comprises $2 \times 2 \times 2$ grid points, in each dimension there will be $12/2 = 6$ blocks. For three dimensions, this results in $6 \times 3 = 18$ neighbor blocks with respect to the central block (Figure 2.12a). Based on this we have two types of blockwise update: one is for the central block, *i.e.*, the block itself (Algorithm 9), and the other for the 18 neighbor blocks (Algorithm 10).

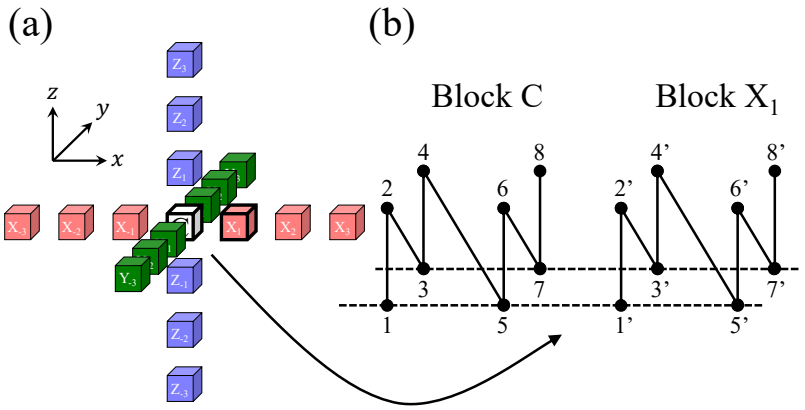

Figure 2.12: Illustration of the update for the central block. (a) The 18 blocks used in updating the central block. (b) The grid points of the central block and block X_1, with the grid points in the regular order.

For example, Figure 2.12b shows the update of the central block from the

contribution of its X_1 block. This can be expressed in a matrix form:

$$\begin{pmatrix} \psi(1) \\ \psi(2) \\ \psi(3) \\ \psi(4) \\ \psi(5) \\ \psi(6) \\ \psi(7) \\ \psi(8) \end{pmatrix} = \begin{pmatrix} c_2 & 0 & 0 & 0 & c_3 & 0 & 0 & 0 \\ 0 & c_2 & 0 & 0 & 0 & c_3 & 0 & 0 \\ 0 & 0 & c_2 & 0 & 0 & 0 & c_3 & 0 \\ 0 & 0 & 0 & c_2 & 0 & 0 & 0 & c_3 \\ c_1 & 0 & 0 & 0 & c_2 & 0 & 0 & 0 \\ 0 & c_1 & 0 & 0 & 0 & c_2 & 0 & 0 \\ 0 & 0 & c_1 & 0 & 0 & 0 & c_2 & 0 \\ 0 & 0 & 0 & c_1 & 0 & 0 & 0 & c_2 \end{pmatrix} \begin{pmatrix} \psi(1') \\ \psi(2') \\ \psi(3') \\ \psi(4') \\ \psi(5') \\ \psi(6') \\ \psi(7') \\ \psi(8') \end{pmatrix}, \tag{2.50}$$

where $\psi(1), \psi(2), ..., \psi(8)$ are the wave functions of the central block, and $\psi(1'), \psi(2'), ..., \psi(8')$ are the wave functions of block X_1. c_i's are the finite-difference coefficients. The subscripts of c_i's denote the distance between grid points. For example, in the equation: $\psi(1) = c_2\psi(1') + c_3\psi(5')$, c_2 is used because points 1 and 1' are at a distance of two (Figure 2.12b), and c_3 is used because points 1 and 5' are at a distance of three. As a result, for 12th-order central differences on a cubic grid, we need $c_0, c_1, ..., c_6$ (shown as the coefficient array, `coeff`, in Algorithms 8, 9, and 10). The pseudocode of the update from an X_1 block is listed in Algorithm 10 as block type 1.

Algorithm 9 Adding the diagonal terms and the contribution from the same block

```
1:  procedure UpdateCentralBlock(vin, vout, coeff)
2:      for j ← 1, 8 do
3:          vout[j] = vout[j] + coeff[0] × vin[j]
4:
5:      for j ← 1, 4 do
6:          vout[j] = vout[j] + coeff[1] × vin[j + 4]
7:          vout[j + 4] = vout[j + 4] + coeff[1] × vin[j]
8:
9:      for j ← 1, 2 do
10:         vout[j] = vout[j] + coeff[1] × vin[j + 2]
11:         vout[j + 2] = vout[j + 2] + coeff[1] × vin[j]
12:         vout[j + 4] = vout[j + 4] + coeff[1] × vin[j + 6]
13:         vout[j + 6] = vout[j + 6] + coeff[1] × vin[j + 4]
14:
15:     vout[1] = vout[1] + coeff[1] × vin[2]
16:     vout[3] = vout[3] + coeff[1] × vin[4]
17:     vout[5] = vout[5] + coeff[1] × vin[6]
18:     vout[7] = vout[7] + coeff[1] × vin[8]
19:     vout[2] = vout[2] + coeff[1] × vin[1]
20:     vout[4] = vout[4] + coeff[1] × vin[3]
21:     vout[6] = vout[6] + coeff[1] × vin[5]
22:     vout[8] = vout[8] + coeff[1] × vin[7]
```

Algorithm 10 Adding the contribution from a neighbor block

```
1: procedure UpdateNeighborBlock(vin, vout, blockType, coeff)
2:     if blockType is 1 then                          ▷ neighbor block X_1
3:         for j ← 1, 4 do
4:             vout[j + 0] = vout[j + 0] + coeff[2] × vin[j + 0] + coeff[3] ×
   vin[j + 4]
5:             vout[j + 4] = vout[j + 4] + coeff[2] × vin[j + 4] + coeff[1] ×
   vin[j + 0]
6:     else if blockType is 2 then                     ▷ neighbor block X_2
7:         for j ← 1, 4 do
8:             vout[j + 0] = vout[j + 0] + coeff[4] × vin[j + 0] + coeff[5] ×
   vin[j + 4]
9:             vout[j + 4] = vout[j + 4] + coeff[4] × vin[j + 4] + coeff[3] ×
   vin[j + 0]
10:    else if blockType is 3 then                     ▷ neighbor block X_3
11:        for j ← 1, 8 do
12:            vout[j + 0] = vout[j + 0] + coeff[6] × vin[j + 0]
13:        for j ← 5, 8 do
14:            vout[j + 0] = vout[j + 0] + coeff[5] × vin[j − 4]
15:    else if blockType is 4 then
16:        ...
17:    else if blockType is 18 then
18:        ...
```

2.8 First-Principles Molecular Dynamics

2.8.1 Equation of Motion for Nuclei

For a nucleus K treated as a classical object, the Newton's second law states that

$$\mathbf{F}_K = M_K \mathbf{a}_K, \tag{2.51}$$

where M_K is the mass of the nucleus and $\mathbf{a}_K$ is its acceleration. Assuming a conservative force field, we have

$$\mathbf{a}_K = \frac{1}{M_K}\mathbf{F}_K = \frac{-1}{M_K}\frac{dE}{d\mathbf{R}_K}, \tag{2.52}$$

where E is the potential energy surface that the nucleus moves along. From Equation (2.8) we know

$$E = E(\{\mathbf{R}_I\}, \{\mathbf{r}_i\}) = \frac{1}{2}\sum_{\substack{I,J \\ I \neq J}} \frac{Z_I Z_J}{|\mathbf{R}_I - \mathbf{R}_J|} + E_{\{\mathbf{R}_I\}}. \tag{2.53}$$

Inserting the energy expression into (2.52), we obtain

$$\mathbf{a}_K = \frac{-1}{M_K}\left(\sum_{\substack{I,K \\ I \neq K}} Z_I Z_K \frac{\mathbf{R}_I - \mathbf{R}_K}{|\mathbf{R}_I - \mathbf{R}_K|^3} + \frac{dE_{\{\mathbf{R}_I\}}}{d\mathbf{R}_K}\right). \tag{2.54}$$

We then invoke the Hellmann–Feynman theorem for the electronic energy term (see Equation (2.6)):

$$\begin{aligned}
\frac{dE_{\{\mathbf{R}_I\}}}{d\mathbf{R}_K} &= \langle\Phi_{\rm e}|\frac{d\hat{H}_{\rm e}}{d\mathbf{R}_K}|\Phi_{\rm e}\rangle \\
&= \langle\Phi_{\rm e}|\frac{d}{d\mathbf{R}_K}(-\sum_i \frac{\nabla_i^2}{2} - \sum_{I,i}\frac{Z_I}{|\mathbf{R}_I-\mathbf{r}_i|} + \frac{1}{2}\sum_{\substack{i,j\\ i\neq j}}\frac{1}{|\mathbf{r}_i-\mathbf{r}_j|})|\Phi_{\rm e}\rangle \\
&= \langle\Phi_{\rm e}|Z_K\sum_i \frac{\mathbf{R}_K-\mathbf{r}_i}{|\mathbf{R}_K-\mathbf{r}_i|^3}|\Phi_{\rm e}\rangle \\
&= Z_K\int d\mathbf{r} n(\mathbf{r})\frac{\mathbf{R}_K-\mathbf{r}}{|\mathbf{R}_K-\mathbf{r}|^3} \\
&= -Z_K\int d\mathbf{r} n(\mathbf{r})\frac{\mathbf{r}-\mathbf{R}_K}{|\mathbf{r}-\mathbf{R}_K|^3}.
\end{aligned} \tag{2.55}$$

We note that, when pseudopotentials are used, the Coulombic term $-\sum_{I,i}\frac{Z_I}{|\mathbf{R}_I-\mathbf{r}_i|}$ should be replaced by the ion core pseudopotentials. We use the original all-electron form in the derivation for brevity.

Combining all together, we obtain the equation of motion for nuclei:

$$\frac{d^2\mathbf{R}_K}{dt^2} = \mathbf{a}_K = \frac{Z_K}{M_K}\left(-\sum_{I\neq K} Z_I\frac{\mathbf{R}_I-\mathbf{R}_K}{|\mathbf{R}_I-\mathbf{R}_K|^3} + \int d\mathbf{r} n(\mathbf{r})\frac{\mathbf{r}-\mathbf{R}_K}{|\mathbf{r}-\mathbf{R}_K|^3}\right). \tag{2.56}$$

The first term on the right-hand side is the contribution from the other nuclei and the second term from that of the electrons.

2.8.2 Born–Oppenheimer Molecular Dynamics

The nuclei move along a potential-energy surface, $E(\{\mathbf{R}_I\}, \{\mathbf{r}_i\})$. According to DFT, the energy depends on the electronic charge density and we can drop the index for electrons and write it as $E(\{\mathbf{R}_I\}, n(\mathbf{r}))$. In Born–Oppenheimer Molecular Dynamics (BOMD) we make sure the electronic charge density, $n(\mathbf{r})$, is always the one of the ground-state solution at every nuclear configuration, $\{\mathbf{R}_I\}$. This contrasts to Car–Parrinello Molecular Dynamics (CPMD) [71], where the electronic charge density might not be the one from the ground-state solution at a certain nuclear configuration.

2.8.3 Langevin Thermostat

We use Langevin thermostat in our work. Langevin thermostat is to simulate the effect of a thermal bath using a Gaussian noise [72, 73].

The Langevin equation is written as:

$$\frac{d^2\mathbf{R}_I}{dt^2} = -\frac{1}{M_K}\frac{dE(\{\mathbf{R}_J\})}{d\mathbf{R}_I} - \beta\frac{d\mathbf{R}_I}{dt} + \frac{1}{M_I}G_I, \tag{2.57}$$

where the last two terms on the right-hand side are Langevin dissipation and fluctuation forces defined by the friction coefficient β and the random Gaussian variables $\{G_I\}$ with a white spectrum:

$$\begin{aligned} &\langle G_I^\alpha(t)\rangle = 0, \\ &\langle G_I^\alpha(t) G_J^\alpha(t')\rangle = 2\beta M_I k_\mathrm{B} T \delta_{IJ}\delta(t-t'), \end{aligned} \tag{2.58}$$

where k_{B} is Boltzmann constant and T is temperature of the thermal bath. The angular brackets denote ensemble or time averages, and α stands for the Cartesian coordinate.

2.8.4 Integration of Equation of Motion

We use Beeman's algorithm [74] to integrate the equation of motion. It is a predictor-corrector algorithm. The predictor for positions is

$$x_{n+1} = x_n + v_n h + \frac{1}{6}(4a_n - a_{n-1})h^2 + O(h^4), \tag{2.59}$$

where h is the time step size. Based on this position we can compute the acceleration of the next step, a_{n+1}, through a DFT calculation, and the velocity of the next step, v_{n+1}, through a corrector of velocity in the following. The correctors for positions and velocities are

$$x_{n+1} = x_n + v_n h + \frac{1}{6}(a_{n+1} + 2a_n)h^2 + O(h^4), \tag{2.60}$$

and

$$v_{n+1} = \frac{1}{h}(x_{n+1} - x_n) + \frac{1}{6}(2a_{n+1} + a_n)h + O(h^4). \tag{2.61}$$

Assuming the prediction for the next position is the same from the predictor and the corrector (Equations (2.59) and (2.60)), we can obtain an estimated

acceleration for the next time step:

$$\tilde{a}_{n+1} = 2a_n - a_{n-1}. \tag{2.62}$$

Replacing a_{n+1} in Equation (2.61) with $\tilde{a}_{n+1}$, we obtain an estimation for the velocity:

$$\tilde{v}_{n+1} = \frac{1}{h}(x_{n+1} - x_n) + \frac{1}{6}(5a_n - 2a_{n-1})h. \tag{2.63}$$

Substituting x_{n+1} from Equation (2.59), we obtain an estimation for the velocity, $\tilde{v}_{n+1}$, based on the current velocity and the previous and current accelerations:

$$\tilde{v}_{n+1} = v_n + \frac{1}{2}(3a_n - a_{n-1})h. \tag{2.64}$$

The estimated velocity will be used in the computation of the Langevin dissipation force of the next time step.

Algorithms 11 and 12 summarize the FPMD methods we will use in this work.

Algorithm 11 Microcanonical Ensembles (NVE) BOMD

Load x_0
Compute the quantum force F_0
Generate v_0 from a Maxwell distribution based on the initial temperature
Adjust v_0 to make the velocity of the center of mass zero
Rescale v_0 to match the initial temperature
Compute the acceleration: $a_0 = \frac{F_0}{M}$
Compute the position of the next step: $x_1 = x_0 + v_0 h + 0.5 a_0 h^2$
Output the current dynamics (x_0, v_0, a_0)

for $n \leftarrow 1, \infty$ **do**
    Compute the quantum force F_n based on x_n
    Compute the acceleration: $a_n = \frac{F_n}{M}$
    Compute the corrector of velocity: $v_n = \frac{1}{h}(x_n - x_{n-1}) + \frac{1}{6}(2a_n + a_{n-1})h$
    Adjust v_n to make the velocity of the center of mass zero
    Compute the predictor of position: $x_{n+1} = x_n + v_n h + \frac{1}{6}(4a_n - a_{n-1})h^2$
    Calculate the kinetic energy (and the temperature T_n)
    Output the current dynamics (x_n, v_n, a_n)

Algorithm 12 Canonical Ensembles (NVT) BOMD

Load x_0
Compute the quantum force F_0
Generate v_0 from a Maxwell distribution based on the initial temperature
Adjust v_0 to make the velocity of the center of mass zero
Rescale v_0 to match the initial temperature
Compute the acceleration that includes the viscous and friction terms $a_0 = \frac{F_0}{M} - \beta v_0 + \frac{G(\beta, T_{\text{set}})}{M}$ ▷ T_{set} is the target temperature
Compute the position of the next step: $x_1 = x_0 + v_0 h + 0.5 a_0 h^2$
Estimate the velocity of the next step: $\tilde{v}_1 = v_0 + a_0 h$
Output the current dynamics (x_0, v_0, a_0)

for $n \leftarrow 1, \infty$ **do**
    Compute the quantum force F_n based on x_n
    Compute the acceleration that includes the viscous and friction terms $a_n = \frac{F_n}{M} - \beta \tilde{v}_n + \frac{G(\beta, T_{\text{set}})}{M}$
    Compute the corrector of velocity: $v_n = \frac{1}{h}(x_n - x_{n-1}) + \frac{1}{6}(2a_n + a_{n-1})h$
    Compute the predictor of position: $x_{n+1} = x_n + v_n h + \frac{1}{6}(4a_n - a_{n-1})h^2$
    Calculate the kinetic energy (and the temperature T_n)
    Estimate the velocity of the next step: $\tilde{v}_{n+1} = v_n + \frac{1}{2}(3a_n - a_{n-1})h$
    Output the current dynamics (x_n, v_n, a_n)

Chapter 3

Results and Discussion

3.1 Spectrum Slicing Method

In this section, we evaluate the performance of the polynomial filtering algorithm. The computational experiments are carried out on the Cori Haswell compute nodes (Intel Xeon E5-2698 v3, 32 cores per node) maintained at the National Energy Research Scientific Computing (NERSC) Center.

We use a silicon nanocrystal with 1,947 Si atoms passivated by 604 H atoms to test the performance of the proposed algorithm. This system is denoted by $Si_{1947}H_{607}$ below. The finite difference order is 12, and the grid spacing is 0.7 bohr, which corresponds to a kinetic energy cutoff of $\sim$20 Ry in a plane wave representation. The simulation domain is a sphere, beyond which the wave functions are set to zero. A margin of at least 7 bohr is preserved between the atoms and the boundary of the simulation domain. The

Hamiltonian size is 1,527,083, and the number of states to be calculated is set to 4,352 (though the number of occupied states is 4,196, we include extra (unoccupied) states to accelerate the convergence, as explained in Section 2.6.1). We use Troullier-Martins norm-conserving pseudopotentials [75], and for H atoms the cutoff radius for 1s is 1.80 bohr, while for Si atoms the cutoff radii for 3s/3p are 2.80/2.80 bohr, respectively. The exchange-correlation functional is local density approximation from Ceperley and Alder, [76] with the parameterization by Perdew and Zunger [77] (LDA CA PZ). The Anderson mixing scheme [78] is used. The mixing ratio is 0.3, and the potential mixing performed at a particular SCF iteration uses the potentials computed in the previous 4 SCF iterations. The convergence criterion (SRE) is set to 0.0001 Ry. Smaller silicon nanocrystals are used to study SCF convergence with filter polynomials of different degrees, and a larger one ($Si_{3893}H_{988}$) is used to study the strong parallel scalability of the SS algorithm.

In all experiments, we use CheFSI in the first few SCF iterations, and then transition to SS in the subsequent SCF iterations. Communication groups are created to map column groups to different spectral slices so that the ith spectral slice owns n_{c_i} column groups, where n_{c_i} is defined in (2.48). Data redistribution among column groups is required as we transition from CheFSI to SS. Some Ritz vectors kept on one column group are sent to adjacent column groups to create Ritz pair overlap that can help accelerate subspace iteration within each slice as discussed in Sections 2.6.1 and 2.6.3.

Component	Description
FLTR	Amplification of lowest-energy states
ORTH	Orthonormalization of wave functions via Cholesky QR
RR	Rayleigh–Ritz procedure (*i.e.*, projection of wave functions into a Ritz matrix, eigen-decomposition of the Ritz matrix, and subspace rotation)

Table 3.1: Main computational components of subspace iteration

3.1.1 Performance Profile

We first report the performance characteristics of both CheFSI and SS when they are run with 256 Cori Haswell cores. The timing measurements of subspace iteration are grouped into the three categories listed in Table 3.1.

We use a Chebyshev polynomial of degree 20 in the CheFSI calculation. The definition of the 2D process grid for the CheFSI calculation is not unique. For example, we can use only one column group and partition the basis vectors into 256 row blocks, which was used in previous implementations of the PARSEC software. At the other extreme, we can partition the basis vectors into 256 column groups (*i.e.*, 1 process per column group). Figure 3.1 shows for one subspace iteration how the timing measurements of different components in the CheFSI calculation change as we change the data layout. By partitioning the basis vectors into multiple column groups that do not require communication during polynomial filtering (FLTR), the performance of this part can be significantly improved. When the basis vectors are distributed among 32 columns groups (*i.e.*, 8 process per column group), the time spent on FLTR is reduced by a factor of 3.9 when compared with

the approach in which 256 processes are used in a single column group to perform the multiplication of H with vectors distributed by rows among the 256 processes. However, as we continue to increase the number of column groups and reduce the number of row blocks within each column group, the performance of FLTR does not change much. This is because the parallel implementation of the sparse matrix-vector multiplication Hx scales well to a modest number of processes (in this case, 8).

However, we can also see that as the number of column groups increases, the orthonormalizaton (ORTH) and Rayleigh–Ritz procedure (RR) in CheFSI become more costly. This is because these components make use of parallel dense matrix–matrix multiplication (PGEMM) which attains the best performance with the ratio between the number of rows and the number of columns is not too large nor small. In order to achieve optimal performance, we use two different data layouts in the CheFSI calculation for FLTR and other dense linear algebra computation. The last bar in Figure 3.1 depicts the performance profile using a 8×32 process grid for FLTR, and transitioning to a 256×1 process grid for other dense linear algebra operations. We use PDGEMR2D to change data distribution. The overhead associated with data reshuffling, which is embedded in ORTH and RR, appears to be small, as we can see from the last bar in Figure 3.1.

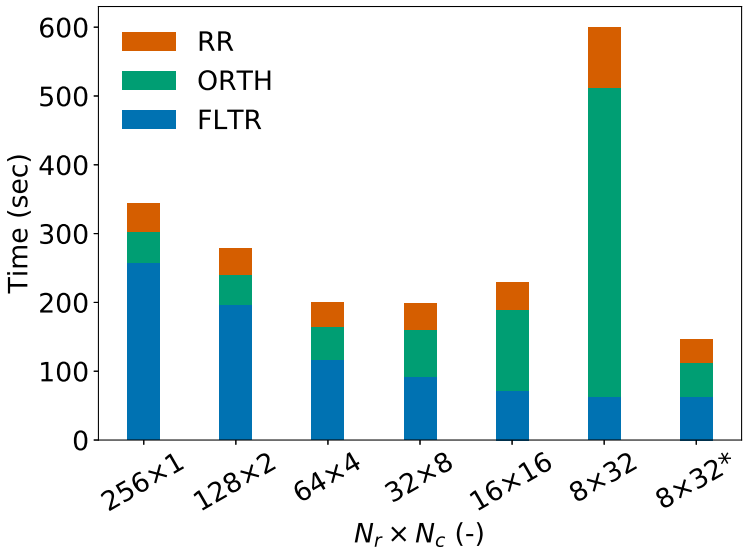

Figure 3.1: Computational time of the key components in the CheFSI calculations with different process grids.

The performance profile of the SS algorithm for one subspace iteration is illustrated in Figure 3.2. The process grid is 16×16, and four slices are used. The numbers of column groups in slice 1–4 are computed to be 1, 5, 6, and 4, respectively. A Chebyshev polynomial of degree 40 is used to compute approximate eigenpairs in the leftmost slice. Bandpass filters of degree 160 are used to compute approximate eigenpairs in the other slices. For comparison, we also plot the performance profile using CheFSI as the leftmost bar. The leftmost slice contains 1,334 eigenvalues among which 808 are actually within the spectral bounds associated with that slice. Slice 2–4 contain 1,970, 2,496 and 1,820 eigenvalues, respectively. Among them 891, 1,482 and 1,015 eigenvalues of interest are within the spectral bounds

associated with these slices. As we can see from Figure 3.2, FLTR takes most of the time in the SS algorithm. All the other computational kernels (ORTH, RR) take a small faction of the total wallclock time. Since $p_i m_i / n_{c_i} \approx 64,000$, we expect the computation to be well load-balanced as is the case. The time spent on each slice is roughly the same ($\sim$ 800 seconds). We can see FLTR in SS costs about an order of magnitude more than that in CheFSI, due to the requirement of high degree filters in SS. Though FLTR in SS is more expensive than that in CheFSI, in this case, the cost of the cubically scaling operations (ORTH+RR) in SS is less than that in CheFSI. This shows the SS algorithm can indeed reduce the cost of the dense linear algebra operations. We note that for slices 2–4 there is no time cost shown for ORTH, because we solve a generalized eigenvalue problem for these interior slices, so the time cost of ORTH is embedded in RR.

Because the degrees of the bandpass filter polynomials used in SS are much higher than the degree of the Chebyshev polynomial used in CheFSI, we expect the wallclock time spent on FLTR to be much longer in SS than that in CheFSI.

For this problem, the significantly higher cost of FLTR in SS cannot be completely offset by the reduction in the cost of ORTH and RR. As a result, the total cost of SS is still higher than that of CheFSI when the problem is solved on 256 cores. We show in Section 3.1.3 that, as the problem size becomes larger and more compute cores are used, the time spent on ORTH and RR will start to dominate in CheFSI calculation regardless how many

compute cores are used, whereas such cost is relatively low in SS calculation. Because FLTR can easily scale to tens of thousands of cores, whereas it is difficult to achieve this kind of scalability for ORTH and RR, SS will scale to much larger number of compute cores and eventually outperform CheFSI.

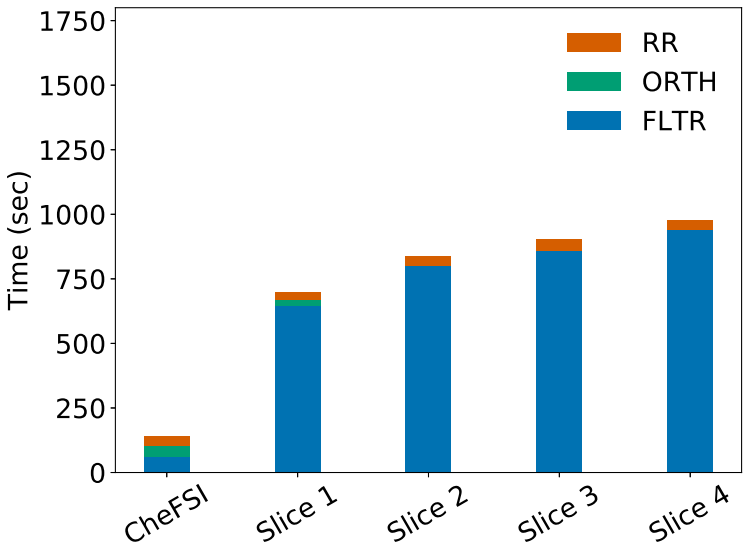

Figure 3.2: Computational time of the key components in the SS calculation. The leftmost bar shows the wallclock time by CheFSI, serving as a reference.

3.1.2 Comparing Subspace Iteration with Lanczos for Spectrum Slicing

As we argued in Section 2.6.1, using subspace iteration to compute the desired eigenpairs within a spectral slice has the advantage of allowing the multiplication of H with different vectors to be performed among different column groups on a 2D process grid. The multiple levels of concurrency are

likely to yield better parallel scalability even though the subspace iteration method has a slower convergence rate compared with the Lanczos algorithm, which cannot perform multiple Hamiltonian and vector multiplications simultaneously.

Figure 3.3 shows, from one of the slices in the simulation of $Si_{1947}H_{604}$, how the cumulative wallclock time spent on both FLTR and ORTH of the Lanczos method increases with respect to the Lanczos iteration number. We compare the cost of the Lanczos method with that of the subspace iteration method for a spectral slice that contains the most eigenpairs. The number of eigenvalues in this slice is $1,363$. The degrees of the polynomials are 200. The Lanczos iteration is performed on a 1D process grid with 256 processes. The subspace iteration is carried out on a 16×16 2D process grid. As expected, the wallclock time increases as we perform more Lanczos iterations. For this slice, $2,538$ Lanczos steps are needed to produce sufficiently accurate approximate eigenpairs. As a result, the total number of Hamiltonian and vector multiplications used by the Lanczos method is $2,538 \times 200 \approx 500,000$. The corresponding wallclock time is nearly $1,500$ seconds.

When subspace iteration is used, we include $3,616$ basis vectors in the subspace to accelerate convergence. The dashed line in Figure 3.3 shows the wallclock time for one subspace iteration, which is around 700 seconds. Even though the subspace iteration perform $3,616 \times 200 > 720,000$ Hamiltonian and vector multiplications, which are more than those performed in the Lanczos method, the wallclock time used by subspace iteration is much

less than that used by the Lanczos method. This is due to the much better parallel scalability of the subspace iteration method as we discussed earlier. We should point out that in many cases, one subspace iteration with a sufficiently high degree polynomial is enough to make the SCF iteration converge. However, in some cases, more than one subspace iteration may be required to ensure SCF convergence.

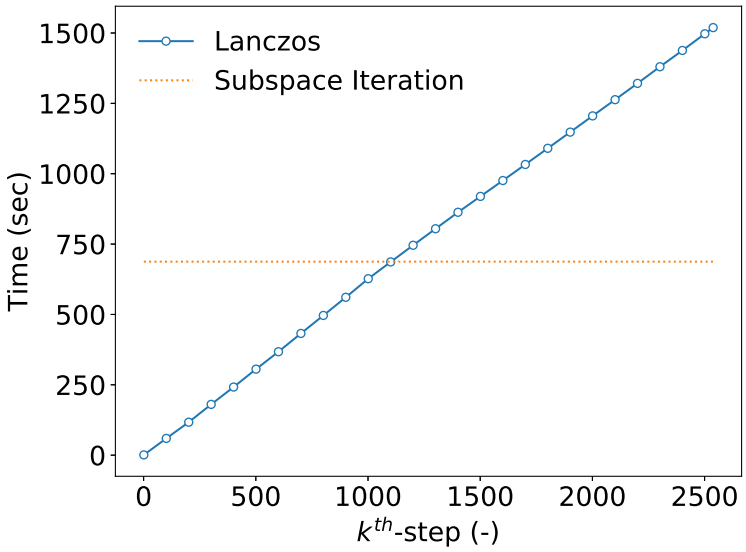

Figure 3.3: Cumulative wallclock time spent on FLTR and ORTH in the Lanczos method with respect to the Lanczos iteration number. The dashed line shows the wallclock time using the subspace iteration method for the same slice.

3.1.3 Parallel Scalability

We now examine the strong parallel scalability of the SS algorithm and compare it with that of CheFSI. We use a $Si_{3893}H_{988}$ silicon nanocrystal for

this test. The dimension of the Hamiltonian is $2,637,711$. The number of states to be computed is $9,216$. Both the CheFSI and SS calculation are performed on 2D process grids with $n_r \times n_c$ processes. Our strong scaling tests use 256, 512, 1,024, 2,048, and 4,096 compute cores. In all tests, n_r is set to 32. For CheFSI, we try $n_c = 8, 16, 32, 64,$ and 128. For SS, the spectrum is divided into 2, 4, 8, and 16 slices. The number of column groups for each slice is $(1, 15)$, $(1, 9, 13, 9)$, $(1, 9, 6, 9, 8, 14, 10, 7)$, and $(1, 6, 9, 8, 5, 6, 10, 7, 6, 10, 12, 14, 12, 7, 4, 11)$, respectively. The degree of the Chebyshev polynomial is 20, and the degrees of the bandpass filter polynomials are 120.

We can clearly see from Figure 3.4 that FLTR in both CheFSI and SS has nearly perfect parallel scalability. The cost of FLTR in SS is ~ 5 times of that associated with CheFSI owing to the higher degree of the bandpass filter polynomials. We also note that due to the large number of states to be included in the SCF calculation, ORTH and RR in CheFSI take more time than FLTR which consists of mostly sparse matrix vector multiplications.

By dividing the spectrum into more slices, the cost of ORTH and RR can be significantly reduced through a trivial parallelization at the slice level. Because ORTH and RR are performed among fewer column groups and uses fewer processes, good scalability can be achieved in these calculations. This can be seen from the green curve marked by empty circles in Figure 3.4. The scalabilty of these types of operations is harder to achieve in CheFSI when many column groups must be distributed on many processes. The green

curve marked by solid circles shows the cost of ORTH and RR in CheFSI actually increases when n_c becomes larger than 16.

As a result, the overall strong scalability of SS is much better than that of CheFSI. When the total number of processes reaches 32×64, the computational time of SS is less than that of CheFSI.

We should note that more states are computed in SS and the overall subspace size in SS is larger than that in CheFSI, because the subspace for each slice is augmented in order to improve convergence and avoid missing states (see Section 2.6.3). However, because SS is compute-bound in comparison with CheFSI which is communication-bound (due to the communication overhead when performing ORTH and RR among all states), it is likely to outperform CheFSI if the degrees of bandpass filter polynomials are not too high and the number of subspace iterations is moderate.

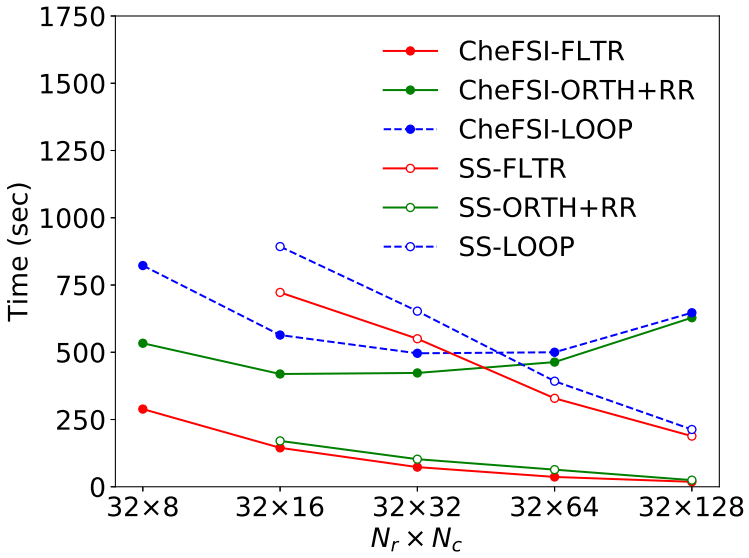

Figure 3.4: Strong scaling test of $Si_{3893}H_{988}$. "LOOP" is the sum of all operations. The computational time shown here is for one subspace iteration.

3.2 Space-Filling Curves Based Grid Partitioning

The simulations in this work were run on NERSC Cori machine. Two types of nodes are available: Haswell and Knights Landing (KNL). Each Haswell node is equipped with two Intel Xeon E5-2698 v3 CPUs, resulting in 32 cores per node, while each KNL node is equipped with one Intel Xeon Phi 7250 CPU, which has 68 cores.

The order of the finite-difference method in Equation (2.35) is $2N$, where we use $N = 6$ through out our simulations. The pseudopotentials are norm-

conserving, constructed by Troullier–Martins method [75] and used in the Kleimann–Bylander form [42]. The cutoff radii for hydrogen and silicon atoms are 1.8 bohr for 1s and 2.78/2.78/2.78 bohr for $3s^2/3p^2/3d^0$, respectively. The exchange-correlation functional is the local-density approximation, as determined numerically by Ceperly and Alder [76] and parameterized by Perdew and Zunger [77]. We use an Anderson algorithm [78] for potential update. The mixing ratio is 0.3, and the mixing history includes four previous steps. The effective potential is converged to within 10^{-4} Ry.

3.2.1 Speedup of Sparse Matrix–Vector Multiplication

Figure 3.5 shows the speedup of SpMV for various problem sizes using the four grid partitioning methods. The system is $Si_{29}H_{36}$ and very fine grids are used in this test. To generate problems of different sizes, three different values of grid spacing are used: 0.180, 0.145, 0.115 bohr. The simulation domain radius is 18 bohr. For the SCO and the Hilbert methods, the resulting numbers of grid points are 4,187,849, 8,013,533, and 16,062,961. For the blockwise methods the numbers of grid points are 4,282,696, 8,159,520, and 16,294,760. Blockwise methods have more grid points because a grid block is counted if any one of its eight grid points is inside the simulation domain. However, we note that the blockwise methods can reference eight grid points at a time by grid-block indices, which is advantageous to data exchange.

The overall speedup (the yellow bars in Figure 3.5) is nearly independent of the problem size. There are two contributions to the speedup: one is the

Hilbert partitioning and the other is the blockwise operation. The speed gain from the blockwise operation, compared to the SCO method, is about 2.2 (the dark blue bars in Figure 3.5), while the speed gain from the Hilbert partitioning is about 2.8 (the green bars in Figure 3.5). As a result, the blockwise Hilbert method is about 6 times faster than the SCO method overall. The speedup for the Hilbert partitioning is due to more balanced communication among processors and reduced communication volume. The speedup from the blockwise operation is due to better utilization of the vector-processing units. We also note that, although the Hamiltonian size for the blockwise methods is slightly bigger than the one of the SCO methods, the use of blockwise operations is still advantageous.

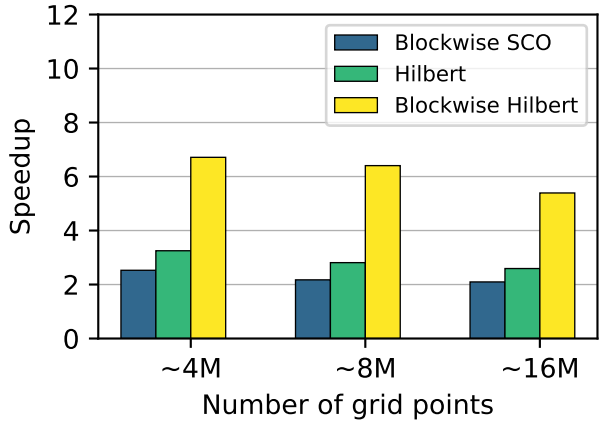

Figure 3.5: The speedup in SpMV using different partitioning methods for various of problem sizes. Eight Haswell nodes (256 processors in total) are used. The speedup within each group is normalized with respect to the one of the SCO method.

3.2.2 Communication Patterns

We examine the amount of transferred data between processors during a Laplacian operation to measure the extent of balance of communication. Figure 3.6 shows typical communication patterns when the number of neighbor processors is small (one to a few). The system is $Si_{29}H_{36}$ and the number of grid points is $\sim$ 4 million. We note that the communication is spread out to more neighbor processors for the Hilbert partitioning methods (the bottom two in Figure 3.6), indicating that network traffic is improved. The advantage of balanced communication is expected to become more evident for large-scale simulations.

On the other hand, if the number of available processors increases, each processor will be in charge of less grid points but may need to communicate with more neighbor processors. In this scenario, the use of Hilbert SFCs can keep the increase in the number of neighbor processors relatively modest compared to that of the SCO method (as illustrated in Figure 2.9).

We note that the use of grid blocks (*i.e.*, the blockwise methods) does not significantly change the communication patterns, because the amount of data exchanged near the boundaries are almost the same. A 12th-order central finite difference still needs 12 grid points (six grid blocks) in each direction.

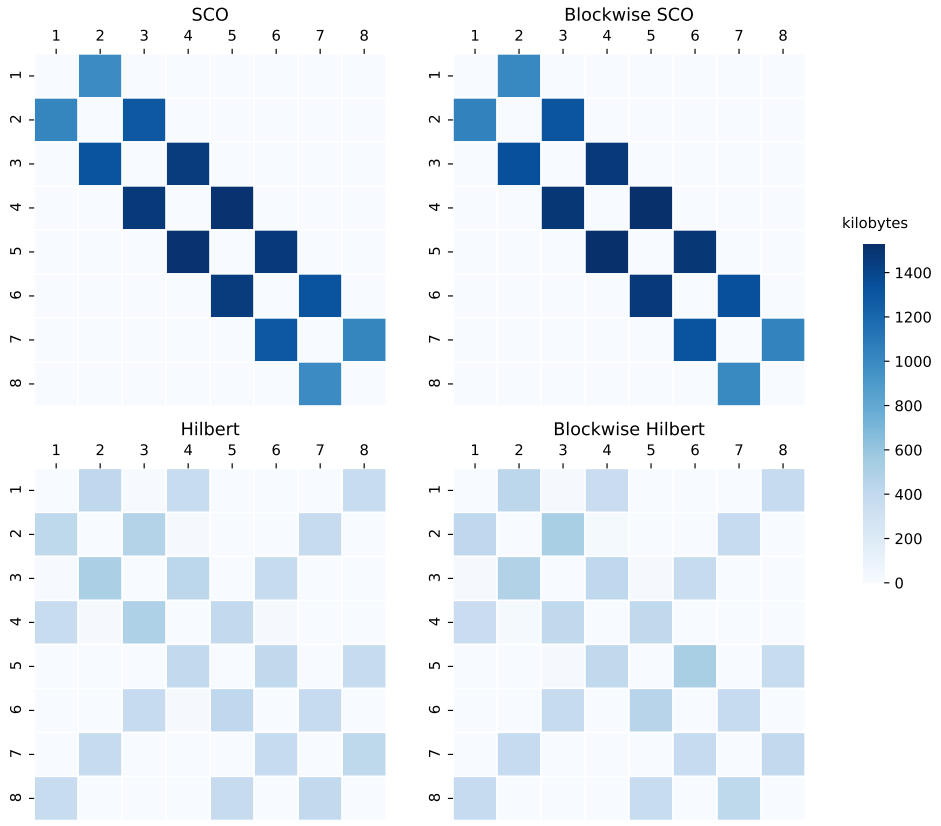

Figure 3.6: Typical communication patterns for the four grid partitioning schemes. Eight processors are used. The color reflects the amount of transferred data during one Laplacian operation, where the darker the more data to be transferred. In each subplot, the y-axes show the sending nodes, and the x-axes show the receiving nodes.

3.2.3 Scalability of Sparse Matrix–Vector Multiplication

The use of Hilbert SFCs not only improves the balance of communication between the processors (Figure 3.6), but also reduces the amount of trans-

ferred data. Figure 3.7 shows the total amount of transferred data in one Laplacian operation and the scalability of the SCO and the blockwise Hilbert methods. The system is $Si_{29}H_{36}$, and the numbers of grid points in this test is 24,733,520, resulting from a grid spacing of 0.1 bohr and a spherical domain radius of 18 bohr.

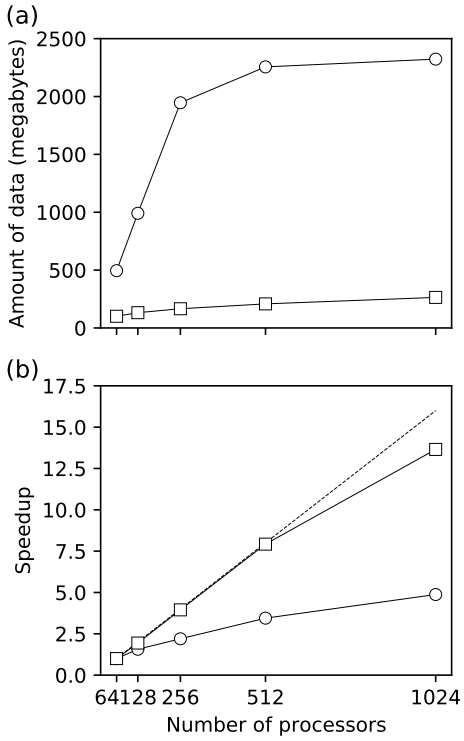

Figure 3.7: (a) Total amount of transferred data in one Laplacian operation and (b) the speedup of SpMV when different number of processors are used. The empty circles denote the SCO method, while the empty squares denote the blockwise Hilbert method. The dashed line in (b) visualizes the ideal scaling. Cori Haswell nodes are used. The speedups are normalized with respect to 64 processors.

Figure 3.7a shows that by the SCO method there is more data to transfer

between the processors. On average the amount of data by the SCO method is an order of magnitude larger compared to that by the blockwise Hilbert method. This impacts the scalability of SpMV by the SCO method. As shown in Figure 3.7b, the SpMV by the SCO method can scale up to around 100 processors, while by the blockwise Hilbert method it is able to scale up to 512 processors. When 1024 processors are used, the speedup by the SCO method drops to 31% of that of the ideal scaling, due to increasing communication overhead and unbalance of communication. The speedup by the blockwise Hilbert method is still 85% of that of the ideal scaling.

3.3 Evolution of Density of States of Silicon Nanocrystals toward Bulk Limit

As an example that illustrates the computational capabilities afforded by the blockwise Hilbert partioning method in a practical scenario of scientific interest, we consider the distribution of Kohn–Sham eigenvalues in a series of Si nanocrystals, which approximately yields the density of states for the nanocrystal. Using the above-described approaches, we are able to simulate silicon nanocrystals with up to 56,555 atoms ($\sim$ 12 nm in diameter), and observe the evolution of density of states from small nanocrystals to the bulk limit.

Table 3.2 lists the sizes of the nanocrystals examined, the computational resources, the number of SCF steps to reach convergence, and the time-to-

solution. We employ a grid spacing of 0.7 bohr except for $Si_{51071}H_{5484}$.

System	N_{grid}	N_{states}	N_{procs}	N_{iter}	Wall time (hr)
$Si_{849}H_{348}$	812,112	2,000	64	21	1.3
$Si_{4001}H_{1012}$	2,707,504	9,216	512	19	4.0
$Si_{10869}H_{1924}$	6,377,184	24,576	4,096	17	8.4
$Si_{23049}H_{3220}$	15,180,904	61,440	16,384	18	27.9
$Si_{51071}H_{5484}$	15,522,368	114,688	65,536	15	28.0

Table 3.2: Test systems and their sizes (the dimension of the Hamiltonian, N_{grid}, and the number of computed states, N_{states}), the number of processors, N_{procs}, the number of self-consistent-field steps to reach a convergence of the energy to within 0.0001 Ry, N_{iter}, and the wall time (time to solution). Cori KNL nodes are used. For the largest one, $Si_{51071}H_{5484}$, we used a grid spacing of 0.9 bohr.

For a system with translational periodicity, the wave vector, $\mathbf{k}$ is a good quantum number and Bloch's theorem can be employed. The resulting energy band structure can be used to determine the density of states [79]. If we label the energy bands as $E_n(\mathbf{k})$ where n is the band index, the density of states, $D(E)$ can be written as

$$D(E) = \sum_n \int \frac{d\mathbf{k}}{\Omega_{BZ}} \delta(E - E_n(\mathbf{k})) \tag{3.1}$$

where Ω_{BZ} is the volume of the first Brillouin zone. The density of states for crystalline silicon is illustrated in Figure 3.8. The energy bands were computed from the same real-space formalism as per the nanocrystals, save periodic boundary conditions were used [80, 81]. The time to compute $E_n(\mathbf{k})$ over a grid of $\mathbf{k}$ is dramatically reduced compared to a nanocrystal, owing

to translational symmetry. We use a Monkhorst–Pack sampling [82] with a density of $20 \times 20 \times 20$.

The structure in crystalline density of states arises from Van Hove singularities, *i.e.*, points where

$$\nabla_{\mathbf{k}} \, E_n(\mathbf{k}) = 0 \tag{3.2}$$

The structure near the singularity can be quantified as corresponding to local maxima, minima or saddle points, and is well-known to have been analyzed accordingly [79]. An interesting question arises for silicon nanocrystals. Namely, one expects the density of states of the nanocrystal to converge to the bulk configuration and presumably replicate structure associated with Van Hove singularities. At what point will this be evident? For the smallest nanocrystal we considered, $Si_{849}H_{348}$, the general features become evident, but not until nanocrystals of several thousand atoms are the Van Hove singularities clearly discernible. For the large nanocrystal, the density of states is virtually the same as the crystalline value, which serves to validate our computational approach to large nanocrystals.

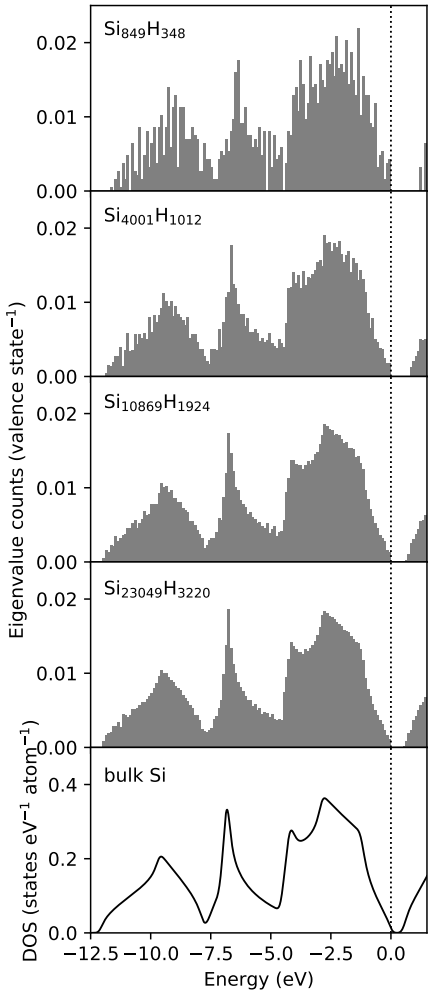

Figure 3.8: Evolution of the density of states. The energy of the highest occupied state is set 0 eV. For the bulk system, the Kohn–Sham eigenvalues are convoluted with a Gaussian function with a standard deviation of 0.1 eV. For the nanocrystals, a histogram bin width of 0.1 eV is used.

The eigenvalue counts of the largest silicon nanocrystal, $Si_{51071}H_{5484}$, is shown in Figure 3.9. The singularities are evident, which indicates that the internal of the nanocrystal well reproduces the bulk environment. This shows

the possibility of simulating charged defects in crystals in a straightforward manner using real-space pseudopotential DFT.

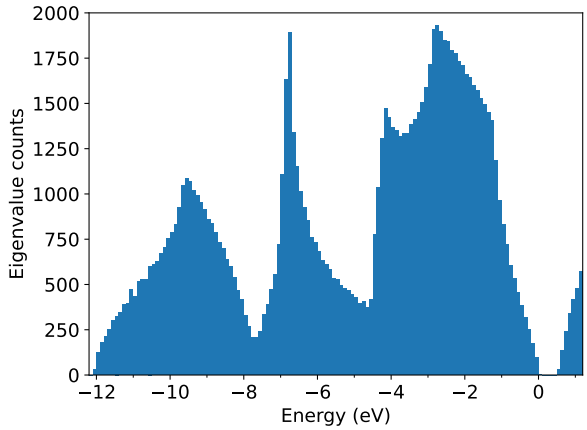

Figure 3.9: Eigenvalue counts for $Si_{51071}H_{5484}$. The energy of the highest occupied state is taken to be 0 eV and a histogram bin width of 0.1 eV is used.

3.4 Proton Transfer in Liquid Water

Water is vital to life and a fundamental study subject. With the improvements in our real-space pseudopotential DFT, we are able to simulate the liquid water system for a long time scale in the level of DFT. We also study the proton transfer in liquid water. More specifically, we study the diffusivity of water and protons in liquid water.

Troullier–Martins pseudopotentials [75] are used. The cutoff radii of $2s^2/2p^2$ for O are 1.45/1.45 bohr. For 1s of H it is 1.00 bohr. The exchange-correlation functional is GGA-PBE. We use GGA-PBE because LDA fails to

reproduces the energetics of water [83]. The grid spacing is 0.25 bohr. The time step size is 20 a.u. (approximately 0.5 fs).

We simulate a system of 64 H_2O. The length of the cubic simulation box is 23.5 bohr. This results in a density of 1.00 g-cm^3.

The initial structure for FPMD is obtained by running classical MD for 1 ns under NVT subjected to 2000 K. LAMMPS [84] is used for the classical MD simulations and TIP4P force field is used for water.

We take the structure of the last time step of the classical MD to be the initial structure for FPMD. For FPMD we first run NVT using Langevin dynamics (see Section 2.8.3) and gradually cool the system down from a temperature of 2000 K to the target ones (300, 325, 350 and 400 K). Afterwards we run NVE and start the sampling.

3.4.1 Thermal Annealing and Energy Conservation

In order to reach lower-energy equilibrium states, we cool down the system using a Langevin thermostat from a high temperature (2000 K), and gradually reduce the thermostat friction coefficient, β. The advantage to reduce the friction coefficient in a multi-stage manner is that there would be higher chance for the system to fall into globally lower-energy states. If we switch to NVE (equivalent to a small β) too early, the system may linger around a globally high-energy state. That is, it cannot jump out of the current local minimum of the potential-energy surface. On the other hand, if we maintain a high β, the system may jump between low- and high-energy states due to

high thermal fluctuation energy, and thus have incorrect dynamics.

Figures 3.10 and 3.11 show the evolution of energies during the multi-stage cooling process for target temperatures 300 and 350 K. There are multiple regions. Region I is NVT. The system is cooled down following a linear temperature change from 2000 K to the target temperature. In Regions II–IV, NVT is performed and temperature is fixed at the target temperature, while β is gradually reduced as 0.01, 0.001, and 0.0001, respectively. The last region is NVE, where we start sampling the dynamics of the system.

One thing worth noting is that in Figure 3.11, there is an evident discontinuity for the total energy between Region IV and V. This is because we take the structure of the last step in Region IV as the initial structure for NVE and renormalize the velocity. Because the structure is the same, we can see the potential energy is continuous. However, the velocity is renormalized and results in a discontinuity in the kinetic energy, and hence a discontinuity in the total energy.

In Region V (NVE), we observe good energy conservation. The energy drift is less than 5 K-ps^{-1}.

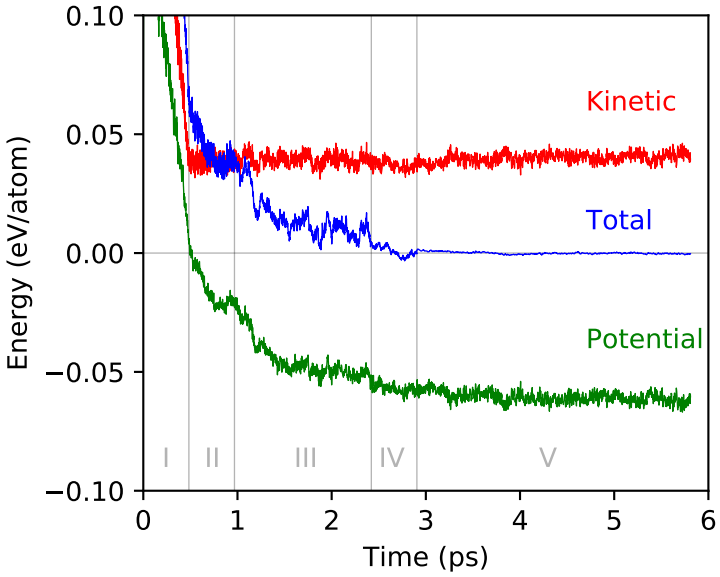

Figure 3.10: Energy evolution of the case cooling from 2000 K to 300 K. In Region I NVT is performed, and the controlled temperature is 2000 K to 300 K. The Langevin damping parameter, β, is 0.01. For Regions II–IV, NVT is performed at 300 K. β is 0.01, 0.001, 0.0001, respectively. In Region V NVE is performed.

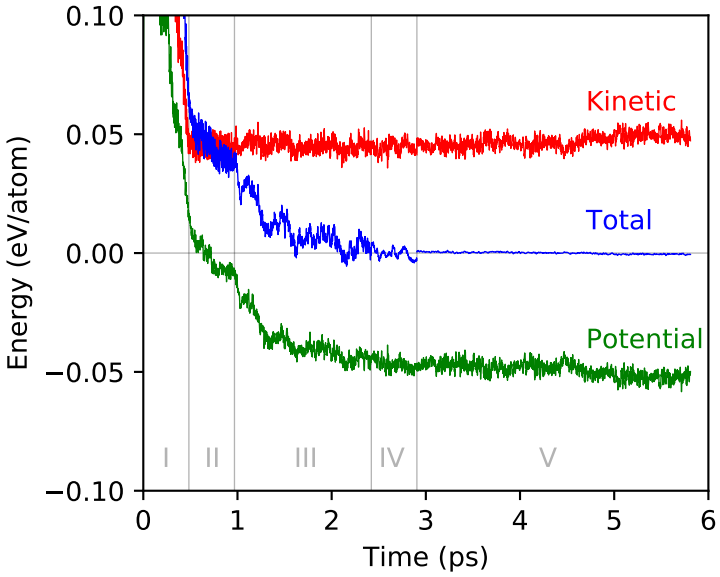

Figure 3.11: Energy evolution of the case cooling from 2000 K to 350 K. In Region I NVT is performed, and the controlled temperature is 2000 K to 350 K. The Langevin damping parameter, β, is 0.01. For Regions II–IV, NVT is performed at 350 K. β is 0.01, 0.001, 0.0001, respectively. In Region V NVE is performed.

3.4.2 Diffusivity of Water and Proton

Diffusivity of water is a good indicator for whether our water model is accurate. Figure 3.12 shows the diffusivity of water under different simulation temperatures. We measure the mean-squared displacements of the oxygen atoms of water and, by using a random-walk model, the diffusivity, D, is computed as

$$D = \frac{1}{6t}\langle|\mathbf{R}|^2\rangle,$$

where $\langle|\mathbf{R}|^2\rangle$ is the ensemble average of the squared displacements (mean-squared displacement) and t is time measured from the beginning of the random walk. The preliminary results of the diffusivity along with experimental data are listed in Table 3.3.

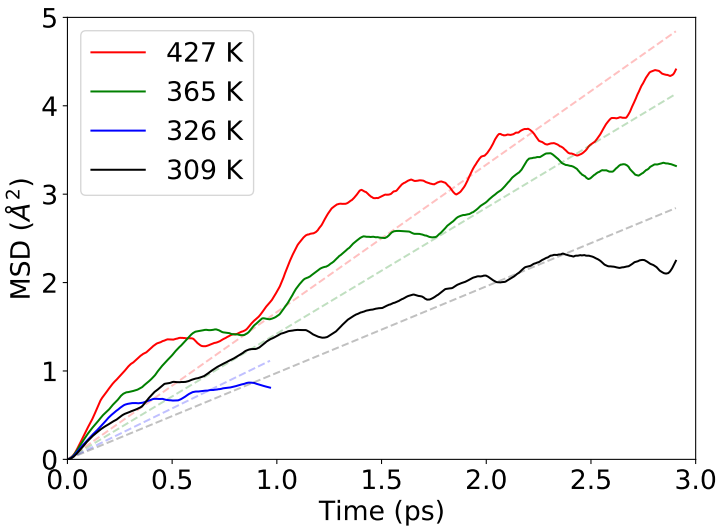

Figure 3.12: Mean-squared displacement (MSD) of water in liquid water under different temperatures. The dotted lines are the fitting curves. The y-intercepts of which are assumed zero.

From Table 3.3 we found the simulated diffusivity is less sensitive to temperature, compared to the experiment data. In the simulation, the change in diffusivity is 0.115 $Å^2$-ps^{-1} across a temperature range of 118 K, while in experiment the diffusivity changes 0.226 $Å^2$-ps^{-1} across a range of only 40 K. We also observed that our simulations diffusivity are lower than the experiment. For example, we obtained a diffusivity of 0.163 $Å^2$-ps^{-1} at 309 K, while in

experiment, at 298 K it is 0.2299 $Å^2$-ps^{-1}. Similar underestimation has been summarized and reported by Gillan *et al.* [83] This is because GGA-PBE functional tends to produce over-structured water in which water molecules can hardly diffuse. To reproduce experimental dynamics, in simulations we can adopt a slightly higher temperature. From Table 3.3 we can see that, in order to obtain a similar diffusivity of water as experiment under room temperature, we may use a temperature of 365 K. That is, 60–70 K higher than the temperatures in experiment.

We note that we do not know the melting temperature of water when GGA-PBE is used. A further study on the melting temperature with GGA-PBE would justify the use of a higher simulation temperature.

Average temperature ± std (K)	Diffusivity ($Å^2$-ps^{-1})
PARSEC	
427 ± 19	0.278
365 ± 21	0.237
326 ± 14	0.192
309 ± 16	0.163
Experiment	
318	0.3575
298	0.2299
278	0.1313

Table 3.3: Diffusivity of water in liquid water. The experimental data are from Mills [1].

To study proton transfer, we add one extra proton into the liquid water system. A compensating background charge is used to keep the system charge-neutral.

Figure 3.13 shows the mean-squared displacement of a proton in liquid water. We monitor the motion of the oxygen atom of the hydronium ion (H_3O^+). We can see there are several "jumps," where the proton transfers occur.

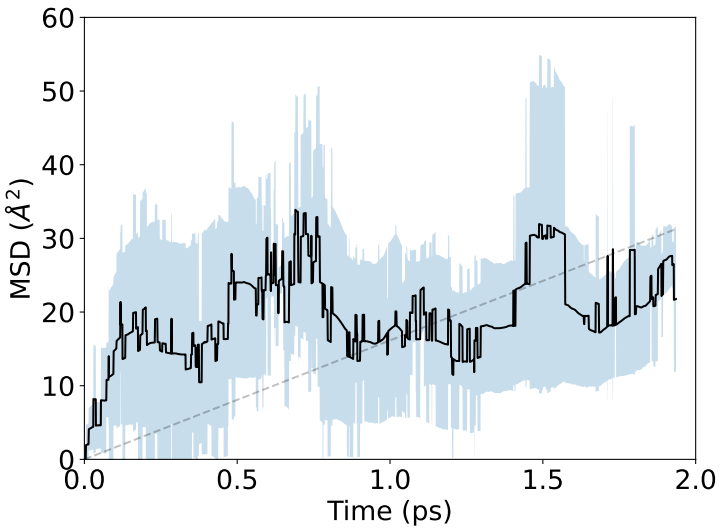

Figure 3.13: Mean-squared displacement (MSD) of a proton in liquid water. The shade denotes the standard deviation computed over three runs. The dotted line is the fitting curve, where the y-intercept is assumed zero.

The preliminary results of the diffusivity of proton in liquid water is listed in Table 3.4. Our simulation result is 2.69 $Å^2$-ps^{-1}. From Table 3.4 we found our diffusivity of proton is similar to those of other simulations but about 3 times higher than the experimental one. This could be reasonable since we simulated at a higher temperature. On the other hand, we also list the ratio of the diffusivity of the proton over that of water. The experimental value is

4.05, and our result is 2–3 times bigger than it, but is similar to those of the other simulations.

We note that there are efforts to improve the exchange-correlation functionals to obtain correct diffusivity of water without simulating at a different temperature [85, 86].

Temperature (K)	Diffusivity ($Å^2$-ps^{-1})	D_{H^+}/D_{H_2O}
	PARSEC	
353	2.69	11.4
	Other simulations	
440	3.00	5.4
300	1.02	23.1
	Experiment	
298	0.9311	4.05

Table 3.4: Diffusivity of a proton in liquid water. The data of other simulations are from Fischer *et al.* [2]), and the experimental value is taken from CRC Handbook of Chemistry and Physics [3].

3.5 Water Adsorption on Titanium Dioxide Surfaces

Titanium dioxide (TiO_2) is a widely used, non-toxic photocatalyst. The water splitting on TiO_2 surfaces is a challenging subject. Surface types, morphology, and treatment, all of which would influence the catalytic reactivity. There are two types of water adsorption on TiO_2 surfaces: molecular and dissociative. The types of TiO_2 surfaces that lead to dissociation of water

have been a subject of discussion [4, 6]. With the advent of computational tools, we are able to explore a great range of parameter space and study the intricacy between the interplay of water and TiO_2 surfaces.

Troullier–Martins pseudopotentials [75] are used. The cutoff radii of $4s^2/4p^0/3d^2$ for Ti are 2.54/2.96/2.25 bohr. Partial core-correction is used for Ti with a core cutoff of 1.96 bohr. For the parameters for O and H, see Section 3.4. The exchange-correlation functional is GGA-PBE. The grid spacing is 0.25 bohr. We use a Monkhorst–Pack sampling [82] with a density of $4 \times 4 \times 4$. For structural relaxation the convergence criterion for the maximum force is 0.01 Ry-bohr^{-1}. For the FPMD simulations the time step size is 20 a.u. (approximately 0.5 fs).

3.5.1 Structural Properties and Cohesive Energy of Titania

Table 3.5 shows the optimized structure of rutile and anatase TiO_2 (the coordinates are listed in Appendix C). We found that the simulated lattice constants with GGA-PBE for both rutile and anatase TiO_2 are a few percents removed from the experimental values. Glassford *et al.* used LDA and obtained results in better agreement with experiment [87]. This indicates that GGA with a four-electron Ti pseudopotential might not be adequate to describe the chemical environment of TiO_2. Tegner *et al.* [88] discussed that a 12-electron pseudopotential, including the semi-core states, can more

accurately describe TiO_2. Another solution is to use multireference pseudopotentials proposed by Reis *et al.* [89].

For the bulk modulus and its derivative, we obtained 194 GPa and 5.24 for rutile TiO_2, which are comparable to the experimental values 211 GPa and 6.5. The results of anatase TiO_2 agree better with experiment. Our simulated bulk modulus and its derivative for anatase TiO_2 are 178 GPa and 4.90, which are very close to the experimental values 179 GPa and 4.5.

For the cohesive energy, we obtained 18.1 eV per TiO_2 unit, which is close to the experimental value 19.9 eV. The cohesive energy of anatase TiO_2 is similar that of rutile TiO_2. This is because cohesive energy measures the easiness of breaking a crystal into its constituent units. The small difference in the cohesive energy between the rutile and anatase TiO_2 reflects that the reorganization energy to transform between the two phases is small.

	a (Å)	c (Å)	B_0 (GPa)	$B_0^{'}$	E^{coh} (eV/TiO_2)
		Rutile			
PARSEC:	4.76 (+3.8%)	3.13 (+6.0%)	194	5.24	18.1
Other sim.	4.634 (+1.0%)	2.963 (+0.3%)	204	4.62	21.44
Exp	4.587	2.954	211 ± 10	6.5 ± 0.7	19.9
		Anatase			
PARSEC:	4.01(+6.0%)	9.64(+1.5%)	178	4.90	18.4
Other sim.	3.786(+0.1%)	9.737(+2.5%)	176	2.99	21.54
Exp	3.782	9.502	179 ± 2	4.5 ± 1.7	-

Table 3.5: Comparison between calculated structural properties of the rutile and anatase TiO_2. The data of the other simulations and experiment are from Lazzeri *et al.* [4] The difference between calculated and measured values are shown as a percentage in parentheses.

3.5.2 Surface Energy

We study the surface energy of various anatase TiO_2 surfaces. Slab semi-periodic boundary condition is used, and there is at least 10 bohr of vacuum above either side of the slab. We use a Monkhorst–Pack sampling [82] with a density of 4×4.

Figure 3.14 shows four different types of surfaces: (101), (100), (001), and (001) 1×4 ad-molecule (ADM) model. Table 3.6 lists the preliminary results of the surface energy. We found that the surface energy of the unrelaxed and relaxed surface structure can be as large as 0.7 eV. Therefore, it is necessary to fully relax the surface in order to get the correct surface energy. Our surface energy (the energy difference between the relaxed surfaces and the bulk) is in good agreement with other simulations (within about 0.2 eV). The (101) surface has the lowest surface energy, suggesting that during synthesis of TiO_2 surfaces, it is easier to form (101) surfaces. On the other hand, the surface energy of (001)-(1×4) is only half of that of the (001) surface. This indicates that it is easier to form reconstructed (001)-(1×4) surfaces compared to pristine (001) surfaces.

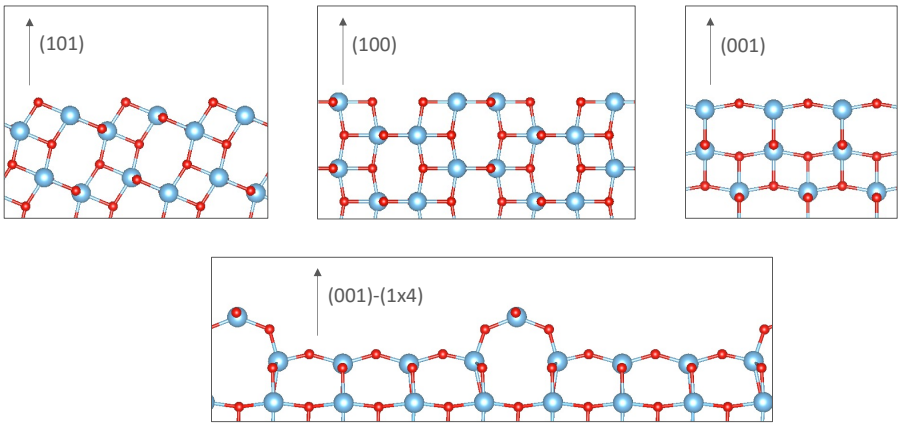

Figure 3.14: Types of surfaces of anatase TiO_2 we examine in this study

			PARSEC		Other simulations	
	N_{L}	N_{at}	E^{unrl}	E^{rel}	E^{unrl}	E^{rel}
(101)	6	36	1.03	0.59	1.28	0.49
(101)	4	24	0.99	0.69	-	0.45
(100)	6	36	1.34	0.66	1.59	0.58
(100)	4	24	1.31	0.73	-	0.63
(001)	6	18	1.40	1.19	1.12	0.98
(001)	4	12	1.30	1.14	-	0.98
(001)-(1×4)	6	78	-	0.73	-	-
(001)-(1×4)	4	54	-	0.79	-	0.51

Table 3.6: Types of anatase TiO_2 surfaces, the number of layers (N_{L}), the number of atoms in the simulation box (N_{at}), the surface energy of the unrelaxed (E^{unrl}) and relaxed (E^{rel}) structure. The surface energy is in J-m^{-2}. The results of the other simulations are from Lazzeri *et al.* [4] and Lazzeri *et al.* [5]

3.5.3 Adsorption Energy of Water

In this section we compute the adsorption energy of water with various coverage percentage and on different surfaces of anatase TiO_2. We examined (101) and (001) surfaces. All of them have four layers, with the bottom two layers fixed. We use supercells that have 48 atoms in both models. Four threefold-coordinated and four fourfold-coordinated Ti are exposed on the (101) and (001) surfaces, respectively. This enables us to examine water coverage of 0.25, 0.50, or 1.00, when we place one, two, or four water molecules into the models. The sizes of the simulation boxes are 19.73 bohr $\times$ 15.16 bohr and 15.16 bohr $\times$ 15.16 bohr, respectively. Slab semi-periodic boundary condition is used, and there is at least 10 bohr of vacuum above either side of the slabs.

Figures 3.15 shows the optimized structure of the (101) surface with water coverage of 0.25 (*i.e.*, one extra water molecule). For the molecular adsorption (Figure 3.15(a)), the O atom of water forms a covalent bond with the surface Ti while the two H atoms form hydrogen bonds with two twofold-coordinated surface O atoms. There are two types of dissociative adsorption of water on a (101) surface: inter (Figure 3.15(b)) or intra (Figure 3.15(c)). For inter-dissociative adsorption one of the water O–H bond is broken and the H atom forms a covalent bond with one of the far twofold-coordinated surface O atoms. As for intra-dissociative adsorption, the H atom forms a covalent bond with the near twofold-coordinated surface O atom.

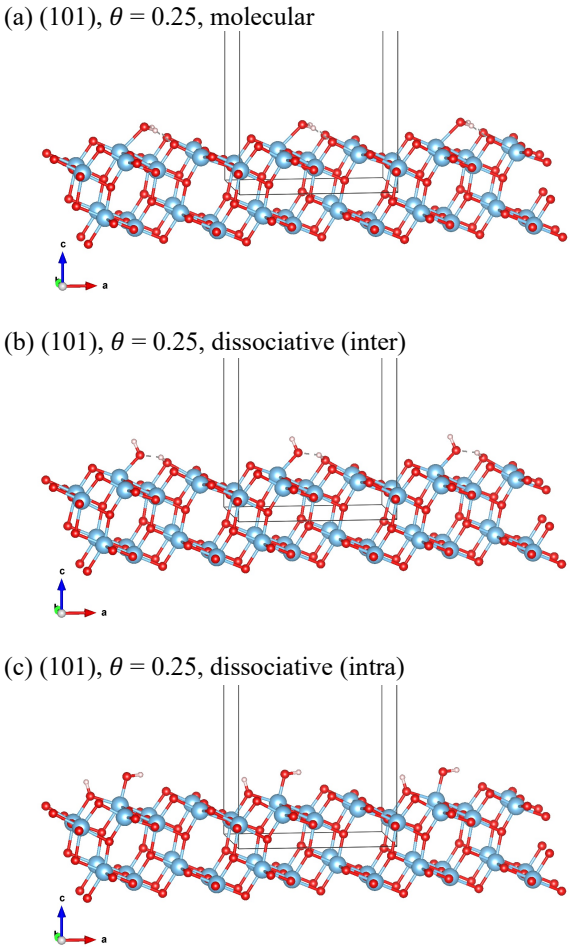

Figure 3.15: Water adsorption on anatase TiO_2 (101) surfaces. The water coverage is 0.25. (a) is molecular adsorption, (b) is dissociative adsorption (inter), and (c) is dissociative adsorption (intra).

Figures 3.16 shows the optimized structure of the (001) surface with water coverage of 0.50 (*i.e.*, two extra water molecules). For the molecular adsorption (Figure 3.16(a)), the O atom of water forms a covalent bond with

the surface Ti while one of the H atoms forms a hydrogen bond with the near twofold-coordinated surface O atoms along ⟨100⟩. For the dissociative adsorption of water on a (001) surface (Figure 3.16(b)), H atoms form covalent bonds with surface O atoms while Ti–O bonds are broken.

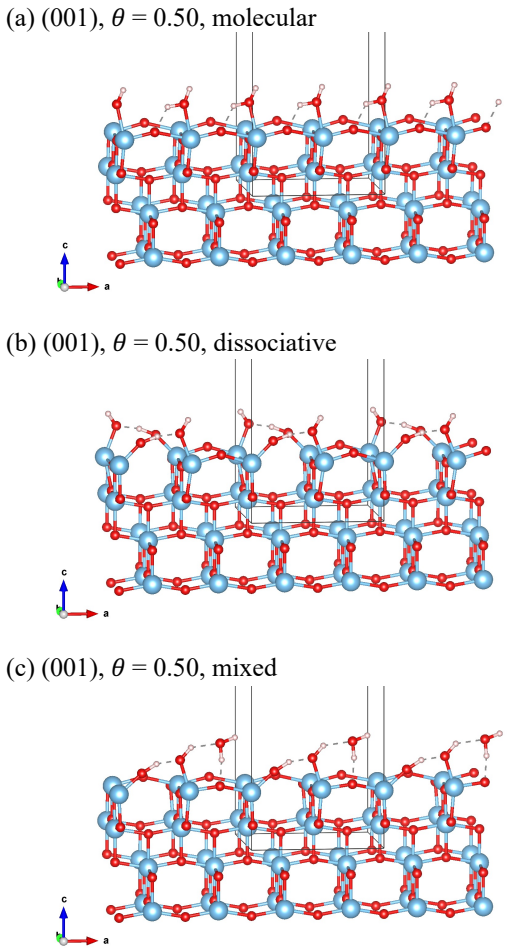

Figure 3.16: Water adsorption on anatase TiO_2 (001) surfaces. The water coverage is 0.50. (a) is molecular adsorption, (b) is dissociative adsorption, and (c) is mixed adsorption.

Interestingly, at water coverage of 0.25, there is only dissociative adsorption on the (001) surface, because of the weakening of the Ti–O bond by the newly formed O–H functional group. However, it is possible to have a stable configuration of molecular adsorption with water coverage of 0.50. That is, if multiple water molecules are approaching the surface simultaneously, the Ti–O bonds may not be weakened to the extent of breaking. It is also possible that, after one water is dissociatively adsorbed on the surface, the reactivity of the surface is weakened and another water molecule may come and be adsorbed physically. We then obtain the mixed adsorption (Figure 3.16(c)).

Table 3.7 shows the preliminary results of water adsorption energy of anatase TiO_2 surfaces. For (101) surfaces, we found the molecular adsorption energy is larger, indicating that it is the preferred adsorption type. As for (001) surfaces, we observed the opposite trend that dissociative adsorption energy is larger. This implies that the (001) surface would be a better surface for water splitting applications. The water coverage percentage does not affect much the adsorption energy per H_2O, and the adsorption energy of the mixed type of the (001) surface falls between the molecular and dissociative types as expected.

	PARSEC			Other simulations		
Surface, θ	ΔH_{H_2O}	$\Delta H_{H,OH}$	ΔH_{mix}	ΔH_{H_2O}	$\Delta H_{H,OH}$	ΔH_{mix}
(101), 0.25	1.30	0.68, 1.03	-	0.74	0.23, 0.30	-
(101), 1.00	0.99	0.75	-	0.72	0.44	-
(001), 0.25	-	1.73	-	-	1.59	-
(001), 0.50	0.78	1.56	0.90	0.81	1.44	-
(001), 1.00	0.77	-	0.79	0.82	-	1.01

Table 3.7: Adsorption energy ΔH per H_2O molecule (eV) on anatase (101) and (001) surfaces with various water coverage θ using PARSEC. The data of the other simulations are from Vittadini *et al.* [6]. Subscripts refer to the character of the adsorption state: H_2O is molecular, H,OH is dissociative, mix is mixed. The two values of $\Delta H_{H,OH}$ refer to two different isomers (interpair, and intrapair) of the dissociated molecule.

We also perform FPMD of water on (001) anatase TiO_2 surfaces to study their interaction and the water splitting process. Figure 3.17 shows some snapshots of a water molecule (with water coverage of 0.25) on the (001) surface. Initially the molecule was placed at 2.8 angstrom above the surface. The water molecule then tilted itself due to the interaction with the surface. At around 0.5 ps the H atom of the water was strongly attracted by the O atom of the surface. Afterwards, the water molecule was dissociated and the Ti–O bond was broken. As a result, for water coverage of 0.25 on the (001) surface the water dissociates spontaneously.

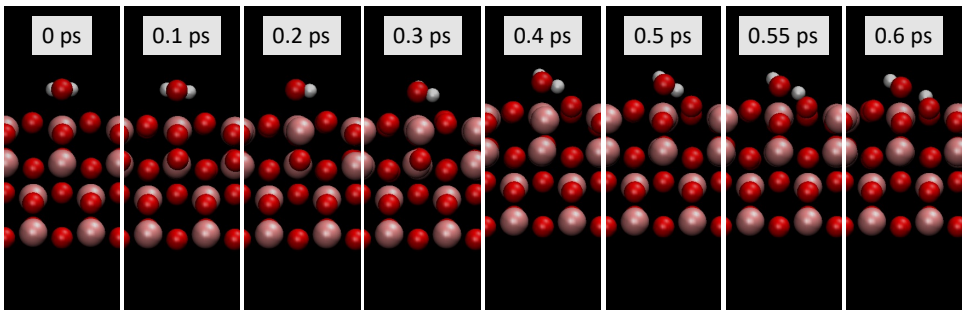

Figure 3.17: FPMD simulation of water on anatase TiO_2 (001) surfaces. The dissociation of a water molecule is observed.

Chapter 4

Conclusion and Outlook

In the dissertation I presented two methods to improve the original CheFSI method. The first improvement is a polynomial filtering SS method, which extends the SS framework established in the work of Schofield *et al.* [55] We discussed several practical issues in implementing polynomial filtering for solving the Kohn–Sham DFT problem in real space. We proposed using a hybrid Chebyshev and bandpass filter polynomial filtering scheme to compute the invariant subspace required to construct the electronic charge density. The latter type of polynomial filtering is combined with the SS method [55] designed to reduce the computational overhead involved in solving the projected eigenvalue problem in the CheFSI method. We showed how to partition the spectrum based on an estimated density of states that can be obtained from a few Lanczos iterations. We proposed using subspace iteration (instead of thick-restarted Lanczos iteration) to compute approximate

eigenpairs within each interior spectral slice in the SS algorithm. Harmonic Ritz vectors (instead of Ritz vectors) are used as approximate eigenvectors. To achieve scalable parallel performance, we used a 2D process grid that allows the multiplication of H with multiple vectors to be performed efficiently. We discussed how to partition and map column groups within the 2D process grid to spectrum slices to achieve good load balance in the SS procedure. Numerical examples were presented to show the performance profiles of both the CheFSI and SS components of the computation.

We demonstrated that for large problems SS can outperform CheFSI on a large number of compute cores due to the higher level concurrency of the algorithm and reduced cost for solving the projected subspace problem, despite the much larger number of Hamiltonian vector multiplications used by the algorithm. The optimal performance of the SS algorithm depends largely on how the spectrum is partitioned and how computational resources are allocated to different spectrum slices. This is currently done in a case by case manner. An efficient procedure can be developed to automate this process.

We should also note that a number of techniques such as the use of mixed precision arithmetic to reduce data movement and memory requirement and overlapping computation and communication can be used to further improve the performance of both CheFSI and SS for large-scale simulations performed on next-generation supercomputers that are equipped with accelerators. These techniques have been demonstrated to be effective by Das *et*

al. [90]

The second method we proposed to improve the original CheFSI method is a grid partitioning method based on SFCs. We implemented grid partitioning methods based on SFCs in a real-space DFT code, PARSEC. We showed that the blockwise Hilbert grid partitioning method speeds up SpMV and improves the scalability of the CheFSI method. We also investigated the evolution of the density of states of silicon nanocrystals up to 56,555 atoms. The use of Hilbert ordering offers better balance of communication and reduces the communication overhead. The blockwise stencil update of Laplacian in SpMV increases the opportunity of vectorization. These methods can also be applied to, and benefit, other real-space codes that discretize the domain on a regular grid. An extension to non-Cartesian grids and a generalized scheme with grid blocks of various sizes is also possible.

The improvements in CheFSI from the SS method and the SFCs based grid partitioning method speed up real-space pseudopotential DFT calculations, and make it possible to solve electronic structure of large systems. Along with the improved algorithms, three applications were presented in the dissertation: density of states of silicon nanocrystals from small ones to the bulk limit, proton transfer in liquid water, and water adsorption on TiO_2 surfaces.

From the density of states of silicon nanocrystals from small ones to the bulk limit, we observed the emergence of Van Hove singularities without invoking Bloch theorem. This triggers interesting questions such as how we

interpret the quantum number k in the Bloch theorem and how the Van Hove singularities connect to the wave functions when the concept of lattice symmetry is not clear in the calculations.

In the second application we examined the dynamics of protons in liquid water by FPMD simulations. The diffusivity of proton is several times larger than the experimental value. We found that the energy conservation is good by our methods. However, more simulation runs are needed to have good statistics.

For the last application we studied the surfaces of anatase TiO_2. We found the trend of surface energy in our calculations is consistent with other simulations. The surface energy of the (101) surface is lower than that of the (001) and (100) surfaces. This indicates that it is easier to form (101) surfaces experimentally. For water adsorption energy, the trend is also similar to other simulations. The (001) surface tends to dissociate water molecules. However, when multiple water molecules are adsorbed at the same time, it is possible that molecular adsorption happens.

Real-space pseudopotential DFT is a powerful tool. With the improvements presented in the dissertation, we are able to explore larger systems of interest and longer-timescale phenomena. It would be interesting to apply the tool to the studies of interfacial systems, such as titanium dioxide-water interfaces and other electrochemical interfacial reactions that are important to battery design, semiconductor surface etching and cleaning.

Appendix A

Hartree Atomic Units

The following quantities are set to 1.

Reduced Planck constant	$\hbar \to 1$	atomic unit of action
Elementary charge	$e \to 1$	atomic unit of charge
Bohr radius	$a_0 \to 1$	atomic unit of length
Electron mass	$m_\mathrm{e} \to 1$	atomic unit of mass

$$4\pi\epsilon_0 \to 1 \tag{A.1}$$

After the normalization, the energies are in Hartree, the lengths are in bohr, the masses are in electron mass, etc.

Appendix B

Derivation of Poisson Equation for the Hartree Potential

The Hartree potential energy is originally written in an integral form:

$$V_{\mathrm{H}}(\mathbf{r}) = \int d\mathbf{r}' \frac{n(\mathbf{r}')}{|\mathbf{r}' - \mathbf{r}|}. \tag{B.1}$$

We can recast it into a differential form,

$$\nabla^2 V_{\mathrm{H}}(\mathbf{r}) = -4\pi n(\mathbf{r}), \tag{B.2}$$

which is the Poisson equation and can be solved efficiently by conjugate gradient method, etc.

Here is the derivation. First we examine the integration of $\nabla^2 \frac{1}{|\mathbf{r}|}$:

$$
\begin{aligned}
\int d\mathbf{r}\nabla^2\frac{1}{|\mathbf{r}|} &= \lim_{\eta\to 0}\int d\mathbf{r}\nabla^2\frac{1}{\sqrt{|\mathbf{r}|^2+\eta^2}} \\
&= \lim_{\eta\to 0}\int_0^\infty 4\pi r^2 dr\nabla^2\frac{1}{\sqrt{r^2+\eta^2}} \\
&= \lim_{\eta\to 0}\int_0^\infty 4\pi r^2 dr\frac{1}{r}(\frac{r}{\sqrt{r^2+\eta^2}})'' \\
&= \lim_{\eta\to 0}\int_0^\infty 4\pi r^2 dr\frac{1}{r}\frac{-3\eta^2 r}{(r^2+\eta^2)^{\frac{5}{2}}} \\
&= \lim_{\eta\to 0} -12\pi\int_0^\infty dr\frac{\eta^2 r^2}{(r^2+\eta^2)^{\frac{5}{2}}},
\end{aligned}
\tag{B.3}
$$

where we have used the Laplacian in spherical coordinates:

$$
\nabla^2 f = \frac{1}{r}\frac{\partial^2}{\partial r^2}(rf) + \frac{1}{r^2\sin\theta}\frac{\partial}{\partial\theta}(\sin\theta\frac{\partial f}{\partial\theta}) + \frac{1}{r^2\sin^2\theta}\frac{\partial^2 f}{\partial\phi^2}.
$$

Let $r = \eta\xi$ and $\xi = \tan\alpha$, we have

$$
\begin{aligned}
\int d\mathbf{r}\nabla^2\frac{1}{|\mathbf{r}|} &= \lim_{\eta\to 0} -12\pi\int_0^\infty d\xi\frac{\xi^2}{(\xi^2+1)^{\frac{5}{2}}} \\
&= \lim_{\eta\to 0} -12\pi\int_0^{\frac{\pi}{2}}\frac{1}{\cos^2\alpha}d\alpha\frac{\tan^2\alpha}{(\frac{1}{\cos\alpha})^5} \\
&= \lim_{\eta\to 0} -12\pi\frac{\sin^3\alpha}{3}\Big|_0^{\frac{\pi}{2}} \\
&= \lim_{\eta\to 0} -4\pi \\
&= -4\pi.
\end{aligned}
\tag{B.4}
$$

From the above equation, we know

$$\nabla^2 \frac{1}{|\mathbf{r}|} = -4\pi\delta(\mathbf{r}). \tag{B.5}$$

Therefore, taking Laplacian on the both sides of Equation (B.1) and using Equation (B.5), we obtain

$$\begin{aligned} \nabla^2 V_{\mathrm{H}}(\mathbf{r}) &= \int d\mathbf{r}' n(\mathbf{r}') \nabla^2 \frac{1}{|\mathbf{r}' - \mathbf{r}|} \\ &= -4\pi \int d\mathbf{r}' n(\mathbf{r}') \delta(\mathbf{r}' - \mathbf{r}) \\ &= -4\pi n(\mathbf{r}). \end{aligned} \tag{B.6}$$

Appendix C

Structure of Rutile and Anatase Titanium Dioxide

The optimized lattice constants of rutile titanium dioxide, a and c, are 8.99 and 5.91 bohr. The optimized lattice constants of anatase titanium dioxide, a and c, are 7.58 and 18.22 bohr.

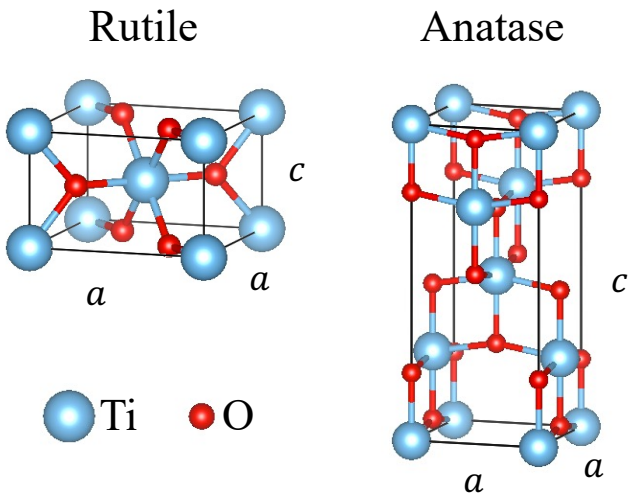

Figure C.1: Unit cells of the rutile and anatase titanium dioxide.

Atom	Position (in fractional coordinates)
Ti1	$(0.0000, 0.0000, 0.0000)$
Ti2	$(0.5000, 0.5000, 0.5000)$
O1	$(0.3034, 0.3034, 0.0000)$
O2	$(-0.3034, -0.3034, 0.0000)$
O3	$(0.8034, 0.1966, 0.5000)$
O4	$(-0.8034, -0.1966, 0.5000)$

Table C.1: Coordinates of the atoms in a unit cell of rutile titanium dioxide

Atom	Position (in fractional coordinates)
Ti1	$(0.000, 0.000, 0.000)$
Ti2	$(0.500, 0.000, 0.250)$
Ti3	$(0.500, 0.500, 0.500)$
Ti4	$(0.000, 0.500, 0.750)$
O1	$(0.000, 0.000, 0.213)$
O2	$(0.000, 0.000, 0.787)$
O3	$(0.500, 0.000, 0.463)$
O4	$(0.500, 0.000, 0.037)$
O5	$(0.500, 0.500, 0.713)$
O6	$(0.500, 0.500, 0.287)$
O7	$(0.000, 0.500, 0.963)$
O8	$(0.000, 0.500, 0.537)$

Table C.2: Coordinates of the atoms in a unit cell of anatase titanium dioxide

Bibliography

[1] R. Mills. Self-diffusion in normal and heavy water in the range 1–45°. *The Journal of Physical Chemistry*, 77(5):685–688, 1973.

[2] S. A. Fischer, B. I. Dunlap, and D. Gunlycke. Correlated dynamics in aqueous proton diffusion. *Chemical Science*, 9(35):7126–7132, 2018.

[3] W. M. Haynes. *CRC Handbook of Chemistry and Physics*. CRC Press, 2016.

[4] M. Lazzeri, A. Vittadini, and A. Selloni. Structure and energetics of stoichiometric TiO_2 anatase surfaces. *Physical Review B*, 63(15):155409, 2001.

[5] M. Lazzeri and A. Selloni. Stress-driven reconstruction of an oxide surface: The anatase $TiO_2(001) - (1 \times 4)$ surface. *Physical Review Letters*, 87(26):266105, 2001.

[6] A. Vittadini, A. Selloni, F. P. Rotzinger, and M. Grätzel. Structure and

energetics of water adsorbed at TiO_2 anatase (101) and (001) surfaces. *Physical Review Letters*, 81(14):2954–2957, 1998.

[7] P. A. M. Dirac and R. H. Fowler. Quantum mechanics of many-electron systems. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, 123(792):714–733, 1929.

[8] P. Hohenberg and W. Kohn. Inhomogeneous electron gas. *Physical Review*, 136(3B):B864–B871, 1964.

[9] W. Kohn and L. J. Sham. Self-consistent equations including exchange and correlation effects. *Physical Review*, 140(4A):A1133–A1138, 1965.

[10] T. L. Beck. Real-space mesh techniques in density-functional theory. *Reviews of Modern Physics*, 72(4):1041, 2000.

[11] L. Frediani and D. Sundholm. Real-space numerical grid methods in quantum chemistry. *Physical Chemistry Chemical Physics*, 17(47): 31357–31359, 2015.

[12] J. Bernholc, M. Hodak, and W. Lu. Recent developments and applications of the real-space multigrid method. *Journal of Physics: Condensed Matter*, 20(29):294205, 2008.

[13] O. Cohen, L. Kronik, and A. Brandt. Locally refined multigrid solution of the all-electron Kohn–Sham equation. *Journal of Chemical Theory and Computation*, 9(11):4744–4760, 2013.

[14] 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. *Journal of Chemical Theory and Computation*, 15(12): 6859–6864, 2019.

[15] S. R. Jensen, S. Saha, J. A. Flores-Livas, W. Huhn, V. Blum, S. Goedecker, and L. Frediani. The elephant in the room of density functional theory calculations. *The Journal of Physical Chemistry Letters*, 8(7):1449–1457, 2017.

[16] V. Blum, R. Gehrke, F. Hanke, P. Havu, V. Havu, X. Ren, K. Reuter, and M. Scheffler. Ab initio molecular simulations with numeric atom-centered orbitals. *Computer Physics Communications*, 180(11):2175–2196, 2009.

[17] V. Michaud-Rioux, L. Zhang, and H. Guo. RESCU: A real space electronic structure method. *Journal of Computational Physics*, 307:593–613, 2016.

[18] J. R. Chelikowsky, N. Troullier, and Y. Saad. Finite-difference-pseudopotential method - electronic-structure calculations without a basis. *Physical Review Letters*, 72(8):1240–1243, 1994.

[19] L. Kronik, A. Makmal, M. L. Tiago, M. M. G. 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. *physica status solidi (b)*, 243(5):1063–1079, 2006.

[20] J. I. Iwata, D. Takahashi, A. Oshiyama, T. Boku, K. Shiraishi, S. Okada, and K. Yabana. A massively-parallel electronic-structure calculations based on real-space density functional theory. *Journal of Computational Physics*, 229(6):2339–2363, 2010.

[21] X. Andrade, D. Strubbe, U. De Giovannini, A. H. Larsen, M. J. T. Oliveira, J. Alberdi-Rodriguez, A. Varas, I. Theophilou, N. Helbig, M. J. Verstraete, L. Stella, F. Nogueira, A. Aspuru-Guzik, A. Castro, M. A. L. Marques, and A. Rubio. Real-space grids and the Octopus code as tools for the development of new simulation approaches for electronic systems. *Physical Chemistry Chemical Physics*, 17(47):31371–31396, 2015.

[22] W. H. Mi, X. C. Shao, C. X. Su, Y. Y. Zhou, S. T. Zhang, Q. Li, H. Wang, L. J. Zhang, M. S. Miao, Y. C. Wang, and Y. M. Ma. ATLAS: A real-space finite-difference implementation of orbital-free density functional theory. *Computer Physics Communications*, 200:87–95, 2016.

[23] S. Ghosh and P. Suryanarayana. SPARC: Accurate and efficient finite-difference formulation and parallel implementation of density functional theory: Isolated clusters. *Computer Physics Communications*, 212:189–204, 2017.

[24] S. Ghosh and P. Suryanarayana. SPARC: Accurate and efficient finite-

difference formulation and parallel implementation of density functional theory: Extended systems. *Computer Physics Communications*, 216: 109–125, 2017.

[25] J. E. Pask, B. M. Klein, P. A. Sterne, and C. Y. Fong. Finite-element methods in electronic-structure theory. *Computer Physics Communications*, 135(1):1–34, 2001.

[26] B. Kanungo and V. Gavini. Real time time-dependent density functional theory using higher order finite-element methods. *Physical Review B*, 100(11):115148, 2019.

[27] J. R. Chelikowsky. The pseudopotential-density functional method applied to nanostructures. *Journal of Physics D: Applied Physics*, 33(8): R33–R50, 2000.

[28] P. Schwerdtfeger. The pseudopotential approximation in electronic structure theory. *ChemPhysChem*, 12(17):3143–3155, 2011.

[29] Y. Saad, J. R. Chelikowsky, and S. M. Shontz. Numerical methods for electronic structure calculations of materials. *SIAM Review*, 52(1):3–54, 2010.

[30] Y. Zhou, Y. Saad, M. L. Tiago, and J. R. Chelikowsky. Parallel self-consistent-field calculations via Chebyshev-filtered subspace acceleration. *Physical Review E*, 74(6):066704, 2006.

[31] Y. Zhou, J. R. Chelikowsky, and Y. Saad. Chebyshev-filtered subspace iteration method free of sparse diagonalization for solving the Kohn–Sham equation. *Journal of Computational Physics*, 274:770–782, 2014.

[32] K.-H. Liou, A. Biller, L. Kronik, and J. R. Chelikowsky. Space-filling curves for real-space electronic structure calculations. *Journal of Chemical Theory and Computation*, 17(7):4039–4048, 2021.

[33] A. S. Banerjee, L. Lin, W. Hu, C. Yang, and J. E. Pask. Chebyshev polynomial filtered subspace iteration in the discontinuous Galerkin method for large-scale electronic structure calculations. *J. Chem. Phys.*, 145 (15):154101, 2016.

[34] A. S. Banerjee, L. Lin, P. Suryanarayana, C. Yang, and J. E. Pask. Two-level Chebyshev filter based complementary subspace method: Pushing the envelope of large-scale electronic structure calculations. *Journal of Chemical Theory and Computation*, 14(6):2930–2946, 2018.

[35] J. Winkelmann, P. Springer, and E. D. Napoli. ChASE: Chebyshev accelerated subspace iteration eigensolver for sequences of hermitian eigenvalue problems. *ACM Transactions on Mathematical Software*, 45(2): 21:1–21:34, 2019.

[36] K.-H. Liou, C. Yang, and J. R. Chelikowsky. Scalable implementation of polynomial filtering for density functional theory calculation in PARSEC. *Computer Physics Communications*, 254:107330, 2020.

[37] W. Hu, L. Lin, and C. Yang. DGDFT: A massively parallel method for large scale density functional theory calculations. *The Journal of Chemical Physics*, 143(12):124110, 2015.

[38] B. Kanungo and V. Gavini. Large-scale all-electron density functional theory calculations using an enriched finite-element basis. *Physical Review B*, 95(3):035112, 2017.

[39] J. R. Chelikowsky. *Introductory Quantum Mechanics with MATLAB: For Atoms, Molecules, Clusters, and Nanocrystals*. Wiley, 2019.

[40] H. Hellmann. A new approximation method in the problem of many electrons. *The Journal of Chemical Physics*, 3(1):61–61, 1935.

[41] E. Fermi. Sopra lo spostamento per pressione delle righe elevate delle serie spettrali. *Il Nuovo Cimento (1924-1942)*, 11(3):157, 2008.

[42] L. Kleinman and D. M. Bylander. Efficacious form for model pseudopotentials. *Physical Review Letters*, 48(20):1425–1428, 1982.

[43] X. Gonze, R. Stumpf, and M. Scheffler. Analysis of separable potentials. *Physical Review B*, 44(16):8503–8513, 1991.

[44] R. Lehoucq, D. Sorensen, and C. Yang. *ARPACK Users' Guide*. Society for Industrial and Applied Mathematics, 1998.

[45] A. V. Knyazev. Toward the optimal preconditioned eigensolver: Locally

optimal block preconditioned conjugate gradient method. *SIAM Journal on Scientific Computing*, 23(2):517–541, 2001.

[46] E. R. Davidson. The iterative calculation of a few of the lowest eigenvalues and corresponding eigenvectors of large real-symmetric matrices. *Journal of Computational Physics*, 17(1):87–94, 1975.

[47] J. A. Duersch, M. Y. Shao, C. Yang, and M. Gu. A robust and efficient implementation of LOBPCG. *SIAM Journal on Scientific Computing*, 40(5):C655–C676, 2018.

[48] E. L. Stiefel. Kernel polynomials in linear algebra and their numerical applications. *U.S. National Bureau of Standards, Applied Mathematics Series*, 49:1–22, 1958.

[49] C. Yang. *Accelerating the Arnoldi Iteration – Theory and Practice*. PhD thesis, Rice University, Houston, TX, 1998.

[50] Y. Zhou, Y. Saad, M. L. Tiago, and J. R. Chelikowsky. Self-consistent-field calculations using Chebyshev-filtered subspace iteration. *Journal of Computational Physics*, 219(1):172–184, 2006.

[51] B. Parlett. *The Symmetric Eigenvalue Problem*. Society for Industrial and Applied Mathematics, 1998.

[52] G. H. Golub and C. F. V. Loan. *Matrix Computations (3rd Ed.)*. Johns Hopkins University Press, 1996.

[53] Y. Saad. Chebyshev acceleration techniques for solving nonsymmetric eigenvalue problems. *Mathematics of Computation*, 42:567–588, 1984.

[54] R. Li, Y. Xi, E. Vecharynski, C. Yang, and Y. Saad. A thick-restart Lanczos algorithm with polynomial filtering for hermitian eigenvalue problems. *SIAM Journal on Scientific Computing*, 38(4):A2512–A2534, 2016.

[55] G. Schofield, J. R. Chelikowsky, and Y. Saad. A spectrum slicing method for the Kohn–Sham problem. *Computer Physics Communications*, 183 (3):497–505, 2012.

[56] L. Lin, Y. Saad, and C. Yang. Approximating spectral densities of large matrices. *SIAM Review*, 58(1):34–65, 2016.

[57] L. N. Trefethen and D. Bau. *Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, 1997.

[58] G. Stewart. *Matrix Algorithms*. Society for Industrial and Applied Mathematics, 2001.

[59] G. L. G. Sleijpen and J. van den Eshof. On the use of harmonic Ritz pairs in approximating internal eigenpairs. *Linear Algebra and its Applications*, 358(1):115–137, 2003.

[60] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stan-

ley, D. Walker, and R. C. Whaley. *ScaLAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997.

[61] H. Sagan. *Space-Filling Curves*. Springer-Verlag, 1994.

[62] J. K. Lawder and P. J. H. King. Using space-filling curves for multi-dimensional indexing. In *Proceedings of the 17th British National Conferenc on Databases: Advances in Databases*, BNCOD 17, pages 20–35, Berlin, Heidelberg, 2000. Springer-Verlag.

[63] P. Xu and S. Tirthapura. Optimality of clustering properties of space-filling curves. *ACM Transactions on Database Systems*, 39(2):1–27, 2014.

[64] A. N. Yzelman and D. Roose. High-level strategies for parallel shared-memory sparse matrix-vector multiplication. *IEEE Transactions on Parallel and Distributed Systems*, 25(1):116–125, 2014.

[65] B. Moon, H. V. Jagadish, C. Faloutsos, and J. H. Saltz. Analysis of the clustering properties of the Hilbert space-filling curve. *IEEE Transactions on Knowledge and Data Engineering*, 13(1):124–141, 2001.

[66] S. P. Sastry, E. Kultursay, S. M. Shontz, and M. T. Kandemir. Improved cache utilization and preconditioner efficiency through use of a space-filling curve mesh element- and vertex-reordering technique. *Engineering with Computers*, 30(4):535–547, 2014.

[67] K. P. Lorton and D. S. Wise. Analyzing block locality in Morton-order and Morton-hybrid matrices. *ACM SIGARCH Computer Architecture News*, 35(4):6–12, 2007.

[68] C. Yount, J. Tobin, A. Breuer, and A. Duran. YASK—yet another stencil kernel: A framework for HPC stencil code-generation and tuning. In *2016 Sixth International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing (WOLFHPC)*, pages 30–39.

[69] R. Borrell, J. C. Cajas, D. Mira, A. Taha, S. Koric, M. Vazquez, and G. Houzeaux. Parallel mesh partitioning based on space filling curves. *Computers & Fluids*, 173:264–272, 2018.

[70] S. Schamberger and J.-M. Wierum. Partitioning finite element meshes using space-filling curves. *Future Generation Computer Systems*, 21(5): 759–766, 2005.

[71] R. Car and M. Parrinello. Unified approach for molecular dynamics and density-functional theory. *Physical Review Letters*, 55(22):2471–2474, 1985.

[72] R. Biswas and D. R. Hamann. Simulated annealing of silicon atom clusters in Langevin molecular dynamics. *Physical Review B*, 34(2): 895–901, 1986.

[73] N. Binggeli, J. L. Martins, and J. R. Chelikowsky. Simulation of si clusters via Langevin molecular dynamics with quantum forces. *Physical Review Letters*, 68(19):2956–2959, 1992.

[74] D. Beeman. Some multistep methods for use in molecular dynamics calculations. *Journal of Computational Physics*, 20(2):130–139, 1976.

[75] N. Troullier and J. L. Martins. Efficient pseudopotentials for plane-wave calculations. *Physical Review B*, 43(3):1993–2006, 1991.

[76] D. M. Ceperley and B. J. Alder. Ground state of the electron gas by a stochastic method. *Physical Review Letters*, 45(7):566–569, 1980.

[77] J. P. Perdew and A. Zunger. Self-interaction correction to density-functional approximations for many-electron systems. *Physical Review B*, 23(10):5048–5079, 1981.

[78] V. Eyert. A comparative study on methods for convergence acceleration of iterative vector sequences. *Journal of Computational Physics*, 124(2): 271–285, 1996.

[79] M. L. Cohen and J. R. Chelikowsky. *Electronic Structure and Optical Properties of Semiconductors*, volume 75 of *Springer Series in Solid-State Sciences*. Springer-Verlag, Berlin, 2nd ed. edition, 1989.

[80] M. M. G. Alemany, M. Jain, L. Kronik, and J. R. Chelikowsky. Real-space pseudopotential method for computing the electronic properties of periodic systems. *Physical Review B*, 69(7):075101, 2004.

[81] A. Natan, A. Benjamini, D. Naveh, L. Kronik, M. L. Tiago, S. P. Beckman, and J. R. Chelikowsky. Real-space pseudopotential method for first principles calculations of general periodic and partially periodic systems. *Physical Review B*, 78(7):075109, 2008.

[82] H. J. Monkhorst and J. D. Pack. Special points for Brillouin-zone integrations. *Physical Review B*, 13(12):5188–5192, 1976.

[83] M. J. Gillan, D. Alfè, and A. Michaelides. Perspective: How good is DFT for water? *The Journal of Chemical Physics*, 144(13):130901, 2016.

[84] A. P. Thompson, H. M. Aktulga, R. Berger, D. S. Bolintineanu, W. M. Brown, P. S. Crozier, P. J. in 't Veld, A. Kohlmeyer, S. G. Moore, T. D. Nguyen, R. Shan, M. J. Stevens, J. Tranchida, C. Trott, and S. J. Plimpton. LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales. *Computer Physics Communications*, 271:108171, 2022.

[85] L. Ruiz Pestana, N. Mardirossian, M. Head-Gordon, and T. Head-Gordon. Ab initio molecular dynamics simulations of liquid water using high quality meta-GGA functionals. *Chemical Science*, 8(5):3554–3565, 2017.

[86] M. Chen, H.-Y. Ko, R. C. Remsing, M. F. Calegari Andrade, B. Santra, Z. Sun, A. Selloni, R. Car, M. L. Klein, J. P. Perdew, and X. Wu.

Ab initio theory and modeling of water. *Proceedings of the National Academy of Sciences*, 114(41):10846, 2017.

[87] K. M. Glassford and J. R. Chelikowsky. Structural and electronic properties of titanium dioxide. *Physical Review B*, 46(3):1284–1298, 1992.

[88] B. E. Tegner and G. J. Ackland. Pseudopotential errors in titanium. *Computational Materials Science*, 52(1):2–6, 2012.

[89] C. L. Reis, J. M. Pacheco, and J. L. Martins. First-principles norm-conserving pseudopotential with explicit incorporation of semicore states. *Physical Review B*, 68(15), 2003.

[90] 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 *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '19, New York, NY, USA, 2019. Association for Computing Machinery.