OPTIMIZED IMPLEMENTATION OF COUPLING MATRIX COMPUTATION IN TIME DEPENDENT DENSITY FUNCTIONAL THEORY

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

SHIVARAJU GOWDA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

October 2004

Acknowledgements

I am indebted to my advisor Prof. Yousef Saad for his guidance, advice and support throughout my graduate studies. I am also thankful to Prof. James Chelikowsky and Associate Prof. George Karypis for their acceptance to serve as the members on the examination committee.

Special thanks are due to Emmanuel Lorin and Murilo Tiago for being a constant source of information for physics part of the implementation. I would also like to thank Manuel Alemany, Russell Burdick and Konstantinos Bekas for providing the related materials and information for this thesis. I am also very grateful to Mitsuyasu Hanamura and Cormac Garvey of NEC Solutions (America), Inc. for being supportive during the writing of this thesis.

Finally, I would like to thank my family and all my friends for their love and friendship, which makes this work all worthwhile.

Contents

List of Tables

List of Figures

Abstract

One of the main objectives of modern computational materials science is the *ab initio* prediction of various material properties. Density Functional Theory (DFT) [23] along with *ab initio* pseudopotentials is generally used for predicting steady-state material properties. For the prediction of time-dependent material properties such as electronic excitations in general and optical properties in particular, *Time-Dependent* Density Functional Theory (TDDFT) [25, 24] formalism was developed and applied successfully to a wide range of atoms and molecules. The TDDFT method requires the computation of a coupling matrix $\mathbf{K}$, which describes the response of the self-consistent field to changes in charge density. The computation of the coupling matrix scales as $\mathcal{O}(n^5)$, where n is the total number of atoms in the system

In this thesis, we present an efficient implementation of the computation of the coupling matrix arising in TDDFT. The two important aspects involved : Solution of Poisson equation and the assembly of the coupling matrix, are investigated in detail and proper approximations are used in order to reduce the complexity. Poisson equation is computed in the reciprocal space and bounded support of the wave functions are used in the numerical integration. Experiments show the new implementation gains an order of magnitude in time when compared with the previous Real-space implementation. The method is tested to compute optical spectra of realistic systems. Details of the formalism and implementation are provided and performance comparisons with the Real-space implementation are reported. The performance of two modern computer architectures (Power4 and Itanium2) are also assessed using the present implementation as a benchmark.

Chapter 1

Introduction

Computational science has evolved in the last few years as a discipline by itself alongside experiment and theory. It involves simulating various properties of nature by using the processing power of a modern computer. Computational science is an interdisciplinary branch of science which encompasses computer science, mathematics and physics. Although a computational scientist might not be an expert in all of these branches, he is required to be thorough in computer science aspects and have a fairly good knowledge of mathematics and physics involved in the algorithms he implements. This awareness increases the efficiency, modularity and extensibility of the code he implements.

The size of the problem studied in many branches of science is limited by the computation time to simulate it on a computer. Two main factors affect the computation time: algorithm used and speed of the processor. As the processor power increases the size of problems handled has also increased proportionally. There is an abundance of problems which are forever in need of computational power. One such branch is computational materials science. Some of the problems in this field take days and sometimes weeks to complete on the most modern supercomputers available today. As a student of scientific computation, in this thesis, I try to apply better algorithms and the latest available tools to push the limits of the size of the problems solved in one particular application in computational materials science. The application we have considered is the coupling matrix construction in Time Dependent Density Functional Theory.

Before we deal with the coupling matrix construction from the next chapter, we discuss in the remaining parts of this chapter the general characteristics of the application and the way we intend to handle them. The following points can be made about the problem at hand.

1. Approximations are key to solving the problem with the available processor speed. These approximations are accepted as long as appropriate

justifications can be made.

2. A regular three-dimensional grid with constant grid spacing is used.
3. The problem can be solved in parallel and thus parallel platforms can be used.
4. Fortran 77 is the computer language of choice (because of the available legacy code).
5. It involves the use of Fast Fourier Transform(FFT).
6. It involves many vector operations which can make use of highly tuned math libraries.

With these aspects in mind we discuss in the following sections the software and the hardware issues for our implementation.

1.1 Software Considerations

1.1.1 Approximations

Approximations are key to handling many of the problems in the real world. With some of the problems scaling as $\mathcal{O}(n^5)$, where n is the number of atoms, it is impossible to even simulate a mole of a matter(6.02×10^{23} atoms) on available supercomputers without approximations. Therefore in order to validate theoretical science with experimental data, approximations are inevitable. The framework for allowing the user to set the desired approximation is required in which the user can turn off the approximations totally if needed or else he can set appropriate approximations to speedup the calculations.

1.1.2 Array storage

With the domain in three dimensional space, the arrays used to store the data can either be three dimensional or one dimensional, both having their advantages and disadvantages. The use of three dimensional arrays increases code readability and is also easy to implement. However, this type of array storage is difficult for the compiler to optimize. The compiler can generate a highly optimized code for the one dimensional arrays (there is a marked difference in performance when compared to three dimensional arrays). However, one dimensional arrays require non-intuitive index manipulations and this makes the code hard to read and to debug. A compromise can be struck by using the three dimensional array in the setup phase and reformatting the three dimensional array to a one dimensional array for computationally intensive parts (which can be found by profiling the code).

1.1.3 Computer Language

The three main languages available for scientific computation are Fortran, C or C++. The scientific community has long used Fortran 77 and for the most part it is slow to move on to C or C++. This is understandable considering the dependence on legacy codes, the ease of use and the learning curve required for C or C++. However, in the recent past researchers have considered Fortran90, which is a extended version of Fortran 77 with dynamic memory allocation and improved array handling features. One of the key features of Fortran 90 is that the indexes to arrays can range from any user specified values, unlike Fortran77 and C. Nevertheless, with so many implicit features in Fortran90, the compilers are not fool-proof. One particular example, would be the one we faced during the passing of array to ddot(dot product) subroutine of ESSL using Fortran90. If the array x is passed as *x(:)* to the subroutine ddot when compared to only x, there is a degradation in performance by 8 to 10 times. This performance gap was observed on IBM xlf90 compilers. For our implementation we initially used C++ and then moved on to Fortran 90(Reused the real space code, written in Fortran 77 and extended it).

1.1.4 FFT Libraries

For small and medium sized problems, the open source FFTw library [10] is competitive with respect to the performance of the vendor supplied libraries (ESSL for IBM, MKL for Intel) for Fast Fourier Transforms. FFTw and MKL place no restrictions on the size of the FFT transformation. However the ESSL library requires that the size of the transformation as well as the stride of the input vector be one of a predefined set of values. ESSL and MKL also require the input array to be in one dimension. Even though the vendor supplied libraries, ESSL and MKL are faster than FFTw, the transformation of the data to one-dimensional array offsets the advantage. For our implementation, we intend to setup the FFT part of the computation in a separate module so that the user can choose between the implementation depending upon the platform he is working on. We plan to implement the code using all the three libraries and allow the user to switch between the libraries by simply making minor changes to the Makefile and recompiling the code.

1.1.5 Parallelization

The computational materials science problem we intend to implement is embarrassingly parallel. Static decomposition of the work didn't show good scalability because of the dependency of computation time on two independent variables. Therefore, we had to consider dynamic allocation of work with "Master-Slave" approach. The previous, Real-space implementation had already implemented this scheme efficiently. We reused the scheme with some modifications. We used the vendor supplied MPI for the implementation.

1.1.6 Math Libraries

Most hardware vendors provide highly tuned math libraries for their platforms. Use of these libraries can fetch substantial savings in CPU time. IBM provides ESSL math libraries and Intel provides MKL math libraries. We have tried to make use of these libraries wherever possible.

1.2 Hardware Platforms

The algorithms in computational materials science are among the most floating point intensive in scientific applications. As mentioned before some large problems take days and sometime weeks to execute. With this trend, optimization of the code is very important, but also important is the machine on which this applications are executed. In the High Performance Computing (HPC) community, 64 bit architecture is the recent trend. 64-bit computing not only has more address space than 32-bit(Max 4GB) but also as several other characteristics that are of advantage to the HPC users. There is more to performance of machine than only the processor speed: the memory latency, memory bandwidth and the I/O. The recent 64-bit platforms are trying to address all these issues as well.

The characteristics of the algorithm we implement is typical of many other computational material science problems. In this thesis we intend to investigate the performance of the two most recent computer architectures IBM power4 and Intel Itanium2 using the implemented code as the benchmark.

1.2.1 IBM Power4 Architecture

We use an 8 processor IBM p655 to evaluate the performance of Power4 system [31]. The Power4 system has the traditional RISC architecture with dual core. The Power4 processor has a clock speed of 1.5GHz. Each processor core has an independent L1 cache. Two processors share an on chip L2 cache of size 1.4MB. The size of the L3 cache is 32MB(not on the chip). Four of the dual core chips are interconnected to form an SMP node with 8 processors. The memory bandwidth is a bit over 11 GBytes/sec. The IBM Power4 processor has two floating-point units capable of executing one FMA per cycle. There are 32 general purpose and 32 floating point registers.

1.2.2 Intel Itanium2 Architecture

We use 16 processor SGI Altix 350 to evaluate Intel Itanium2 64-bit microprocessor. Each Itanium2 (Madison) processor has a clock speed of 1.5GHz and 6MB integrated L3 cache [32]. The main characteristics of the Itanium2 are its enormous resources on the chip to effectively use the Instruction Level Parallelism(ILP). It has 128 floating point registers and several other features for effective speculation. The main drawback being the front-side bus. Pairs of Itanium2 processors access memory through a shared 16B-wide 400MHz front-side bus (FSB), which delivers a peak bandwidth of 6.4GB/s. These pairs of processors are connected to the whole system via 2 NUMAlink connectors. SGI 350 is a shared memory machine with 32 GB memory and running on Red-Hat-compatible Linux which is referred to as SGI ProPack.

The problem of interest is the computation of coupling matrix arising in Time-Dependent Density Functional Theory. The goal is to push the limits of the problem sizes that can be solved at present. The computational material science problem is defined in Chapter 2. Chapter 3 describes the details of the implementation. In Chapter 4 we validate the method used by comparing it with the previous implementation and further list the progress made with this implementation. Chapter 5 lists the conclusions along with some recommendations for future work.

Chapter 2

Time-Dependent Density Functional Theory

A successful approach that is common in electronic structures calculations is to use *ab initio* pseudopotentials within density functional theory (DFT) [23]. This approach has been used to predict mechanical, chemical, and electronic properties of many classes of solids, liquids, molecules, and more. In density functional theory, the original N-electron problem is converted into an effective one-electron problem, where all non-classical electron interactions (namely, exchange and correlation) are replaced by an additive one-electron potential that is a functional of the charge density. While this mapping is formally exact, it is approximate in practice because the exact functional is unknown. A common approximation to this functional, which is used in this paper, is the Local Density Approximation (LDA) where the exchange-correlation functional is defined as a local function of the charge density. A second important approximation made in this is approach is the use of pseudopotentials. In pseudopotential theory, only valence electrons, the ones forming chemical bonds, are treated explicitly. The effect of core electrons is accounted for by replacing the true atomic potential with an effective "pseudopotential". The pseudopotential is a smooth, slowly-varying potential, whereas the true atomic potential is rapidly varying and has a $\sim 1/r$ singularity at the nuclear position. In addition, the removal of core electrons from the picture, results in a much smaller number of eigenvectors to compute.

However, because the traditional time-independent DFT is inherently a ground-state theory [23] it is not suitable for the computation of electronic excitations in general, and optical properties in particular. A generalized, *time-dependent* DFT (TDDFT) formalism has been developed which allows the computation of excited state properties. This approach was applied successfully to a wide range of atoms and small molecules [24, 25, 1, 2]. Optical properties of larger systems of molecules, clusters, or “quantum dots” (small fragments of bulk material) in the range of many hundreds of atoms are much harder to address despite their outstanding importance in physics. This is primarily due to severe computational limitations.

The recent article [2] demonstrated how this pseudopotential-TDDFT formalism can enable the calculation of properties for systems with several hundreds of atoms. Parallel computing was a major part of the success of this method, and so enhancements of the parallel efficiency of the method undertaken in the article [1] resulted in significant additional gains. The parallel code in these two articles uses a real-space implementation of a frequency-domain formulation of the TDDFT equations.

At present, the main computational problem in implementing the TDDFT approach lies in the computation of a coupling matrix K. This matrix is dense and symmetric and its dimension n_K is of the order of square of the number of states considered. Each row of the matrix requires the solution of a Poisson equation. In [2] and [1] these equations were handled by a preconditioned Conjugate Gradient method. Most of the time to complete the calculation is spent in solving these systems.

This chapter is organized as follows. In section 2.1 we present the formalism along with a direct method to compute the coupling-matrix based on a simple planewave approach. The advantage of this method is that it exploits features of the physical and mathematical problems. In Section 2.2 we look at the simplified formulation of the coupling matrix. Section 2.3 explains the implementation in [1] using Real-space method.

2.1 Formalism

We follow [2] and [1] and use the frequency-domain formulation of TDDFT as discussed by Casida [25]. In the following we give a brief background of the method. Details and physical justifications may be found in [25, 2, 22, 4].

The procedure begins with the solution of the time-independent Schrödinger equation, using the pseudo-potential DFT formulation [23] in the domain Ω,

$$\left(-\frac{\triangle}{2} + \sum_{R_a} V_{ps}(\mathbf{r} - \mathbf{R}_a) + V_H(\rho(\mathbf{r})) + V_{xc}(\rho(\mathbf{r}))\right) \psi_n(\mathbf{r}) = \epsilon_n \psi_n(\mathbf{r}). \quad (2.1)$$

In the above equation, ψ_n and ϵ_n denote the nth single-electron eigenfunction and eigenvalue respectively. The charge density, denoted by $\rho(\mathbf{r})$, is the sum of squares of the eigenfunctions $\psi_n(r)$ associated with the occupied states. Finally, V_H is the Hartree (electron-electron) interaction potential, V_{ps} is the pseudo-potential which represents the interactions between ions located in $\mathbf{R}_a$ and V_{xc} is the exchange-correlation potential. Once the above equation is solved for the eigenvalues, eigenvectors and occupied states, the coupling matrix K is assembled in the form:

$$K_{ij,kl} = 2\int_\Omega\int_\Omega \bar{\psi}_i(\mathbf{r})\psi_j(\mathbf{r})\left(\frac{1}{|\mathbf{r}-\mathbf{r}'|} + \frac{dV_{xc}(\rho(\mathbf{r}))}{d\rho(\mathbf{r})}\delta(\mathbf{r}-\mathbf{r}')\right)\psi_k(\mathbf{r}')\bar{\psi}_l(\mathbf{r}')d\mathbf{r}d\mathbf{r}' \quad (2.2)$$

where Ω the physical domain, which a subset of $\mathbb{R}^3$. Given this, the optical absorption spectrum of a system of molecules is computed as,

$$QF_I = \omega_I^2 F_I, \quad (2.3)$$

where

$$Q_{ij,kl} = \delta_{ik}\delta_{jl}\omega_{kl}^2 + 2\sqrt{\lambda_{ij}\omega_{ij}}K_{ij,kl}\sqrt{\lambda_{kl}\omega_{kl}} \quad (2.4)$$

with $\lambda_{ij} = f(\epsilon_i) - f(\epsilon_j)$ the difference between occupied states (f denotes the Fermi-Dirac function), $\omega_{ij} = \epsilon_i - \epsilon_j$. The term ω_{ij} is directly related to the optical transition energy.

The eigenvalues and eigenfunctions of the matrix $Q(\omega)$ are then computed and, finally, the strengths of the absorption associated with the transition energy ω_I are obtained as follows:

$$f_I = \frac{2}{3} \sum_{\beta=\{x,y,z\}} |B_\beta^T R^{1/2} F_I|^2. \tag{2.5}$$

Here, $R_{ij,kl}(\omega) = \delta_{ik}\delta_{jl}\lambda_{kl}\omega_{kl}$ and $(B_\beta)_{ij} = \int \psi_i^*(\mathbf{r})\beta\psi_j(\mathbf{r})d\mathbf{r}$ for the three Cartesian directions $\beta = \{x, y, z\}$.

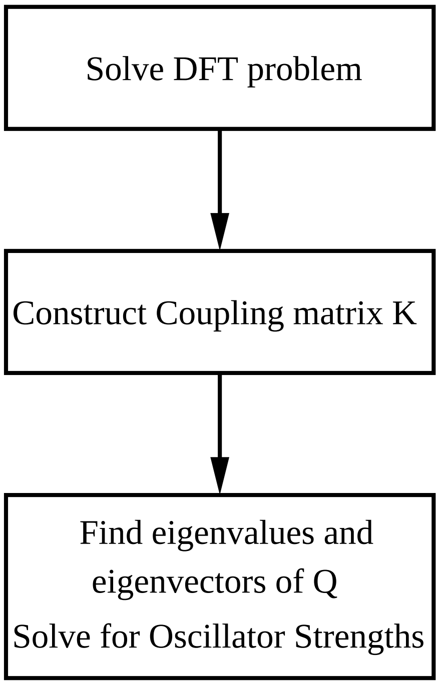

Solve DFT problem
Construct Coupling matrix K
Find eigenvalues and eigenvectors of Q
Solve for Oscillator Strengths

Figure 2.1: Flowchart for finding oscillator strengths using TDDFT

This formalism can be divided into three computational blocks as shown in Fig. 2.1. The first block computes the DFT solution to give the eigenvalues, occupation states and wave functions. Then the coupling matrix is constructed using the output of the first block. Finally the eigenvalues and eigenvectors of the dense matrix Q are determined to compute the optical absorption spectrum of the system. The three block are implemented in separate programs with the intermediate outputs written to a file in binary format, which is subsequently read as an input in the next block. All the three blocks are implemented in parallel.

The pseudo-potential-DFT equation 2.1 is implemented on real-space grid using finite difference approach. The implementation has been discussed extensively with respect to underlying physics in [2] and computer science aspects in [1]. In the next section we will discuss the implementation of the coupling matrix construction. The third block in the Fig. 2.1 is the calculation of the eigenvectors and eigen values of the $Q_{ij,kl}$ matrix. $Q_{ij,kl}$ is symmetric, dense and real matrix. Its dimension is of the order of product of number of filled states and number of empty states. SCALAPACK [13] is used for computing its eigenvalues and eigenvectors in parallel.

2.2 Coupling matrix construction

The evaluation of the coupling matrix given in Eq. (2.2), is the most expensive part of a TDDFT calculation. Computing the matrix K consists of evaluating a large number of entries, each of which is very costly to compute.

Equation (2.2) consists of a sum of two integrals which can be computed separately. The first, double integral can be rewritten as:

$$\int_\Omega\int_\Omega \bar{\psi}_i(\mathbf{r})\psi_j(\mathbf{r})\frac{1}{|\mathbf{r}-\mathbf{r}'|}\psi_k(\mathbf{r}')\bar{\psi}_l(\mathbf{r}')d\mathbf{r}d\mathbf{r}' = \int_\Omega d\mathbf{r}'\psi_k(\mathbf{r}')\bar{\psi}_l(\mathbf{r}')\int_\Omega d\mathbf{r}\frac{1}{|\mathbf{r}-\mathbf{r}'|}\bar{\psi}_i(\mathbf{r})\psi_j(\mathbf{r})\ . \tag{2.6}$$

If we define

$$\rho_{ij}(\mathbf{r}) \equiv \bar{\psi}_i(\mathbf{r})\psi_j(\mathbf{r}), \tag{2.7}$$

and

$$\Phi_{ij}(\mathbf{r}') \equiv \int d\mathbf{r}\frac{1}{|\mathbf{r}-\mathbf{r}'|}\bar{\psi}_i(\mathbf{r})\psi_j(\mathbf{r}), \tag{2.8}$$

then we see that

$$\Phi_{ij}(\mathbf{r}') = \int \frac{\rho_{ij}(\mathbf{r})}{|\mathbf{r}-\mathbf{r}'|}d\mathbf{r}. \tag{2.9}$$

With appropriate boundary conditions, this is well-known to be the solution of the Poisson equation:

$$\nabla^2 \Phi_{ij}(\mathbf{r}') = -4\pi \rho_{ij}(\mathbf{r}'). \tag{2.10}$$

Note that ρ_{ij} is not a physical charge density and Φ_{ij} is not a physical potential. However, they do have charge and potential units, respectively, and obey the same Poisson relation as a “true” charge density and Coulomb potential do. Solving the Poisson equation 2.10 using some effective solvers, may be more economical than evaluating Φ_{ij} from the evaluation of the integral Eq. 2.9 by a “direct summation”. There is indeed a wide variety of fast Poisson solvers available for solving Eq. 2.10 on regular meshes.

Once $\Phi_{ij}(\mathbf{r}')$ is found by solving the Poisson equation, we end up with the following form of Eq. (2.2):

$$K_{ij,kl} = 2\int_\Omega d\mathbf{r}' \rho_{lk}(\mathbf{r}')\Phi_{ij}(\mathbf{r}') + 2\int_\Omega d\mathbf{r} \rho_{ij}(\mathbf{r}) \frac{dV_{xc}[\rho(\mathbf{r})]}{d\rho(\mathbf{r})} \rho_{lk}(\mathbf{r}) \tag{2.11}$$

The two single integrals in Eq. (2.11) must be computed by direct summation but this can be done for both of them at the same time:

$$K_{ij,kl} = 2\int_\Omega d\mathbf{r} \Big[\Phi_{ij}(\mathbf{r}) + \rho_{ij}(\mathbf{r}) \frac{dV_{xc}[\rho(\mathbf{r})]}{d\rho(\mathbf{r})}\Big] \rho_{lk}(\mathbf{r}). \tag{2.12}$$

In practice, the computation of K is performed row-wise, and consists of two phases. First, $\Phi_{ij}(\mathbf{r}')$ is obtained from the solution of Eq. (2.10). Then, the integral Eq. (2.12) is evaluated. Eq. (2.10) must be solved only *once per row*, i.e., once per pair i, j, whereas Eq. (2.12) needs to be evaluated for each matrix entry, i.e., for each pair of pairs of indexes $\{(i, j), (k, l)\}$. Fig. 2.2 illustrates the assembly of the coupling matrix.

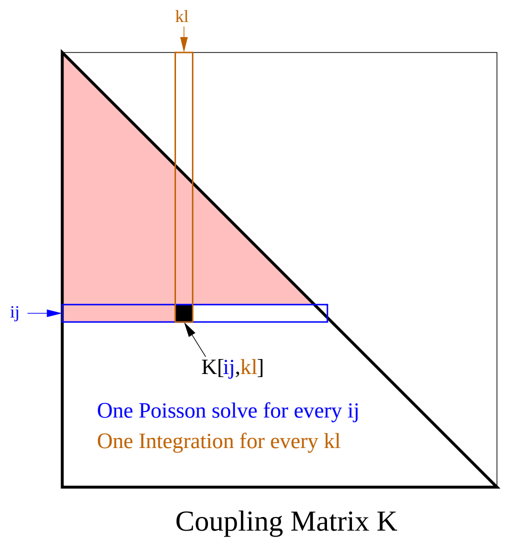


Figure 2.2: Construction of the coupling matrix

The dimension $kdim$, of the coupling matrix is of the order of square of the number of states considered. The coupling matrix is symmetric, so only the lower triangular part is assembled. The calculation of each row is independent of the others, so it can be easily parallelized. Wave functions are localized in Real-space, i.e., they have a small support and this property can be exploited to reduce the cost of computing the integral in Eq. (2.11).

2.3 Real-Space Implementation

The Real-Space Implementation of the coupling matrix construction uses the same real-space mapping as used in the solution of the pseudopotential-DFT equation - Eq. (2.1). The real-space mapping is shown schematically in Fig. 2.3. Both potentials and wave functions are set up on a simple Cartesian three-dimensional grid within a spherical domain. The grid points inside the sphere are described by their discrete space coordinates, (x, y, z). Outside the boundary domain, wave functions are required to vanish. The Laplacian term in Eq. (2.10) is approximated by a high-order finite-difference expansion, which replaces spatial derivatives with a weighted sum of the wave-function values at neighboring grid points:

$$\begin{aligned}\nabla^2 \psi_i(x,y,z) &= -\frac{1}{h^2}\sum_{n=-N}^{N} C_{N,n}\Big[\psi_i(x+nh,y,z)+ \\ &\qquad \psi_i(x,y+nh,z)+\psi_i(x,y,z+nh)\Big], \qquad (2.13)\end{aligned}$$

where h is the grid spacing and $C_{N,n}$ are the coefficients in the order N finite difference expansion for the second derivative [30]. In the context of Eq. (2.13), two points are defined as neighbors if the vector connecting them is parallel to one of the three major axes and the points are up to N grid points away from each other. For a given accuracy of calculation, the finite-difference order should be chosen as a compromise between having a fine grid and a large but very sparse Hamiltonian matrix, and a coarse grid with a smaller but less sparse Hamiltonian. The Real-space code used N=6 for the implementation.

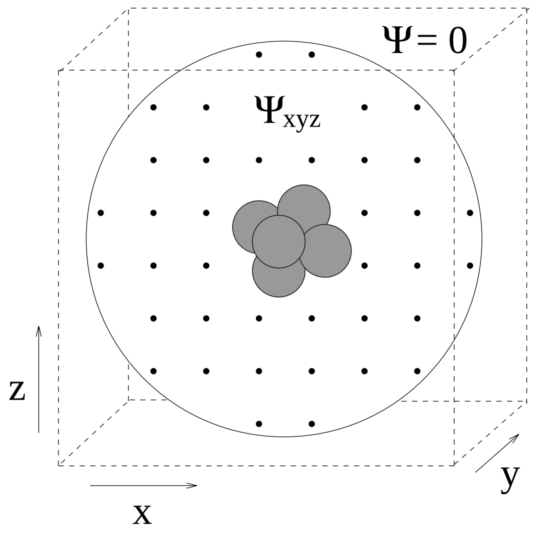


Figure 2.3: Schematic illustration of the boundary domain for a cluster

The appropriate boundary conditions on the surface of the sphere were setup using the multipole expansions. Once the boundary conditions were set up, the Poisson equation was solved using Preconditioned Conjugate Gradient method. The remaining part of the construction, which involves the integration part was done as described in Section 2.2. The Real-space implementation used “Master-Slave” model for parallelization and numerical results showed excellent scalability. For the $Si_{34}H_{36}$, quantum dot problem, the Real-space implementation took a total of 15:30 hrs on 8 processors. Out of this, 6 hrs were spent in the solution of the Poisson equation.

The optimized implementation of Real-space code in [1] was very efficient in terms of CPU time as well as scalability. The code showed linear scaling and reduced the CPU time by a factor of five when compared to an earlier implementation. The largest system the Real-space code investigated was $Si_{147}H_{100}$. However, this system is still not in the range of the available data for experimental comparisons. In order to use the method for larger systems and thus compare with the experimental data of those systems and validate the method, we need a faster implementation. It would also be beneficial to have a set of parameters to toggle between accuracy and speed, so that we can get a rough estimate within limited time and then switch the parameter for more accuracy. Our implementation attempts to accomplish this. Chapter

3, analyzes the construction of the coupling matrix first and then describes the new implementation. In Chapter 4 numerical examples of several clusters are presented to validate the new implementation and test the performance of the new code.

Chapter 3

Implementation

3.1 Introduction

Before we move on to the implementation part it would be worthwhile to discuss the main characteristics of the coupling matrix K and the drawbacks of the Real-space implementation which can be rectified for a better performance. Since we intend to use plane wave based methods for the present implementation, from now onwards we refer to the new implementation as Fourier-space implementation. The main characteristics of the coupling matrix $\mathbf{K}$ can be summarized as follows:

1. The order of $\mathbf{K}$ is equal to the number of transitions considered.

	Real-space implementation	**Fourier-space implementation**
Poisson equation solver	Preconditioned Conjugate Gradient (PCG) method	FFT based methods
Parallel implementation	"master-slave" model	Reuse the Real-space model
Language	Fortran 77	Fortran 90
Approximations	No approximations	Framework in which user can set the desired approximations
Math libraries	Doesn't significantly make use of math libraries	Use math libraries for all computational loops

Table 3.1: Comparison of the intended development of the new implementation with the Real-space implementation

2. **K** is symmetric, thus only lower(or upper) half of the matrix needs to be constructed.

3. Calculation of each row of **K** is independent of others, thus construction of **K** is embarrassingly parallel (if implemented correctly).

4. Construction of every row of **K** requires a solution of a Poisson equation. Since a complete row is assembled on a single processor, parallel implementation of the Poisson equation solver is not required.

5. Wave functions are localized in Real-space i.e., have small support (sparse). This property can be used in approximations while integrating and also can be used to save on memory space.

Table 3.1. lists the major changes we intend to use to improve the performance when compared to the Real-space implementation. The Preconditioned Conjugate Gradient (PCG) method used in the Real-space implementation was slow in convergence, required application of boundary conditions, and furthermore had no predefined time for solution. In the Fourier space implementation we plan to take advantage of the efficient implementation of the vendor supplied FFT routines in constructing the matrix. The parallel implementation of the Real-space code was very effective and showed linear scalability, we plan to use the same model for the Fourier space implementation. Fortran 77 was used for the Real-space implementation, which had several drawbacks. The most important one being the lack of dynamic memory allocation, this caused the user to reset the parameters and recompile the code every time the problem size was increased. We intend to use Fortran 90 and make use of its dynamic memory allocation and multi-dimensional array features. The multi-dimensional array capabilities and user specified indexing of the arrays of Fortran90 will be very helpful in manipulating the three dimensional arrays. These capabilities also simplify the programming and increase readability of the code. However, there are some reported problems for manipulation of arrays in Fortran90. We plan to avoid them by using

	Grid	Grid spacing	Radius	nstate	nocc	kdim
Si_5H_{12}	$33 \times 33 \times 33$	0.75 a.u	12 a.u	30	16	224
$Si_{10}H_{16}$	$33 \times 33 \times 33$	0.75 a.u	12 a.u	60	28	896
$Si_{34}H_{36}$	$65 \times 65 \times 65$	0.75 a.u	24 a.u	240	86	13244

Table 3.2: Description of the test cases for Si_nH_m clusters. *nstate* stands for total number of states considered. *nocc* stands for number of occupied states and *kdim* for the dimension of the coupling matrix.

one dimensional arrays in the hot-spots of the code. As reported earlier in this chapter, the wave-functions have localized support and therefore during integration efficient techniques can be used to integrate only on this domain. However, the implementation of such techniques may require more memory space. A compromise may be struck between the memory space and speed by using approximations and letting user control the parameter. Math libraries are highly tuned and very effective in reducing the total elapsed time. We intend to formulate the code so that we can leverage its benefits.

In order to explore the different schemes and evaluate their significance we use three test cases : Si_5H_{12}, $Si_{10}H_{16}$ and $Si_{34}H_{36}$. Si_5H_{12} is a well localized system, whereas $Si_{10}H_{16}$ is on the same grid as Si_5H_{12} and represents systems that are not significantly localized. $Si_{34}H_{36}$ is quantum dot problem of $Si_{35}H_{36}$ and used as a benchmark for comparing with the Real-space code.

The chapter is organized as follows, First we investigate the scalability of the coupling matrix construction. In the section 3.3, we investigate an effective method for solution of Poisson equation in Fourier space. Then in section 3.4 we look into the modifications of coupling matrix assembly procedure in order to reduce the complexity of integration and in section 3.5 the parallel implementation is described.

3.2 Scalability Analysis

As seen in the previous chapter, the two main parts of the coupling matrix construction are

- Solution of Poisson equation
- Integration over the domain

The solution of Poisson equation is performed for every row of the coupling matrix and the integration is performed for every element of the coupling matrix. Let $kdim$ denote the number of transitions in the system(which is also the dimension of the coupling matrix) and $ndim$ denote the number of grid points in the system. Then the total cost of computing the coupling matrix is of the form:

$$C_1 kdim^2 \cdot ndim + C_2 kdim \cdot ndim \cdot \log(ndim), \tag{3.1}$$

where C_1 and C_2 are constants to indicate the cost of the direct summation and solution of Poisson equation respectively.

If n is the total number of atoms in the system, $kdim$ scales somewhere between n and n^2 depending on the choice of "trustworthy" energy window. If a relatively fast solution is desired with some compromise in accuracy then fixed number of empty states can be used in which case $kdim$ scales as n. But for more accurate solution we should use a fixed ratio of empty to filled states. In this case $kdim$ scales as n^2. $ndim$ scales roughly as the number of atoms in the system n. So in the worst case, if $kdim \approx \alpha n^2$ and $ndim \approx \beta n$ the algorithm scales as:

$$C_1 \alpha^2 \beta n^5 + C_2 \alpha \beta n^3 \cdot \log(\beta n); \tag{3.2}$$

In the best case, $kdim \approx \alpha n$ and the algorithm scales as:

$$C_1 \alpha^2 \beta n^3 + C_2 \alpha \beta n^2 \cdot \log(\beta n). \tag{3.3}$$

In summary, the coupling matrix calculation scales somewhere between $\mathcal{O}(n^5)$ and $\mathcal{O}(n^3)$, where n is the total number of atoms in the system. For investigating the efficient implementation of the coupling matrix construction, we consider *kdim* as constant(we allow physicists to modify it) and concentrate on the terms $C_1 ndim$ and $C_2 ndim.log(ndim)$ in Eq.(3.1). The former term corresponds to the integration part and later term corresponds to the solution of Poisson equation.

For solution of Poisson equation we intend to use Fourier-space method to gain in speed by using Fast Fourier Transform(FFT) libraries. Although the complexity of the FFT based solution of the Poisson equation is similar to the conjugate gradient method, the constant C_2 varies substantially between both methods in favor of FFT based method as will be seen later. So in the next section we investigate the FFT methods available for the solution of Poisson equation for a finite localized system.

3.3 Solution of the Poisson Equation

In the Real-space code the solution of the Poisson equation constitutes the biggest part of the cost of the coupling matrix construction. Preconditioned

Conjugate Gradient(PCG) method was used to solve the Poisson equation.

The mesh we are dealing with is a three-dimensional grid with constant grid spacing and constant coefficients. The coupling matrix is built in parallel, but each row of the matrix is assembled in a single processor. So the Poisson equation corresponding to that row is solved on a single processor, thus no parallel implementation of the Poisson equation solver is required. As a result, we have the best configuration for the use of a "Fast Poisson Solver". A Fast Poisson Solver is any technique which can solve the Poisson equation in $\mathcal{O}(N \log N)$ operations where N is the number of points in the discretization.

Table 3.3 shows the complexity of different algorithms for solving Poisson equation. The class of "Real-space methods" works directly in physical space. The Poisson equation is discretized by, e.g., finite differences and the resulting linear system is solved using an iterative method or a multigrid technique. In contrast, planewave methods work in the Fourier (or reciprocal) space. In this case, the Poisson equation is trivially solved because the Fourier modes diagonalize the Laplacean.

In [1], a Real-space method with Preconditioned Conjugate Gradient (PCG) method was used to solve the resulting linear systems. Among the

Method	Complexity	Memory
Gaussian Elimination	N^3	N^2
PCG	$N^{3/2}$	$N^{1/2} log N$
FFT	$N log N$	$log N$
Multigrid	N	$log^2 N$

Table 3.3: Comparison of the computational complexity and memory requirements for several fast Poisson solvers

conclusions in [1] is the fact that the Conjugate Gradient is somewhat ill-adapted to the problem at hand. The discretization uses high-order finite differences. This has a negative impact on sparsity as well as on convergence. Preconditioning was not helpful as in other cases because the modest gain in the number of steps to converge was counterbalanced by the additional cost of applying the preconditioner. Multigrid methods might be a better choice, but there are two difficulties. The first comes again from the use of high order finite difference discretization – which the standard MG would have to be adapted. The second is a practical problem: a proper implementation would be complex due to nonuniform data access patterns and the use of additional memory.

Plane wave basis methods are attractive for computing Φ_{ij} since they exploit the fast Fourier transform. Discrete truncated sets of plane wave bases on the reciprocal lattice is a natural basis set for periodic systems. Hence, they are the preferred method in density functional theory (DFT) for infinite periodic systems such as bulk solids. This method is very popular because of the use of the highly optimized vendor supplied FFT routines. However, for finite localized systems that are not periodic, such as slabs or molecular clusters,the use of a discrete set of plane waves will implicitly generate unwanted periodic images of the cell being studied. Nevertheless, discrete plane wave basis sets are often used for studying finite systems because of the available efficient implementation of the FFT, and errors in the potential are usually ignored, or diminished by using a super-cell. However, this error might be substantial and might affect the equilibrium structure and dynamics of weakly bounded molecules or clusters. Several methods are proposed for treating this problem while at the same time using the computational speed of the FFTs. The spurious effects of the periodic cells can be removed and appropriate boundary conditions applied with some modifications. This usually involves additional computations and complications.

Several methods have been proposed to minimize or eliminate the effects of the periodic images for localized systems. Castro and Rubio [5] investigated cut-off method and multipole correction method and compared them to the real-space approach. The cut-off based methods consider a bigger cell and implement a cut-off between the values of the periodic cells. Multipole-correction methods, treat long range potential analytically and short range potential using FFT's. In this Section we describe the uncorrected planewave method first, and next we will discuss methods to remove the spurious interaction of the periodic images. The cluster we consider is on a spherical domain which is circumscribed by a cubic domain as shown in Fig. 2.3.

3.3.1 Computation of uncorrected Φ_{ij}

Consider the term Φ_{ij}, which is the solution of equation 2.10. The Fourier based Fast Poisson Solver is formally equivalent to applying a forward discrete Fourier transform (denoted by $\mathcal{F}$) followed by an inverse Fourier transform (denoted by $\mathcal{F}^{-1}$):

$$\Phi_{ij}(\mathbf{r}) = 4\pi\mathcal{F}^{-1}\ \left[\ \mathcal{F}(\Psi_i\bar{\Psi}_j)(\mathbf{k})/\|\mathbf{k}\|^2\ \right](\mathbf{r}). \tag{3.4}$$

If we denote $f^{cut} \longrightarrow \frac{1}{\|\mathbf{k}\|^2}$ then the above equation becomes:

$$\Phi_{ij}(\mathbf{r}) = 4\pi\mathcal{F}^{-1}\ \left[\ \mathcal{F}(\Psi_i\bar{\Psi}_j)(\mathbf{k}) \cdot f^{cut}\ \right](\mathbf{r}). \tag{3.5}$$

For the particular type of functions we have in TDDFT, the Fourier transform

$\mathcal{F}(\Psi_i\bar{\Psi}_j)(\mathbf{k}) = 0$, for $\mathbf{k} = 0$. Therefore the singularity at $\mathbf{k} = 0$ is treated by setting $f^{cut}(0) = 0$. f^{cut} doesn't depend on the wave-functions hence it can be computed once and stored as an array and subsequently used for all the rows of the coupling matrix. This is indicated in the Algorithm 3.3.1.

Algorithm 3.3.1: Poisson Equation Solution

```
 1: Compute $f^{cut} \longrightarrow \frac{1}{\|\mathbf{k}\|^2}$
 2: Start of Coupling Matrix Assembly Loop
 3: for all the transitions $ij$ considered do
 4:    $\rho_{ij} = \psi_i\psi_j$
 5:    C ——— Start Poisson Equation Solver —————-
 6:    Apply Forward Fourier Transform: $\mathcal{F}(\rho_{ij})(\mathbf{k})$
 7:    Multiply by $f^{cut}$ : $\mathcal{F}(\rho_{ij})(\mathbf{k})f^{cut}$.
 8:    Apply Inverse Fourier Transform: $\mathcal{F}^{-1}(\mathcal{F}(\rho_{ij})(\mathbf{k})f^{cut})$
 9:    C ——— End Poisson Equation Solver —————-
10:    Assemble the remaining elements of the row corresponding to $ij$
11: end for
12: End of Coupling Matrix Assembly Loop.
```

The method described above is well-suited for periodic system, for localized system the spurious interactions with periodic images are present and are dealt with in the coming sections. In terms of CPU-time the above method gives excellent performance when compared to PCG. It is still possible to further reduce the CPU-time by working on a reduced domain because of the observation made below.

Since wave-functions are functions with nearly bounded support, their Fourier transforms never vanish at infinity (Paley-Wiener theorem), i.e., they cannot theoretically be constrained to be in a bounded domain. However, since the wave-functions are sufficiently differentiable, and decrease to zero fast enough at infinity[1], their Fourier transform does decrease to zero at infinity. In practice, wave-functions have bounded support and they verify these assumptions.

Since in practice, $\Psi_i \bar{\Psi}_j$ are functions with bounded support, their Fourier transform is also such that:

$$\mathcal{F}(\Psi_i \bar{\Psi}_j)(\mathbf{k}) = \sum_{\mathbf{r}} e^{i\mathbf{k}.\mathbf{r}} (\Psi_i \bar{\Psi}_j)(\mathbf{r}) = \sum_{\mathbf{r} \,\in\, \mathrm{Supp}(\Psi_i \bar{\Psi}_j)} e^{i\mathbf{k}.\mathbf{r}} (\Psi_i \bar{\Psi}_j)(\mathbf{r}). \tag{3.6}$$

This remark can be exploited to reduce the domain on which to compute the Fourier transform, that is to reduce the frequency cut-off in the Fourier expansion. For the inverse Fourier transform the argument is slightly different. The inverse Fourier transform gives the Hartree potential (C is a positive constant):

[1]Typically, wave-functions belonging to a Schwartz space

$$\Phi_{ij}(\mathbf{r}) = C \sum_{\mathbf{k}} e^{-i\mathbf{k}.\mathbf{r}} \mathcal{F}(\Psi_i \bar{\Psi}_j)(\mathbf{k}) / \|\mathbf{k}\|^2. \tag{3.7}$$

As $\mathcal{F}(\Psi_i \bar{\Psi}_j)(\mathbf{k})$ is bounded (regularity of the wave-functions) and the general term $1/\|\mathbf{k}\|^2$ series converges, we can truncate the sum in 3.7 by dropping those terms with $\mathbf{k}$ for $\|\mathbf{k}\|$ large enough (say $\|\mathbf{k}\| > K_{\text{max}}$). Note that the orthogonality of the wave-functions implies that $\mathcal{F}(\Psi_i \bar{\Psi}_j)(\mathbf{0}) = \mathbf{0}$, so there is no division by zero when $\mathbf{k} = \mathbf{0}$ and $i \neq j$.

The result is that it is possible to work with a reduced box:

$$\Phi_{ij}(\mathbf{r}) \sim C \sum_{\|\mathbf{k}\| \leq K_{\text{max}}} e^{-i\mathbf{k}.\mathbf{r}} \mathcal{F}(\Psi_i \bar{\Psi}_j)(\mathbf{k}) / \|\mathbf{k}\|^2.$$

In practice one can reduce the domain on which to compute the fast Fourier transform: instead of working on the whole domain $\mathbf{\Omega}$, the computation of the Fourier transform and inverse Fourier transforms can be restricted to the sub-domain $\mathbf{\Gamma} \subset \mathbf{\Omega}$ obtained from enforcing the restriction $\|k\| \leq K_{max}$.

3.3.2 Long-range and short-range splitting

To circumvent the effect of interaction between the local cell and its neighbors, this method uses a technique described in [8] and called FALR (Fourier Analysis for Long-Range). FALR allows to treat separately the long-range and short-range parts of the potential. Specifically, the long-range part of the potential is treated analytically while the short-range part is treated using the Fourier transform. Formally, we solve the following systems:

$$\begin{cases} -\triangle \Phi_{ij}^{(\text{long})}(\mathbf{r}) = 4\pi \rho_{ij}^{(\text{long})}(\mathbf{r}), \\ -\triangle \Phi_{ij}^{(\text{short})}(\mathbf{r}) = 4\pi \rho_{ij}^{(\text{short})}(\mathbf{r}) = 4\pi \left(\rho_{ij}(\mathbf{r}) - \rho_{ij}^{(\text{long})}(\mathbf{r}) \right), \end{cases} \tag{3.8}$$

and then add the results, $\Phi = \Phi^{(\text{long})} + \Phi^{(\text{short})}$, to obtain the final solution. Because we treat only the short-range part by discrete Fourier transform, the interactions between the different copies of the potential can be reduced. The way to determine "analytically" the long-range part of the potential is to use a multipole expansion. Physical insights into the formulation can be found in [8]. Here we merely present the pertinent equations without proof so that we can analyze their computational implications.

Recall that we want to solve 2.10 knowing that the solution of this equa-

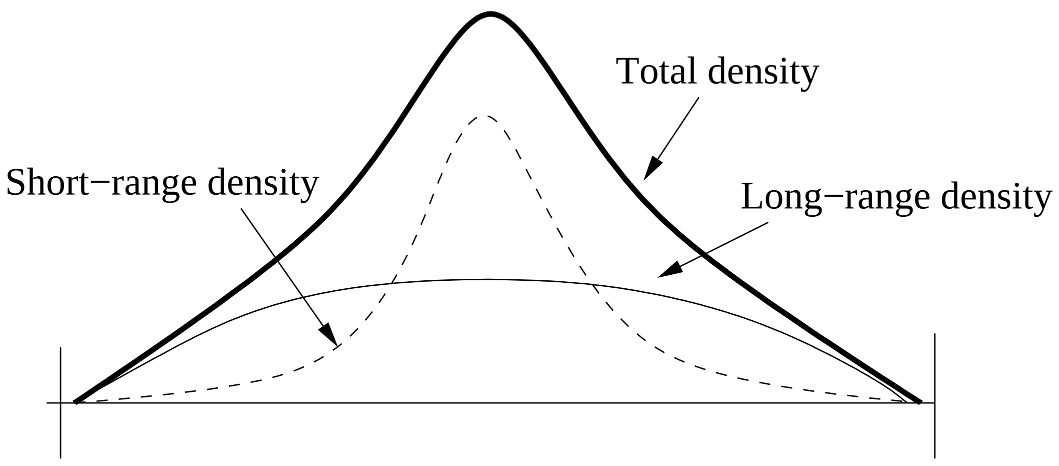


Figure 3.1: Long-range part and short range part of the density

tion outside the support of $\Psi_i\bar{\Psi}_j$ can be expressed in spherical coordinates by a multipole expansion. Denoting by q_{lm} the spherical multipole moments, Y_{lm} the spherical harmonics, and (r,θ,ϕ) the spherical coordinates, this expression is:

$$\Phi_{ij}(\mathbf{r}) = \sum_{l=0}^{l=\infty}\sum_{m=-l}^{m=+l} \frac{4\pi}{1+2l}\frac{q_{lm}}{r^{l+1}}Y_{lm}(\theta,\phi),\ \mathbf{r}\in\left(\mathrm{Supp}(\Psi_i\bar{\Psi}_j)\right)^{\mathrm{C}} \tag{3.9}$$

with:

$$q_{lm} = \int \rho_{ij}(\mathbf{r})Y_{lm}(\theta,\phi)\mathbf{r}^l d\mathbf{r} \tag{3.10}$$

and

$$Y_{lm}(\theta,\phi) = \sqrt{\frac{1+2l}{4\pi}\frac{(l-m)!}{(l+m)!}}P_l^m(\cos\theta)e^{im\phi}. \tag{3.11}$$

In the above expression, P_l^m are the standard Legendre polynomials.

As seen in Equation 3.9, $l \in [0,1,...,\infty)$, and the coulomb potential of the lth moment behaves as $1/r^{l+1}$. Thus, the long range parts are concentrated in the low-l components. To separate out the low multipole momenta we define a long range density denoted by $\rho_{ij}^{(\text{long})}$ which accounts for all low multipole momenta. In spherical harmonics long range density is given by :

$$\rho_{ij}^{(\text{long})} = \sum_{l=0}^{l=l_{cut}} \sum_{m=-l}^{m=+l} a_{lm}\rho_l(|\mathbf{r}|)Y_{lm}(\theta,\phi) \tag{3.12}$$

where ρ_l is a reference density which is loosely close to the actual density ρ_{ij}. [8] suggest one such reference density which is independent of ρ_{ij} and only dependent on the grid. Hence, the same ρ_l can be used for all ij's. a_{lm} are the expansion co-efficients such that $q_{lm}^{(long)} = q_{lm}$ for all $l \leq l_{cut}$. Therefore,

$$a_{lm} = \frac{q_{lm}}{4\pi \int drr^{2+l}\rho_l(r)} \tag{3.13}$$

Since we consider long distance interactions the charge density is almost punctual and spherical symmetry can be assumed. As a result the three-dimensional Poisson equation becomes a one-dimensional radial equation for each l component.

$$\Phi_l(|\mathbf{r}|) = -4\pi\rho_l(|\mathbf{r}|) \tag{3.14}$$

The long-range potential $\Phi_{ij}^{(\text{long})}$ is then expressed in spherical harmonics as follows:

$$\Phi_{ij}^{(\text{long})}(\mathbf{r}) = \sum_{l=0}^{l=l_{cut}} \sum_{m=-l}^{m=+l} a_{lm}\Phi_l(|\mathbf{r}|)Y_{lm}(\theta, \phi) \tag{3.15}$$

After having solved for long-range potential the short range density is obtained as

$$\rho_{ij}^{(short)} = \rho_{ij} - \rho_{ij}^{(long)} \tag{3.16}$$

Then the short-range potential $\Phi_{ij}^{(short)}$, is obtained by the regular uncorrected forward and inverse Fourier transforms as discussed in section 3.3.1. Finally, the two potentials are added to obtain the total potential for the system.

$$\Phi_{ij} = \Phi_{ij}^{(long)} + \Phi_{ij}^{(short)} \tag{3.17}$$

Algorithm 3.3.2, shows the major steps involved in the computation of the coupling matrix using long-range and short-range splitting. It can be clearly seen that this implementation is more involved than the uncorrected form discussed in section 3.3.1. For normal clusters, $l_{cut} = 2$ (i.e., monopole+dipole corrections) gives sufficiently accurate results. [5] report that increasing l_{cut} to more than two doesn't improve the accuracy. In addition to forward Fourier transform and backward Fourier transform, the CPU time is also dominated by step 8, for computing q_{lm}, which is a three dimensional integration and step 10 and step 14 for calculating the spherical harmonics. Even-though the product of the spherical harmonics with the ρ_l and Φ_l can be computed and stored before the start of the assembly process, we choose not to do it since we need memory for 14 additional three dimensional arrays.

Algorithm 3.3.2: Long-range and short-range splitting

```
1: Compute and store $\rho_l$ for all $l \leq l_{cut}$.
2: Compute the moments $\int drr^{2+l}\rho_l$ which are required in Eq.(3.13).
3: Solve the one dimensional Eq.(3.14 ) to obtain $\Phi_l$.
4: C ——- Start of Coupling Matrix Assembly Loop. ————
5: for all the transitions $ij$ considered do
6:     $\rho_{ij} = \psi_i \psi_j$.
7:     C ——— Start Poisson Equation Solver ————-
8:     Compute $q_{lm}$ for all $l \leq l_{cut}$ according to Eq.(3.10).
9:     $a_l m$ follows from Eq.(3.13) using stored $\int drr^{2+l}\rho_l$.
10:    Compute $\rho_{ij}^{(long)}$ from Eq.(3.12) and then $\rho_{ij}^{(short)}$ from Eq.(3.16).
11:    Apply Forward Fourier Transform: $\mathcal{F}(\rho_{ij}^{(short)})(\mathbf{k})$.
12:    Multiply by $f^{cut}$ : $\mathcal{F}(\rho_{ij}^{(short)})(\mathbf{k})f^{cut}$.
13:    Apply Inverse Fourier Transform: $\mathcal{F}^{-1}(\mathcal{F}(\rho_{ij}^{(short)})(\mathbf{k})f^{cut})$.
14:    Compute $\Phi_{ij}^{(long)}$ from Eq.(3.15) and then the total potential, $\Phi_{ij}$ is
       obtained as in Eq.(3.17).
```

```
15:     C ——- End of Coupling Matrix Assembly Loop. ————
16:     Assemble the remaining elements of the row corresponding to $ij$.
17: end for
18: End of Coupling Matrix Assembly Loop.
```

Our initial implementation of this method showed that it performed poorly when compared to the Cut-off methods(even when compared to the Cut-off method with D= 1.0) . Therefore this method was not pursued further.

3.3.3 Cut-off based methods

One way to reduce the interactions of the periodic images is to use a super-cell in which the cluster is placed at the center of the super-cell and it is surrounded by vacuum(which is done by padding with zeros in the remaining part of the super-cell). This method is effective and gives accurate results if the cluster is neutral and the super-cell is twice the size of the original cluster. However, since we are dealing with three dimensional grid increasing the length of the cell by twice results in eight times increase in the complexity. Furthermore, for charged clusters, this might induce charge fluctuations on the periodic images, resulting in slow convergence with super-cell size [7].

Cut-off based methods [7, 6, 5] impose a cut-off on the coulomb interaction depending on what image $\mathbf{r}$ and $\mathbf{r}'$ are present. Equation (2.10) is an infinite

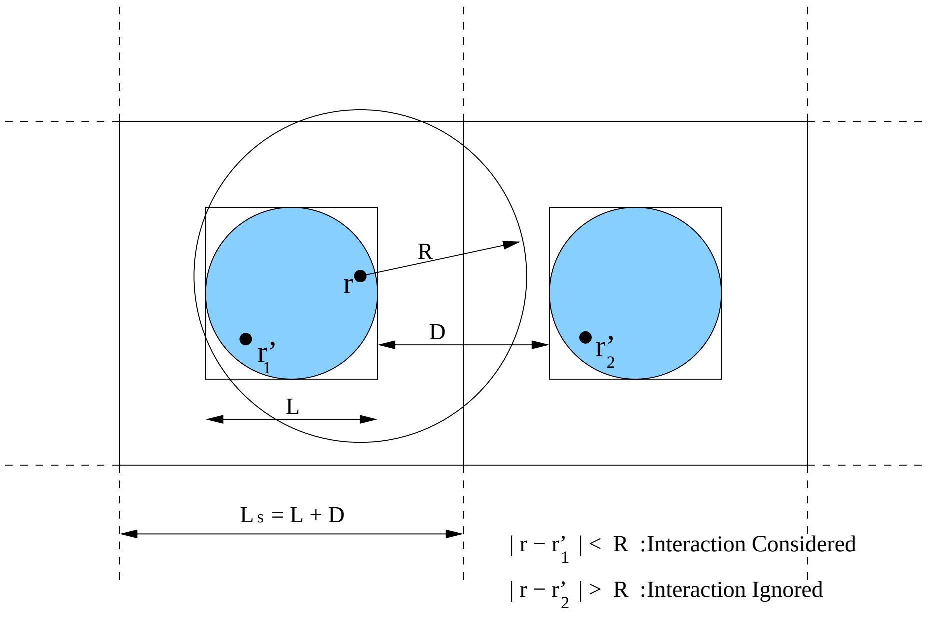

R
r
D
r'_1
r'_2
L
Ls = L + D
| r − r'_1 | < R :Interaction Considered
| r − r'_2 | > R :Interaction Ignored

Figure 3.2: Schematic illustration of the padding procedure used in Cut-off based method for solving the Poisson equation. The original spherical domain of the cluster of diameter L is enclosed in a box of size L and then is padded by a distance D. The interaction of points only within a sphere of radius R are considered.

convolution, but in practice can be evaluated as in equation (3.5). However a Fourier convolution of $\Phi(\mathbf{r})$, Eq. (3.5), includes interactions between points $\mathbf{r}$ and $\mathbf{r}'$ when both are on the same cell *and* when they belong to different cells. In order to avoid the last possibility, cut-off based methods make the following modification, which results in exact solution.

$$\Phi^{cut}(\mathbf{r}) = \int_{cell} \frac{\rho(\mathbf{r'})}{\|\mathbf{r} - \mathbf{r'}\|} d\mathbf{r'}. \tag{3.18}$$

where the integral is performed only over the cell containing $\mathbf{r}$. This effectively defines a modified Coulomb interaction,

$$f^{cut}(\mathbf{r}, \mathbf{r'}) = \begin{cases} \dfrac{1}{|\mathbf{r} - \mathbf{r'}|} & \text{For } \mathbf{r}\text{, } \mathbf{r}\text{' in same cell} \\ 0 & \text{Otherwise} \end{cases} \tag{3.19}$$

The interaction now is no longer a function of $\mathbf{r} - \mathbf{r'}$ only, and therefore does not have the simple multiplicative form for f^{cut} as in the Algorithm 3.3.1.

Onida *et al.* suggest a spherical cut-off method which does preserve the simple multiplicative form. To understand this, consider a cluster which has spherical domain as shown in Figure 3.2. For our implementation we embed this cluster in a box of the same size L, by padding the remaining part of the grid with zeros. We define two parameters D and R. D is the padding parameter for the super-cell such that the size of the super-cell is (L+D). The parameter R corresponds to the interaction cut-off and according to Onida *et* *al.* spherical cut-off method the Coulombic interactions are considered only if $|\mathbf{r} - \mathbf{r}'| < R$. i.e,

$$f^{cut}(\mathbf{r}, \mathbf{r}') = \begin{cases} \dfrac{1}{|\mathbf{r} - \mathbf{r}'|} & \text{For } |\mathbf{r} - \mathbf{r}'| < R \\ 0 & \text{Otherwise} \end{cases} \tag{3.20}$$

If D and R are set equal to L we obtain exact results. In summary, the Fourier transform has to be performed numerically in the super-cell (L+D) and the cut-off in the Coulombic interaction is implemented by setting

$$f^{cut} \rightarrow \frac{1}{|\mathbf{k}|^2} \left[1 - \cos(\mathbf{k}R)\right]$$

Based on this implementation our uncorrected Algorithm 3.3.1 is modified as Algorithm 3.3.3. As we can see in the algorithm the implementation maintains its simplicity as the uncorrected Algorithm with two major differences: First, f^{cut} is modified to take care of the cut-off of coulomb potential and second, the original cell is projected to a super-cell which is twice its size before solving the equations. The performance of this method is poor when compared to uncorrected method but we do get accurate results. The performance however, is still far better than the real-space method and multipole corrections method as will be shown in the numerical experiments chapter.

Algorithm 3.3.3: Poisson Equation Solution with cut-off

```
1: Compute $f^{cut} \longrightarrow \frac{1-\cos(\mathbf{k}R)}{\|\mathbf{k}\|^2}$ for the Super-Cell
2: Start of Coupling Matrix Assembly Loop
3: for all the transitions $ij$ considered do
4:     $\rho_{ij} = \psi_i \psi_j$
5:     C ——— Start Poisson Equation Solver ————-
6:     Build a super-cell twice the size of the Original Cell
7:     Project the Original cell to the center of the Super-Cell
8:     Apply Forward Fourier Transform: $\mathcal{F}(\rho_{ij})(\mathbf{k})$
9:     Multiply by $f^{cut}$ : $\mathcal{F}(\rho_{ij})(\mathbf{k}) f^{cut}$.
10:    Apply Inverse Fourier Transform: $\mathcal{F}^{-1}(\mathcal{F}(\rho_{ij})(\mathbf{k}) f^{cut})$
11:    Collect the values of the potential from the Super-Cell
12:    C ——— End Poisson Equation Solver ————-
13:    Assemble the remaining elements of the row corresponding to $ij$
14: end for
15: End of Coupling Matrix Assembly Loop.
```

Because of the observation made in section 3.3.1 the size of the super-cell in the above algorithm doesn't strictly have to be twice the size of the original cell. We can reduce R and D by compromising little on accuracy and gaining much in speed. Reducing R corresponds to cutting off the interactions between points on the opposite sides of the same cell. D can always be reduced to R. Reducing D further corresponds to letting spurious interactions of the points on the edges of the neighboring cells.

In order to understand the behavior of the error with respect to the size of the super-cell, we studied two systems $Si_{10}H_{16}$ and Si_5H_{12}. Fig. 3.5

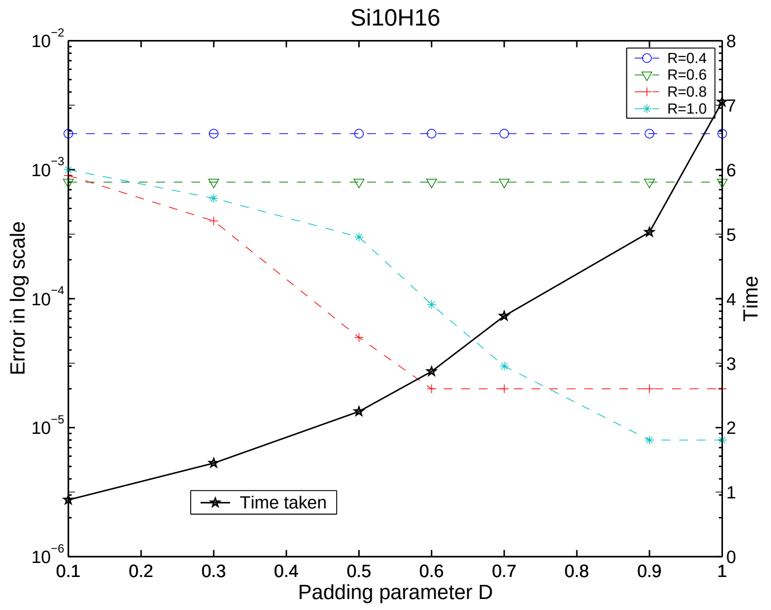


Figure 3.3: Error in the potential due to reducing the Padding parameter(D) and Interaction cut-off parameter(R) for $Si_{10}H_{16}$. D and R are normalized by L.

and 3.6 show the parametric studies of reducing R and D for these two systems. The error in the potential for the Fourier space code (Real-space code potential is taken as reference) is plotted against D for four values of R. The parameters R and D are normalized by L and the time taken is reported for 100 Poisson equation solutions in seconds. The super-cell size for each value of D is L+D. The dashed lines correspond to the error in potential and the solid line corresponds to the time taken for the calculation. As expected the computation time increases rapidly as the size of the supercell increases.

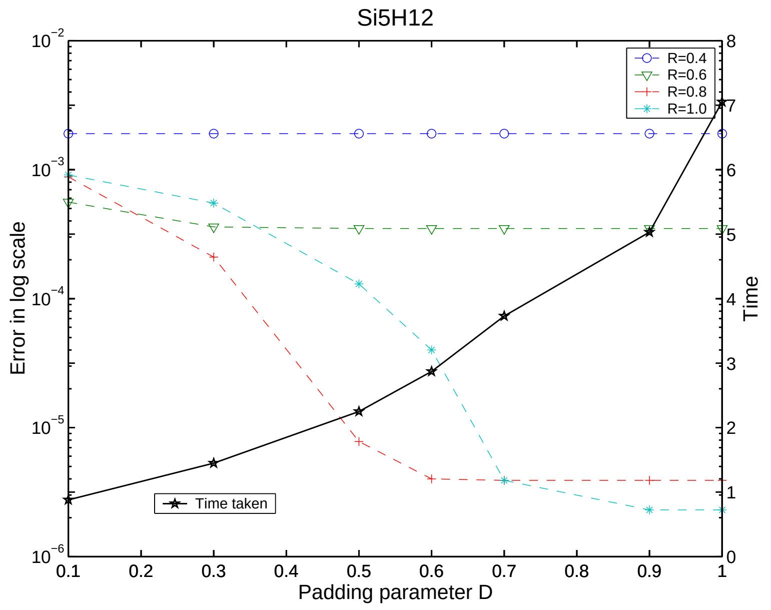


Figure 3.4: Error in the potential due to reducing the Padding parameter(D) and Interaction cut-off parameter(R) for Si_5H_{12}. D and R are normalized by L.

Both clusters are placed on the grid of same size, i.e, (33 x 33 x 33). As observed in both plots for R=0.4 and for R=0.6, the increase in super-cell size doesn't have any effect, this is because the size of the super-cell is double the size of the interaction cut-off parameter, R, for any value of D≥ -0.2 and D$\geq$ 0.2 respectively. Cluster Si_5H_{12} is more localized than $Si_{10}H_{16}$. Again for both cases, for R=0.8, the error drops quickly as the super-cell size increases and this is more so for Si_5H_{12} since it is more localized than $Si_{10}H_{16}$. For R = 1.0, the error doesn't drop as quickly as for R = 0.8 and is only marginally better than R = 0.8 for D$\geq$ 0.75. Clearly the optimal break even point with regards to accuracy and speed is R=0.8 and D = 0.6. However, it is interesting to note that D can be further reduced to 0.4 while maintaining an error of 1e-4. For our implementation, we recommend using parameters R = 0.8 and D = 0.6.

3.3.4 Practical computation of Φ_{ij}

The use of highly tuned Fast Fourier Transform (FFT) routines is key to the performance of the Fourier-space code. The routines related to FFT in the Fourier-space implementation were written as a separate module so that they can be swapped depending on the hardware platform used. The present implementation can run on three different FFT libraries: FFTw, ESSL and MKL 7.0. FFTw is a public domain open source software which. ESSL is proprietary IBM software which works on IBM platforms and MKL 7.0 is Intel math library implemented for Itanium2 based Linux platforms.

When implementing the FFT-based solution of the Poisson equation, several opportunities for optimization will be found. Since the wave functions are real, their Fourier transforms are Hermitian, so f^{cut} can be applied only on half of the domain, which saves arithmetic operations and memory. Usually the FFT software is efficient for input sizes equal to the product of small prime numbers. So it would beneficial to consider the size of the super-cell to be one of these numbers.

The use of vendor supplied FFT routines ESSL and MKL 7.0 result in quicker Fast Fourier Transform. However they accept the input data only in one-dimensional form, so the advantage gained in the speed of Fast Fourier transform is offset while converting the data to one-dimensional form. The overall performance of FFTw is comparable with ESSL and MKL.

3.4 Assembling the coupling-matrix

In this section we discuss the available opportunities for optimized implementation for the computation of each individual element of the coupling matrix. The coupling matrix is given by:

$$K_{ij,kl} = 2\int_{\Omega} \left[\Phi_{ij}(\mathbf{r}) + \Psi_i(\mathbf{r})\bar{\Psi}_j(\mathbf{r})\frac{dV_{\mathrm{XC}}[\rho(\mathbf{r})]}{d\rho(\mathbf{r})}\right]\Psi_k(\mathbf{r})\bar{\Psi}_l(\mathbf{r})d\mathbf{r}. \tag{3.21}$$

In the above equation, $\rho(\mathbf{r})$ is constant throughout the construction of the coupling matrix because it is set by the DFT stage. The analytic form of $V_{xc}[\rho(\mathbf{r})]$, and therefore $dV_{xc}[\rho(\mathbf{r})]/d\rho(\mathbf{r})$, is known. This term is therefore computed once and for all during the setup stage and used subsequently. Our implementation used the Ceperley-Alder exchange-correlation functional [3]. Following [2] we have slightly modified its analytic parameterization to assure a continuous derivative. The $\Phi_{ij}(\mathbf{r})$ term is obtained by the solution of the Poisson equation.

Thus, the assembly of the coupling matrix can be broken down into two distinct steps:

Compute the kl independent kernel of the integral, this needs to be done only once per row:

$$\mathrm{ker}_{ij}(\mathbf{r}) = \Big[\Phi_{ij}(\mathbf{r}) + \rho_{ij}(\mathbf{r})\frac{dV_{xc}[\rho(\mathbf{r})]}{d\rho(\mathbf{r})}\Big]. \tag{3.22}$$

Integrate over the domain to obtain coupling matrix entry corresponding to ij, kl:

$$K_{ij,kl} = \sum_{\mathbf{r}} \Big[\mathrm{ker}_{ij}(\mathbf{r})\Big]\psi_k(\mathbf{r})\bar{\psi}_l(\mathbf{r}). \tag{3.23}$$

The above calculation is repeated for each element of the row corresponding to ij. One could, in principle, use sophisticated integration algorithms to this end. However, we invariably found that for a grid spacing that was small enough to guarantee convergence of the DFT part of the calculation, direct summation was sufficiently accurate. In our implementation we overwrite $K_{ij,kl}$ with $Q_{ij,kl}$ using Eq. (2.4). It is important to note that because of our use of a real-space grid, all wave functions are real and therefore both $K_{ij,kl}$ and $Q_{ij,kl}$ are inherently symmetric, so we *only compute and store their upper triangular half.*

The integration part of Eq. (3.23) is the most time consuming loop in the whole construction of the coupling matrix, since it is executed $kdim^2$ times, where $kdim$ is the size of the coupling matrix. Each integration is of order $\mathcal{O}(ndim)$, where $ndim$ is the number of grid points. Optimized implementation of this part will fetch us more dividends when compared to optimizing any other part of the code. We will first discuss the implementation in the Real-space code and then suggest the improvements for the present implementation.

3.4.1 Real-space implementation

Algorithm 3.4.1 shows the implementation of the assembly loop in Real-space code. There are two important points to note about the implementation: First, for the steps 4, 6 and 9 highly tuned math libraries can be used. However, the Real-space code doesn't make use of the math libraries. Instead do loops are used for those steps. The $-O3$ compiler optimization is very effective in optimizing such simple do loops. Nevertheless, our experiments showed that use of math libraries can speed up the calculation at-least by a factor of two when compared to the do loops. The second point to note is that for the integration kernel in step 9, the product $\text{ker}_{ij}(\mathbf{r})\psi_k(\mathbf{r})$ is independent of l and thus can be pulled outside the l loop on step 8. Even though this might look inconsequential, there is a substantial improvement in performance with this modification since step 9 forms the innermost loop of the whole construction process.

Algorithm 3.4.1: Coupling matrix assembly, Real-space code

```
1: Compute $\frac{dV_{xc}[\rho(\mathbf{r})]}{d\rho(\mathbf{r})}$
2: Start of Coupling Matrix Assembly Loop
3: for all the transitions $ij$ considered do
4:    $\rho_{ij}(\mathbf{r}) = \psi_i(\mathbf{r})\bar{\psi}_j(\mathbf{r})$
5:    Solve for $\Phi_{ij}(\mathbf{r})$ by using PCG with $\rho_{ij}(\mathbf{r})$
6:    Compute $kl$ independent kernel : $\text{ker}_{ij}(\mathbf{r}) = \left[\Phi_{ij}(\mathbf{r}) + \rho_{ij}(\mathbf{r})\frac{dV_{xc}[\rho(\mathbf{r})]}{d\rho(\mathbf{r})}\right]$.
7:    for all $k$ do
8:       for all $l$ do
9:          $K_{ij,kl} = \sum_{\mathbf{r}} \text{ker}_{ij}(\mathbf{r})\psi_k(\mathbf{r})\bar{\psi}_l(\mathbf{r})$.
10:      end for
11:   end for
12: end for
13: End of Coupling Matrix Assembly Loop.
```

3.4.2 Fourier-space implementation

Algorithm 3.4.2 shows the implementation of the assembly loop in Fourier-space code. The modifications suggested above are incorporated and math libraries are used wherever possible. Since there was no BLAS routine for the elemental product of two vectors the routine DVEM from ESSL library was used. This routine is specific only for IBM platforms. In step 8, a temporary vector is used to store the intermediate product. The need for extra memory space was avoided by reusing a vector used in the Poisson equation solver. In the actual implementation we overwrite $K_{ij,kl}$ with $Q_{ij,kl}$ using Eq. (2.4), for simplicity we avoided including it in the above descriptions.

Algorithm 3.4.2: Coupling matrix assembly, Fourier-space code

```
 1: Compute $\frac{dV_{xc}[\rho(\mathbf{r})]}{d\rho(\mathbf{r})}$
 2: Start of Coupling Matrix Assembly Loop
 3: **for** all the transitions $ij$ considered **do**
 4:     $\rho_{ij}(\mathbf{r}) = \psi_i(\mathbf{r})\bar{\psi}_j(\mathbf{r}) \longleftarrow$ Use DVEM
 5:     Solve for $\Phi_{ij}(\mathbf{r})$ by using FFT with $\rho_{ij}(\mathbf{r})$
 6:     $\mathrm{ker}_{ij}(\mathbf{r}) = \left[\Phi_{ij}(\mathbf{r}) + \rho_{ij}(\mathbf{r})\frac{dV_{xc}[\rho(\mathbf{r})]}{d\rho(\mathbf{r})}\right] \longleftarrow$ Use DVEM
 7:     **for** all $k$ **do**
 8:         $\mathrm{tmp}_{ij}(\mathbf{r}) = \mathrm{ker}_{ij}(\mathbf{r})\psi_k(\mathbf{r}) \longleftarrow$ Use DVEM
 9:         **for** all $l$ **do**
10:             $K_{ij,kl} = \sum_{\mathbf{r}} \mathrm{tmp}_{ij}(\mathbf{r})\bar{\psi}_l(\mathbf{r}) \longleftarrow$ Use DDOT
11:         **end for**
12:     **end for**
13: **end for**
14: End of Coupling Matrix Assembly Loop.
```

3.4.3 Using bounded support of the wave functions for optimized integration

The fact that wave-functions are functions with nearly bounded support can be exploited to reduce the cost of the summation in Eq. (3.23). It is only necessary to sum over $\mathrm{Supp}(\Psi_k) \cap \mathrm{Supp}(\Psi_l) \subset \Omega$, where $\mathrm{Supp}(\Psi_k) = \{\mathbf{r} \in \Omega, \Psi_k(\mathbf{r}) \neq 0\}$.

$$K_{ij,kl} = \int_{\mathrm{Supp}(\Psi_k)\,\cap\,\mathrm{Supp}(\Psi_l)} \left(\Psi_i(\mathbf{r})\bar{\Psi}_j(\mathbf{r})\frac{dV_{\mathrm{XC}}(\mathbf{r})}{d\rho(\mathbf{r})} + \Phi_{ij}(\mathbf{r}) \right) \Psi_k(\mathbf{r})\bar{\Psi}_l(\mathbf{r})d\mathbf{r}.$$

A rigorous implementation to use the above scheme requires storing of an additional index array for each of the wave-functions. An easier approach, from an implementation viewpoint, would be to exploit the decaying property of the charge density. It is observed that the charge density is essentially zero in a fairly large part of the physical domain Ω. Since the wave-functions of occupied states vanish wherever $\rho(\mathbf{r}) = 0$, it is only necessary to perform the integration where $\rho(\mathbf{r}) > \varepsilon$, where ε is some small number, i.e.,

$$K_{ij,kl} = \int_{\{\mathbf{r}\ \mid\ \rho(\mathbf{r})>\varepsilon\}} \left(\Psi_i(\mathbf{r})\bar{\Psi}_j(\mathbf{r})\frac{dV_{\mathrm{XC}}(\mathbf{r})}{d\rho(\mathbf{r})} + \Phi_{ij}(\mathbf{r}) \right) \Psi_k(\mathbf{r})\bar{\Psi}_l(\mathbf{r})d\mathbf{r}.$$

It is easy and inexpensive to build a list of all the r-points which satisfy the condition $\rho(\mathbf{r}) > \varepsilon$ in the setup phase. During integration, a smaller summation is then computed using this list.

In order to understand the behavior of the error with respect to ε, we studied its effect on Si_5H_{12} and $Si_{10}H_{16}$ clusters. Fig. 3.6 and Fig. 3.5 show the plot of the error in the coupling matrix elements due to reduction in domain for integration for only points with density greater than ε. The dashed line indicates the error and the solid line indicates the percent of total original spherical domain with values of density $> \varepsilon$. The density cut-off value, $\varepsilon = 1e-6$ produces an error of magnitude $1e-7$ in both the cases. The error produced is an order of magnitude less than the error in the solution of Poisson equation. For our implementation we suggest using density cut-off($dcut$) parameter = 1e-6. Even though the savings in percentage of the original domain for $dcut = 1e-6$ is only 20% in Si_5H_{12} case and 5% in $Si_{10}H_{16}$ case, we expect to see greater savings in percentage of the original domain used for integration for larger systems.

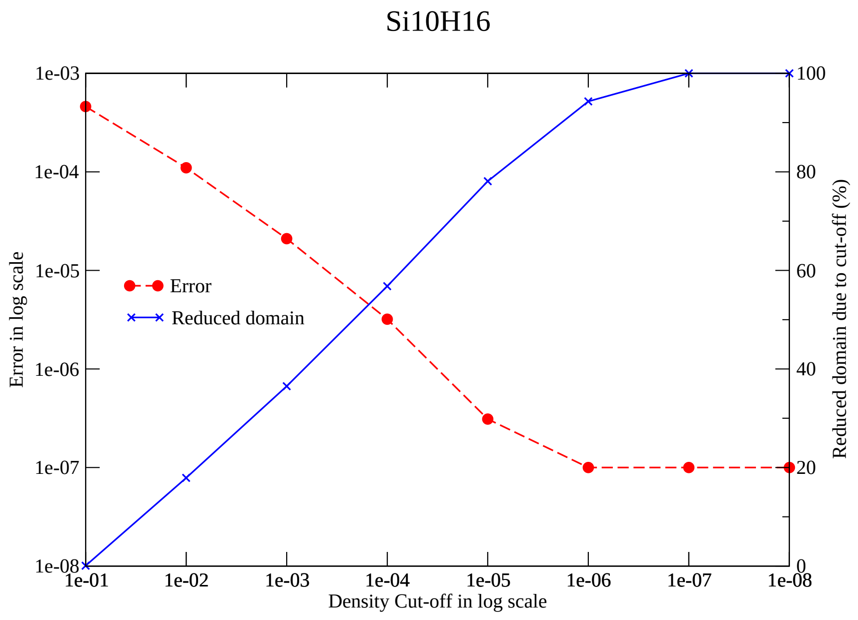


Figure 3.5: Parametric study of density cut-off effect for $Si_{10}H_{16}$

3.5 Parallel implementation and code description

The construction of each row of the coupling matrix is completely independent from the construction of other rows, parallelization by coupling matrix row is natural. Each processor has a copy of all the wave-functions and assembles only the rows which are allocated to it. We initially implemented the static allocation of the work as shown in Fig. 3.7. Since we only compute the upper triangular half of the coupling matrix, the amount of work required for

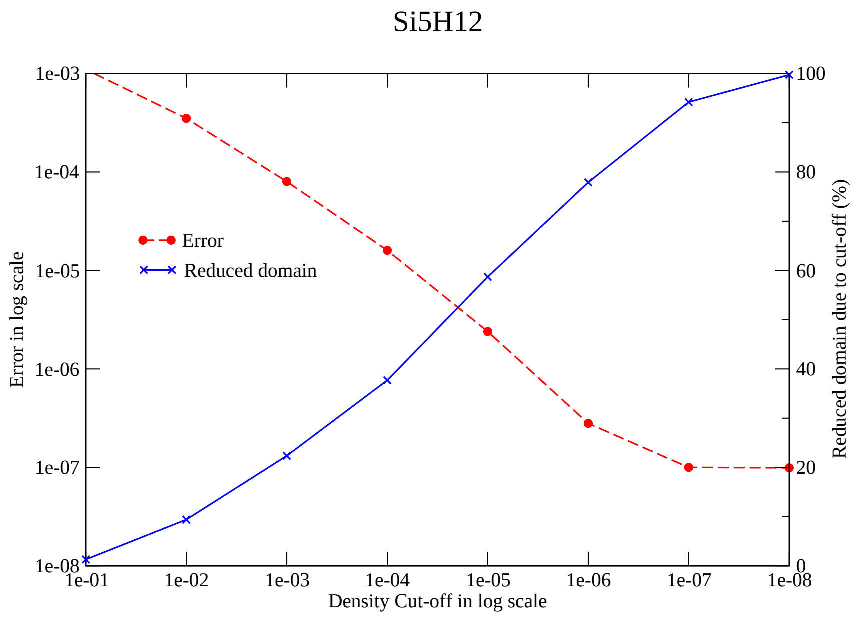
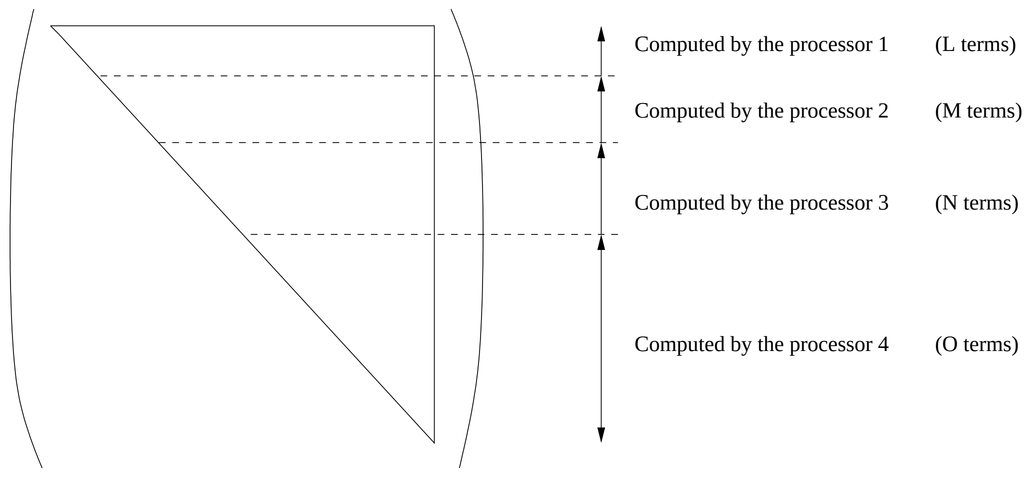


Figure 3.6: Parametric study of density cut-off effect for Si_5H_{12}

each row is not constant. The static allocation of work didn't efficiently use all the processors. Therefore, we reused the “Master-slave” implementation of Real-space code. The details of the model and implementation can be found in [1].

Initially, the code was written in C++ with static load balancing. Later we reused the Fortran77 Real-space code and modified it to match our implementation. The following points list the major modifications to the code:

1. The program was converted from Fortran 77 to Fortran 90.


Computed by the processor 1 (L terms)
Computed by the processor 2 (M terms)
Computed by the processor 3 (N terms)
Computed by the processor 4 (O terms)

$$\boxed{L + M + N + O = kdim(kdim+1)/2 \quad \text{AND} \quad L \sim M \sim N \sim O}$$

Figure 3.7: Static allocation of the work for the construction of the coupling matrix on 4 processors

2. Used allocatable arrays and removed the dependency on the parameter file.

3. Consolidated multiple message passing calls by aggregating messages.

4. Replace the GOTO calls by the do loops.

5. Previously the slave processor used to allocate and deallocate the memory every time the work was allocated to it. In the present implementation the master and the slave processor allocate the memory only

once.

The code is written in Fortran90 and MPI and is tested on `linux`, and the `IBM/SP`. after computing the wave functions the code performs the following tasks:

- Read the wave-functions, the eigenvalues and the occupied states (ϵ_i, ψ_i, $f(\epsilon_i)$).
- Compute the density and the strengths/spectrum that is f_I, using B_β, R and F_I.
- Compute the exchange-correlation potential V_{xc}.
- Compute the coupling-matrix.

Once the coupling matrix $\mathbf{K}$ is constructed the dense eigenvalue problem is solved using ScaLAPACK and oscillator strengths are computed.

Chapter 4

Numerical experiments

In order to validate and compare the methods and the implementation described in this thesis we now present some numerical examples. For our calculations, we selected two semiconductor materials: silicon and gallium arsenide. We compare the optical spectra obtained by the Fourier-space method with the Real-space method. We concentrate on the construction of the coupling matrix (i.e second block in Fig 2.1) and compare it to the results and timings obtained from the Real-space code [1]. The remaining two blocks (i.e Construction of the wave functions and Computation of the oscillator strengths) are already implemented and we use them as a black-box.

The parallel tests have been executed on the `IBM/SP2` at the Minnesota Supercomputer Institute (MSI) (`www.msi.umn.edu`) of the University of Minnesota. The `IBM/SP2` has 82 nodes, with four 375 Ghz power3 processors, sharing 4 GB of memory. -*O3* and -qtune=pwr3 optimization flags are used with the compilation. The FFT routines are implemented using FFTw ([10])library and the ESSL library. For reporting numerical tests for this section we have used FFTw for the implementation. Both FFTw and ESSL libraries produce identical results, however, depending upon the problem size there might be difference in the total elapsed time.

The effectiveness of the TDLDA method in calculating the absorption spectra for the molecules when compared to experiments is explained in detail in [2]. In this chapter, we are concerned only with the performance of the code and we validate the results by comparing it with the Real-space implementation. The first two sections compare the optical spectra obtained by the present implementation with the Real-space code. In the third section we look into the performance of the new implementation with respect to the elapsed time, memory used and scalability. By default, all the tests reported in here use the optimal parameters suggested in the previous chapter, i.e, interaction cut-off parameter, R = 0.8, padding parameter, D = 0.6 and density cut-off parameter, dcut = 1e-6. These parameters are suggested as a trade-off between accuracy and speed. In Section 4.3 we compare the results obtained by the above parameters with the best possible scenarios in terms of speed and accuracy. Finally, in Section 4.4 we discuss the additional gain in speed obtained by just using faster machines.

4.1 Optical Absorption of Galium Arsenide clusters

The $GaAsH_6^{'}$, $Ga_3As_3H_{12}^{'}$ and $Ga_{10}As_{10}H_{30}^{'}$ were recommended as stable clusters and thus we use them here for our comparison with the Real-space implementation. Table. 4.1 describes the various parameters of these test cases.

Fig.4.1 show the calculated TDLDA absorption spectra of $Ga_nAs_mH_l^{'}$ clusters with Real-space implementation and Fourier-space implementation. For comparison, the plots also show the spectra of time-independent Kohn-Sham LDA eigenvalues. The Fourier-space implementation of the TDLDA spectra agrees very well with the Real-space implementation for all the test cases.

	Grid	Grid spacing	Radius	nstate	nocc	kdim
$GaAsH_6^{'}$	$49 \times 49 \times 49$	0.45 a.u	11 a.u	55	7	336
$Ga_3As_3H_{12}^{'}$	$47 \times 47 \times 47$	0.6 a.u	14 a.u	82	18	1152
$Ga_{10}As_{10}H_{30}^{'}$	$61 \times 61 \times 61$	0.6 a.u	18 a.u	151	55	5280

Table 4.1: Description of the test cases for $Ga_nAs_mH_l^{'}$ clusters. *nstate* stands for total number of states considered. *nocc* stands for number of occupied states and *kdim* for the dimension of the coupling matrix.

4.2 Optical Absorption of Hydrogenated Silicon

Hydrogenated silicon clusters are a class of nanostructures obtained by simply cutting a roughly spherical chunk of crystalline silicon, with diameter of up to 20 $\AA$. The effect of surface states on the absorption spectrum can be reduced by passivating the outer atomic layer. Clusters with sufficiently large diameter retain many of the electronic properties of bulk silicon. By reducing its diameter, we can observe the behavior of various electronic properties such as the fundamental electronic gap and absorption spectrum as function of system size. For example, the LDA absorption threshold (defined as the energy difference between the highest occupied molecular orbital and lowest unoccupied molecular orbital obtained in DFT-LDA) is expected to increase smoothly from the bulk limit, around 1 eV, to the range of 5 to 8 eV for small

	Grid	Grid spacing	Radius	nstate	nocc	kdim
SiH_4	$21 \times 21 \times 21$	0.75 a.u	7.5 a.u	15	4	44
Si_5H_{12}	$33 \times 33 \times 33$	0.75 a.u	12 a.u	30	16	224
$Si_{10}H_{16}$	$33 \times 33 \times 33$	0.75 a.u	12 a.u	60	28	896
$Si_{34}H_{36}$	$65 \times 65 \times 65$	0.75 a.u	24 a.u	240	86	13244

Table 4.2: Description of the test cases for Si_nH_m clusters. *nstate* stands for total number of states considered. *nocc* stands for number of occupied states and *kdim* for the dimension of the coupling matrix.

clusters. The optical spectrum is also blue-shifted as the cluster diameter is reduced [2].

We constructed quasi-spherical silicon quantum dots on the computer from "shells" of equi-distant silicon atoms situated around a central atom. The silicon atoms were placed in sites corresponding to the bulk structure of silicon, where each silicon atoms is surrounded by four other silicon atoms possessing tetrahedral symmetry. This results in dangling bonds for silicon atoms on the outer shell, which have "missing" neighbors. Each dangling bond was then eliminated by adding a surface hydrogen atom, thus removing optically active states associated with the surface. Finally, we relaxed the outer layers of the dot by moving the atoms so as to minimize the forces acting on the hydrogen cap atoms.

A selection of optical spectra obtained by computing the TDLDA spectra using Real-space and Fourier-space methods for four hydrogenated quantum dots is shown in Fig.(4.2). It starts with the smallest “quantum dot” one could construct using the above procedure - a $\mathrm{SiH_4}$ molecule based on the central Si atom - and ends with $\mathrm{Si_{34}H_{36}}$. For comparison, Fig.(4.2) also show absorption spectra of the same dots computed using conventional (time-independent) LDA, where the spectrum is based on LDA eigenvalue differences weighted by the matrix element between the associated eigenvectors. It is readily observed in the plots that our calculations agree very well with the Real-space code for all the hydrogenated quantum dots considered.

4.3 Performance

4.3.1 Timings

Table 4.3 shows the wall clock runtime for the coupling matrix generation for $\mathrm{Si}_n\mathrm{H}_m$ molecules. It can be clearly absorbed that there is substantial gain in speed with the Fourier-space implementation. Furthermore, as the problem size gets bigger the speed-up factor also increases. For the $Si_{34}H_{36}$ there is an order of magnitude improvement in speed with no visible loss in accuracy(as seen in the plots of the optical spectra). This improvement not only helps us

	SiH_4	Si_5H_{12}	$Si_{10}H_{16}$	$Si_{34}H_{36}$
Number of Processors	1	1	1	8
Real-space Code	6 sec	121 sec	9:35 min	15:30 hrs
Fourier-space Code	1 sec	15 sec	1:29 min	1:35 hrs
Speed-up	6.0 times	8.0 times	6.5 times	9.8 times

Table 4.3: Comparison of wall-clock runtime of the parallel TDLDA code (not including the generation of Kohn-Sham eigenvalues and wave-functions) using Fourier space and Real Space for Si_nH_m clusters

consider bigger problems but also reduces the time for investigating medium sized problems.

In order to investigate the relative effect of parameters we have suggested to use for the calculation on the total elapsed time, we studied the $Si_{34}H_{36}$ case with the best possible scenario in terms of speed and accuracy. Fig.4.3 shows the optical spectra obtained with three configurations: case A, the optical spectra obtained with the best possible case in terms of speed, case B, the optical spectra obtained with the suggested parameters and case C, the optical spectra obtained with no approximations. In the case A, the spectra obtained is visibly off from the reference spectrum but the whole calculation is completed within 50 minutes. In case C, the spectra cannot get any better than that in the case of case B. Furthermore, the time taken is also increased to 5 hours(which is still better than Real-space implementation by a factor

	SiH_4	Si_5H_{12}	$Si_{10}H_{16}$	$Si_{34}H_{36}$
Real-space Code	17 MB	33 MB	38 MB	420 MB
Fourier-space Code	11 MB	21 MB	29 MB	375 MB
Percent reduction	54%	57%	31%	12%

Table 4.4: Comparison of memory used per processor for the Real space code and the Fourier space code for Si_nH_m clusters

of three). Thus it can be seen that the suggested parameters are optimal in terms of speed with no visible loss in accuracy.

4.3.2 Memory

Table 4.4 shows the memory used for both the implementations. In case of Fourier-space code the listed values correspond to the case of the use of suggested parameters. The memory consumed by the Fourier-space code is clearly less than Real-space code. The percent difference however, reduces as the problems size gets larger. In the present implementation of the Fourier-space code, the wave-functions are stored twice: once with original values and the second time with the reduced domain with respect to the density cut-off. This is done because storing only the index values of the original domain for the density cut-off and using the index array while integrating didn't allow us to make use of the critical "ddot" math library routine. Fig 4.3 also shows the memory used for the the calculation of the $Si_{34}H_{36}$ spectra. Even though we see that the memory consumed for case B is less than the Real-space code, for case C, the memory consumed is 35% more than the Real-space code. It is possible to reshuffle the wave-functions in place instead of using twice the storage. We plan to do it in our future work.

4.3.3 Scalability

The Real-space code had near-linear scalability. Fig.4.4 shows the scalability of the Fourier-space code for $Si_{34}H_{36}$ cluster. Even though the algorithm is embarrassingly parallel, we see in the plot that as the number of processors increases there is a deviation from the ideal curve. The allocation of the work by the master node might become a bottleneck as the number of processors increases. However, for this case we feel the deviation from the ideal curve is because of the size of the problem. For 16 processors the total elapsed time is 51 minutes and the setup time starts to dominate. We expect for larger cases the code should show the same scalability as the Real-space implementation.

	IBM p655	SGI Altix
Processor	Power4	Intel Itanium2
Processor Speed	1.5GHz	1.5GHz
Memory requested	500 MB/Processor	500 MB/Processor
Compiler	xlf_r	ifort 8.0
Compiler options	-O3 -qtune="pwr4"	-O3 -ftz
FFT Library	FFTw	FFTw
BLAS Library	ESSL	SCSL

Table 4.5: Configuration of IBM p655 and SGI Altix

	Setup Time	Poisson solver time	Total CPU time	Wall-Clock Time
IBM Regatta	70 secs	2151 secs	19093 secs	42:00 mins
SGI Altix	28 secs	3447 secs	15660 secs	35:24 mins

Table 4.6: Comparison of time taken by IBM Power4 and SGI Altix executing the Fourier-space code for $Si_{34}H_{36}$ clusters on 8 processors. The Poisson solver time and the Total CPU time are the sum of the time taken by each of the 8 processors.

4.4 Comparison of hardware performance.

For the most part, the computational nodes used by Electronic materials structure group at the University of Minnesota are the IBM SP and the IBM Regatta, available at the Minnesota Supercomputing Institute (MSI). A recent addition to MSI in the last year is SGI Altix, which is a Linux cluster with 64-bit Itanium2 processor. The Itanium2 processor with 128 floating-point registers and large L3 cache and other resources for floating-point intensive codes looks ideal for the applications in computational material science. In this section we intend to compare the performance of the Itanium2 system with the IBM Power4 system. Since the Fourier-space code is similar to the other applications in the computational materials science, we use the Fourier-space code to benchmark the two machines.

The two main parts of the code are the FFT routine and *ddot* BLAS function. We used FFTw for benchmarking. For *ddot* function we used ESSL library on IBM Power4 system and SCSL for SGI Altix system. Table 4.5. lists other configurations and the options used for the respective machines. For the benchmark, we choose to execute the coupling matrix construction of $Si_{34}H_{36}$ cluster on 8 Processors.

Table 4.6 presents the results in execution time for the Power4 and Itanium2 systems. The lowest elapsed time is obtained on the Itanium2 system. The Poisson solver time and the Total CPU time denote the time taken by all the 8 processors. For the Poisson solution part Power4 system is faster than Itanium2. However, for the *ddot* part Itanium2 is much faster than the Power4 system. Overall there was a reduction of factor of 2 by using these machines when compared to the IBM SP system. The Itanium2 system shows good performance relative to Power4 for the Fourier-space code and thus can be used for other applications in computational material science.

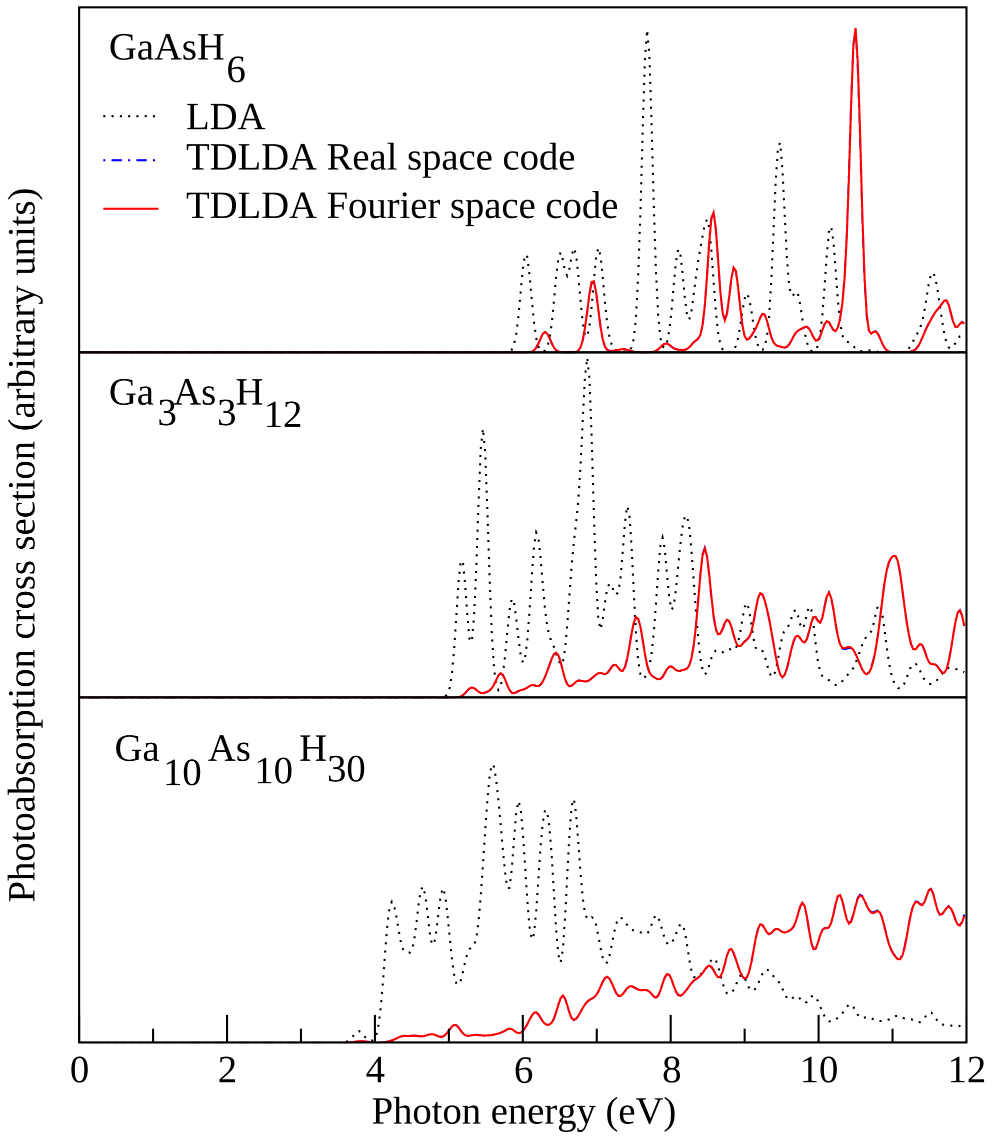


Figure 4.1: Calculated TDLDA spectra by Real space code(dashed lines) and Fourier space code(solid lines) of $Ga_nAs_mH_l'$ clusters. Spectra of time-independent Kohn-Sham LDA eigenvalues (dotted lines) are shown for reference.

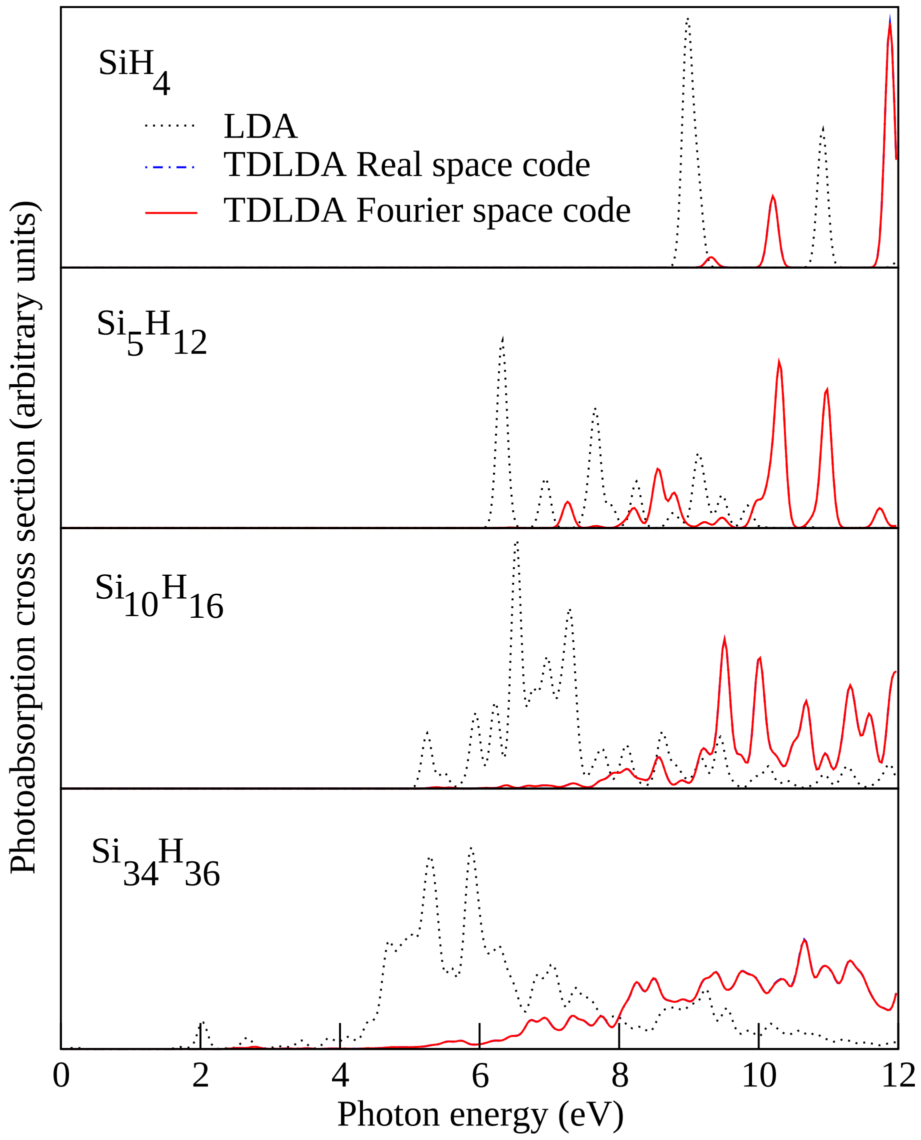


Figure 4.2: Calculated TDLDA spectra by Real space code(dashed lines) and Fourier space code(solid lines) of Si_nH_m clusters. Spectra of time-independent Kohn-Sham LDA eigenvalues (dotted lines) are shown for reference.

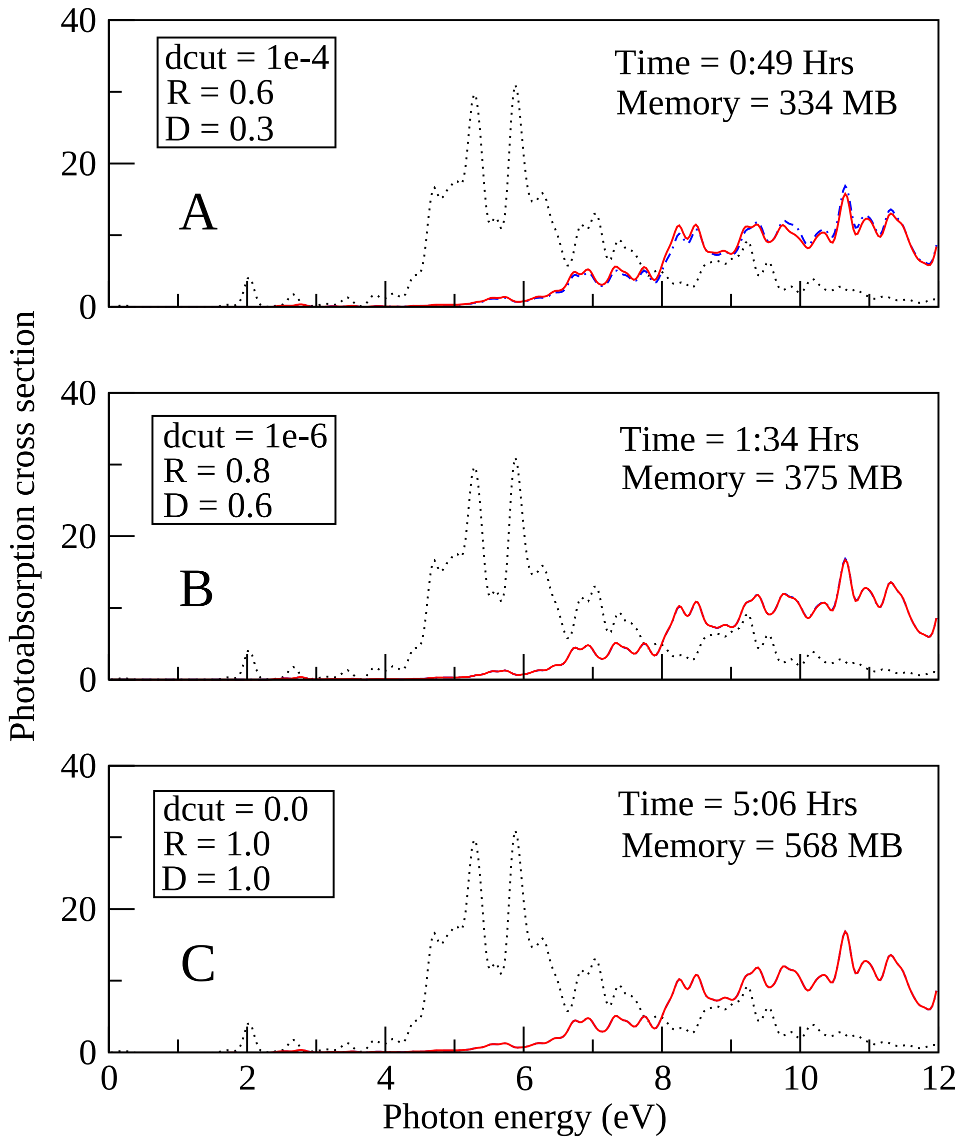


Figure 4.3: Calculated TDLDA spectra by Real space code(dashed lines) and Fourier space code(solid lines) of $Si_{34}H_{36}$ cluster.

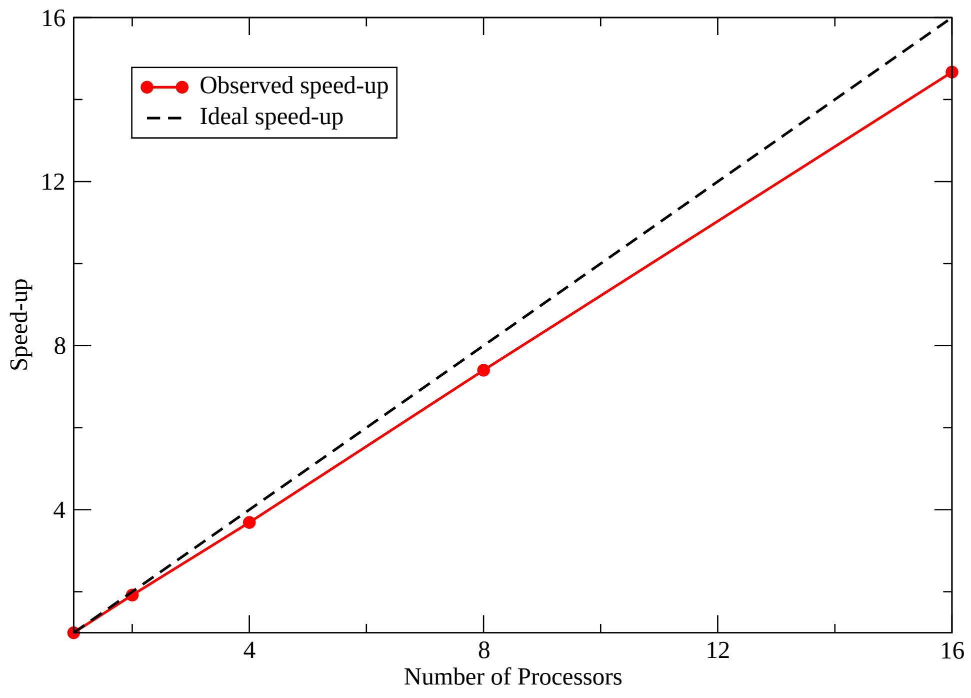


Figure 4.4: Scalability of coupling matrix construction for $Si_{34}H_{36}$ cluster.

Chapter 5

Conclusion

5.1 Conclusion

In conclusion, we have implemented a highly optimized version of the coupling matrix construction, which arises in time-dependent density functional theory. The implementation is based on using a Fourier space method leading to fast solutions of the Poisson equations. A detailed description of the Poisson equation solution and the integration were given. The main advantage of this approach versus the use of Preconditioned Conjugate Gradient method used previously [1] for handling the Poisson problems, is that it leads naturally to fast-solutions of the Poisson equations. The issue of the influence of near-by cells when solving Poisson's equation by FFT for non-periodic systems, can be handled in a number of ways, including the cut-off approach described in [7, 6, 5]. Though this increases the cost slightly, this increase is rather limited for sufficiently localized systems. A systematic parametric study was done in order to evaluate optimal parameters. Numerical results show that the use of these parameters don't alter the visible accuracy of the clusters.

Among other strategies which lead to a significant reduction in the overall computational time is the use of Math Libraries for the integration part. Local support for efficient integration was also made use to reduce the computation time further. The numerical results indicate that the performance of the method is far superior than the real-space code. In summary, the following points can be listed as the main contributions of the thesis

1. Implemented FFT based solution of the Poisson equation
2. Reorganized the integration scheme to use highly-tuned Math Libraries
3. Used local support of wave-functions for efficient integration
4. Conducted parametric studies of the approximations involved and suggested optimal parameters.

5. Conducted numerical tests on $Ga_mAs_nH_l'$ and Si_mH_n clusters to validate the results.

6. Reduced the total CPU time by an order of magnitude when compared to the Real-space implementation

7. Compared the performance of the Itanium2 with the IBM Power4 system

5.2 Future work

There are few other modifications which can be considered to further improve the performance. The integration for each of the coupling matrix elements is the most time consuming part of the code in the Fourier-space implementation. Sophisticated integration techniques, other than plain summation can be considered. In the present version, each element of the coupling matrix is assembled one at a time using level-1 BLAS routines. Level-2 BLAS routines can be considered to assemble a block of elements at a time and thus increase the cache utilization.

With the performance improvement in the calculation of the coupling matrix the bottleneck has now shifted from the construction of the coupling matrix to the eigenvalue and eigenvector calculation, i.e, the last block in the Fig. 2.1. For the $Si_{34}H_{36}$ case, coupling matrix construction takes 1:35 hours on 8 processors when compared to 5 hrs for the eigenvalue and eigenvector calculation using SCALAPACK on identical number of processors.

Bibliography

[1] W. R. Burdick, Y. Saad, L. Kronik, I. Vasiliev, M. Jain, and J. R. Chelikowsky, “Parallel Implementation of Time-Dependent Density Functional Theory”, CPC, **156**, 22-42 (2003).

[2] I. Vasiliev, S. Öğüt, and J. R. Chelikowsky, ”First Principles Density Functional Calculations for Optical Spectra of Clusters and Nanocrystals”,Phys. Rev. B **65**, 115416 (2002).

[3] D. M. Ceperley and B. J. Alder, Phys. Rev. Lett. **45**, 566 (1980).

[4] I. Vasiliev, Ph.D thesis, University of Minnesota (2001).

[5] A. Castro , A. Rubio, and M.J. Stott, “ Solution of Poisson’s equation for finite systems using plane-wave methods”, Canadian J. of Phys. **81**, 1151 (2003).

[6] M.R. Jarvis, I.D. White, R.W. Godby and M.C. Payne, "Supercell technique for total-energy calculations of finite charged and polar systems", Phys. Rev. B **56**, 14972 (1997)

[7] G. Onida, L. Reining, R.W. Godby, R. Del Sole, and W. Andreoni, "Ab-initio calculations of the quasiparticle and absorption spectra of clusters: the sodium tetramer", Phys. Rev. Lett. **75**, 818 (1995)

[8] G. Lauritsch, P.-G. Reinhard "An FFT solver for the Coulomb problem", Int. J. Mod. Phys. C **5** (1994) 65

[9] L. D. Landau and E. M. Lifshitz, the classical theory of fields, 4th edition (Pergamon, Oxford, 1975), pp. 97-99.

[10] Frigo, Matteo and Johnson, Steven G., "FFTW: An adaptive software architecture for the FFT, Proc. 1998 IEEE Intl. Conf. Acoustics Speech and Signal Processing", pp. 1381–1384, 1998, **3**, IEEE

[11] W. Hackbusch, *Multi-Grid Methods and Applications* (Springer, New York, 1986)

[12] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen.

LAPACK Users Guide, 3rd Edition (Society for Industrial and Applied Mathematics, Philadelphia, 1999).

[13] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users Guide* (Society for Industrial and Applied Mathematics, Philadelphia, 1997).

[14] MPI Standard, http://www-unix.mcs.anl.gov/mpi/

[15] I. Foster, *Designing and Building Parallel Programs*, (Addison-Wesley, Reading, 1995).

[16] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing*, (Benjamin/Cummings, Redwood City, 1994)

[17] J. W. Demmel, *Applied Numerical Linear Algebra*, (Society for Industrial and Applied Mathematics, Philadelphia, 1997)

[18] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd edition (http://www-users.cs.umn.edu/ saad/books.html, 2000), pp. 244-250.

[19] Y. Saad, *Numerical Methods for Large Eigenvalue Problems: Theory and Algorithms*, (John Wiley, New York, 1992)

[20] L. D. Landau and E. M. Lifshitz, *Quantum Mechanics*, 3rd edition (Pergamon Press, Oxford, 1977).

[21] See, e. g., J. R. Chelikowsky and S. G. Louie (Eds.), *Quantum Theory of Real Materials* (Kluwer, Boston, 1996); K. Ohno, E. Esfarjani, and Y Kawazoe, *Computational Materials Science* (Springer-Verlag, Berlin, 1999).

[22] I. Vasiliev, S. Öğüt, and J. R. Chelikowsky, Phys. Rev. Lett **86**, 1813 (2001).

[23] See, e. g., M. L. Cohen and J. R. Chelikowsky, *Electronic Structure and Optical Properties of Semiconductors*, 2nd ed. (Springer-Verlag, Berlin 1989); J. R. Chelikowsky and M. L. Cohen, in *Handbook on Semiconductors*, T. S. Moss, Ed. , 2nd Edition (Elsevier, Amsterdam, 1992); W. E. Pickett, Comput. Phys. Reports **9**, 115 (1989); M. C. Payne, M. P. Teter, D. C. Allan, T. A. Arias, and J. D. Joannopulos, Rev. Mod. Phys. **64**, 1045 (1992).

[24] E. K. U. Gross, J. F. Dobson, and M. Petersilka, in *Density Functional Theory*, edited by R. F. Nalewajski, (Springer-Verlag, Berlin, 1996), p.

81; M. Petersilka, U. J. Gossmann, and E. K. U. Gross, Phys. Rev. Lett. **76**, 1212 (1996).

[25] M. E. Casida, in *Recent Advances in Density-Functional Methods*, Part I, edited by D. P. Chong (World Scientific, Singapore, 1995), p. 155; in *Recent Developments and Applications of Modern Density Functional Theory*, edited by J. M. Seminario (Elsevier, Amsterdam, 1996), p. 391.

[26] J. R. Chelikowsky, N. Troullier, and Y. Saad, Phys. Rev. Lett **72**, 1240 (1994); J. R. Chelikowsky, N. Troullier, K. Wu, and Y. Saad, Phys. Rev. **B50**, 11355 (1994).

[27] L. Kronik, I. Vasiliev, M. Jain, and J. R. Chelikowsky, J. Chem. Phys. **115**, 4322 (2001).

[28] Y. Saad, A. Stathoupolos, J. R. Chelikowsky, K. Wu, and S. Öğüt, BIT **36**, 563 (1996).

[29] A. Stathopoulos, S. Öğüt, Y. Saad, J. R. Chelikowsky, and H. Kim, Comput. Sci. Eng., **2**, 19 (2000).

[30] G. D. Smith, *Numerical Solutions of Partial Differential Equation: Finite Difference Methods*, 2nd ed. (Oxford, New York, 1978).

[31] POWER4 System Microarchitecture,

http://www-1.ibm.com/servers/eserver/pseries/hardware/whitepapers/power4.html

[32] Intel Itanium2 processor,

http://www.intel.com/products/server/processors/server/itanium2/