İSTANBUL TECHNICAL UNIVERSITY ★INFORMATICS INSTITUTE

A PARALLEL IMPLEMENTATION OF REAL-SPACE GREEN'S FUNCTION METHOD FOR CALCULATION OF VIBRATIONAL DENSITY OF STATES

Ms. of S. Thesis

Berk ONAT

Department : Informatics
Programme : Computational Science and Engineering

Supervisor: Assoc. Prof. Dr. Sondan DURUKANOĞLU FEYİZ

February 2006

İSTANBUL TECHNICAL UNIVERSITY ★ INFORMATICS INSTITUTE

A PARALLEL IMPLEMENTATION OF REAL-SPACE GREEN'S FUNCTION METHOD FOR CALCULATION OF VIBRATIONAL DENSITY OF STATES

Ms. of S. Thesis

Berk ONAT

Department: Informatics
Programme: Computational Science and Engineering

February 2006

İSTANBUL TECHNICAL UNIVERSITY ★ INFORMATICS INSTITUTE

A PARALLEL IMPLEMENTATION OF REAL-SPACE GREEN'S FUNCTION METHOD FOR CALCULATION OF VIBRATIONAL DENSITY OF STATES

Ms. of S. Thesis

Berk ONAT

702031016

Date of Submission : **17 February 2006**
Date of Defence Examination : **30 January 2006**

Supervisor (Chairman) : **Assoc. Prof. Dr. Sondan DURUKANOĞLU FEYİZ**
Examining Committee : **Prof. Dr. Serdar ÇELEBİ**
Assoc. Prof. Dr. Nazmi POSTACIOĞLU

February 2006

ACKNOWLEDGEMENTS

I would like to thank Dr. Sondan Durukanoğlu Feyiz for her guidance and advice in every single step of my research and studies. It is her endeavor, care and energy which makes my studies a complete Thesis. I also thank to Dr. Hasan Dağ for his invaluable advises. I am also indebted to my friends for their patience and advises. I will always appreciate their great company and great help. Finally, I am also grateful to my family for their endless support and care. In every state of demand they are all ready for assistance.

February 2006 Berk ONAT

TABLE OF CONTENTS

LIST OF FIGURES

Figureler

ÖZET

Bu tez çalışmasında yerel titreşim durum yoğunlukları hesaplamalarında kullanılan, gerçek uzayda Green fonksiyonları (RSGF) metodu için MPI tabanlı paralel hesaplama yöntemi geliştirilmiştir, böylece kusur ve kirlilik barındıran gerçekçi sistemler için daha çok araştırma yapılabilecektir. Paralel hesaplamada mükemmel paralellik (embarrassingly parallel) sağlayacak olan, farklı frekanslar için gerekli hesaplamaların işlemcilere dağılımı yöntemi kullanılmıştır. Parallel RSGF yöntemini test etmek için Cu kristalinin (100) düz yüzeyi prototip sistem olarak seçilmiştir. Örnek katıyı oluşturan atomların etkileşimlerini tanımlayan Hamiltonyen, Gömülü Atom Yöntemi (EAM) ile elde edilmiştir. Yapılan bir çok küçük ve büyük etkileşim matrisi barındıran testler sonucunda, paralel uygulamanın, hesaplamaları dikkate değer şekilde hızlandırdığı görülmüştür. Tezde, geliştirilen paralel hesaplama yönteminin detayları aktarılmış ve test sonuçları irdelenmiştir.

ABSTRACT

In this Thesis, an MPI based parallel algorithm is developed to implement the real space Green's function method for calculating the vibrational density of states corresponding to a solid so that one can conduct more studies for realistic systems involving defects and impurities. The parallel implementation is carried out through embarrassingly parallel algorithm by simply distributing the frequencies to processors for a range of spectrum. The (100) flat surface of Cu crystal is chosen as the prototype system to test the parallel version of RSGF technique. The Hamiltonian describing the interactions between the atoms of Cu(100) within the system is obtained from the embedded atom method. From the investigation of various test runs, involving smaller and larger interaction matrices, it is found that the parallel implementation speeds up the calculations, on the average, by an order of magnitude. The parallel implementation details and results are presented in the Thesis.

CHAPTER 1

INTRODUCTION

Investigating vibrational or electronic states of a solid involve calculations of the eigenvalue problem, $Hu = Eu$, or equivalently the Green's functions corresponding to the Hamiltonian, H. In the case of vibrational density of states of a solid, one may directly diagonalize the dynamical matrix portraying the interactions between the atoms in N layers of slab instead of finding the Green's function corresponding to the Hamiltonian. This approach is called the slab method and is the most commonly used technique to obtain the frequency spectrum for a solid with or without surface in k-space. In many cases the calculations are easily be carried out for systems with high symmetries such as bulk systems with no defects. On the other hand, for systems with reduced symmetries which require large computational cells, slab method calculations become costly. If ones interest lies in calculating vibrational spectrum for a local region of interest in real space, then one has to obtain the Green's function corresponding to the Hamiltonian. However, for large systems, systems with atoms of the order of thousands, obtaining the Green's function is a formidable task, as the interaction matrix H is order of $3Nx3N$, where N is the number of atoms in the system and the Green's function associated with it is defined as $G = (zI - H)^{-1}$. The continued fraction method proposed by Haydock [1] is the pioneering recursive method to circumvent such problems for complex systems requiring big interaction matrix H. Although the continued fraction method is a local approach in real space, it is not an efficient technique to follow, as the Green's function corresponding to predefined locality is written in a continued fraction form and one has to enforce a truncation for the level of coefficients of the continued fraction which in turn leads to the inaccuracies in the calculations.

Another local approach in real space is the real space Green's function (RSGF) method.[2] In this method one can focus on any 'local' region according to his/her need and analyze the effect of the rest of the system on that particular region. Also there is no need for the system to be periodic and it is, thus, particularly suitable for studying local vibrational density of states

in complex systems with defects, disorder, and reduced symmetry. The only prerequisite is that the interatomic potential between the atoms in the system be of finite range, as it is then possible to write the force constant matrix in a block-tridiagonal form in which the sub-matrices along the diagonal represent interactions between atoms within a chosen local region and the sub-matrices in the 'off-diagonal' corresponds to interactions between neighboring localities. Thus, the method allows one to work with much smaller matrices, whose dimensions are defined by the subsystems, rather than the interaction matrices for entire system. Since an infinite/semi-infinite system is converted quite naturally into an infinite/semi-infinite set of local regions, there is no a *priori* truncation on the system size, as would be the case for matrix diagonalization methods based on k-space. The RSGF method also has an advantage over the continued fraction method as it does not involve truncation schemes to determine the recursion coefficients, rather a more general and simpler recursive scheme is applied.

However, for systems with lack of spacial periodicity, such as quasi-crystals [3], systems with defects [4], the calculations are hampered by the need of large sub-matrices to describe the complexity of the structure. This, in turn, leads to costly calculations. One way to reduce the time wise cost might be to develop a parallel program for the implementation of RSGF method. It is therefore the aim of this Thesis to improve a parallel implementation of RSGF technique so that one can computationally mimic more realistic systems with defects and reduced symmetry rather than studying more idealistic systems.

The rest of the Thesis is organized as follows: Chapter 2 gives a brief summary for crystal structures. Chapter 3 presents the technical specifications of RSGF method and EAM potentials. The parallel computation methods are narrated in chapter 4. Followed by the simulation model and the parallel implementation details for the application in chapter 5. The results and discussions are summarized in chapter 6. The conclusions are presented in chapter 7.

CHAPTER 2

CRYSTAL STRUCTURE

2.1 Classification of crystal systems

The physical structure of a solid is essentially described by the way that the atoms, ions, or molecules are arranged within the solid. If atoms of a solid are arranged in a periodically repeating cell in three dimensions on a lattice, they form a crystalline solid. The repeating cell which tiles with suitable crystal translational operations and spans the crystal is said to be the unit cell of the crystal. According to the structure of unit cells, the crystals are classified into 7 systems based on their lattice constants and interaxial angles.[5],[6] If the unit cell is a parallelopiped with sides a, b, and c and if α, β, and γ are, respectively, the angles between b and c, a and c, and a and b, as shown in Fig. 2.1, the systems may be classified as in Table 2.1.

Most element metals crystallize into three closely packed crystal structures, body-centered cubic (bcc) (Fig. 2.2b), face-centered cubic (fcc) (Fig. 2.2c), and hexagonal close pack (hcp) (Fig. 2.2d). Here we only consider Cu which is crystallized into fcc crystal structure. In an fcc crystal structure, atoms or ions of a solid are placed on each corner, as well as on the center of each face of the unit cubic cell as shown in Fig. 2.2c.

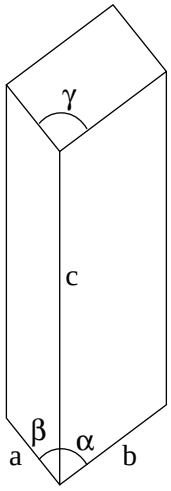

Figure 2.1: A Parallelopiped with sides a, b, c.

Table 2.1: Classification of crystal systems

Crystal system	Unit cell lengths and interaxial angles	Lattice unit cell
Cubic	$a = b = c, \alpha = \beta = \gamma = 90^0$	Simple cubic Body-centered cubic Face-centered cubic
Tetragonal	$a = b \neq c, \alpha = \beta = \gamma = 90^0$	Simple tetragonal Body-centered tetragonal Simple orthorhombic Body-centered orthorhombic
Orthorhombic	$a \neq b \neq c, \alpha = \beta = \gamma = 90^0$	Base-centered orthorhombic Face-centered orthorhombic
Trigonal	$a = b = c, \alpha = \beta = \gamma \neq 90^0$	Simple trigonal
Hexagonal	$a = b \neq c, \alpha = \beta = 90^0, \gamma = 120^0$	Simple hexagonal
Monoclinic	$a \neq b \neq c, \alpha = \gamma = 90^0 \neq \beta$	Simple monoclinic Base-centered monoclinic
Triclinic	$a \neq b \neq c, \alpha \neq \beta \neq \gamma \neq 90^0$	Simple triclinic

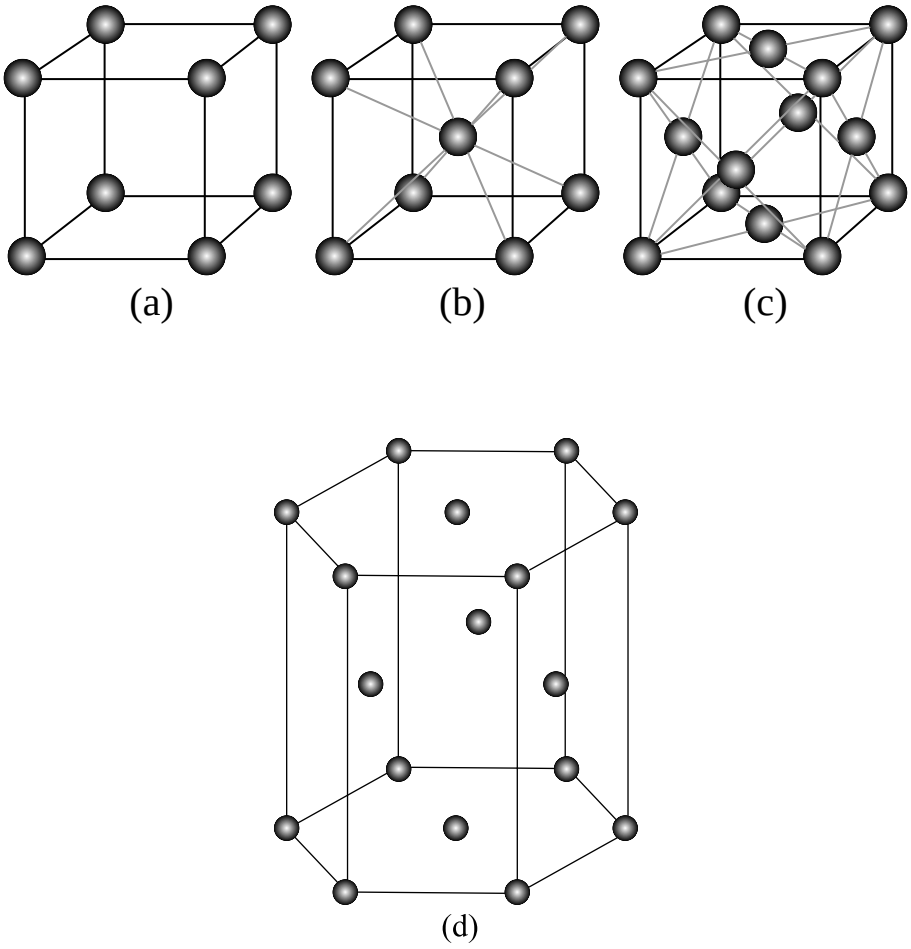

Figure 2.2: Crystal structures: (a) simple cubic, (b) body-centered cubic, (c) face-centered cubic, (d) hexagonal close-packed.

2.2 Indices for crystal planes in a cubic unit cell

Since some crystallized solids have orientation dependent properties, it is necessary to specify the orientation of planes in crystal lattices. The orientation of a lattice plane is usually identified by its Miller indices in parenthesis. The Miller indices of a crystal plane are defined as the reciprocals of the intercepts (with fractions cleared) which the plane makes with the axis of the cubic unit cell. Thus, a cubic crystal plane with intercepts $(1, \frac{1}{4}, 0)$ is called (410) plane. Since fractional intercepts are not allowed, the fractional intercepts are multiplied by 4 to clear the $\frac{1}{4}$ fraction. Some cubic lattice planes with corresponding Miller indices are presented in Fig. 2.4. There is also a notation to identify a family of lattice planes. Crystallographicaly equivalent planes form a family of lattice planes. For example, the (100),(010), and (001) are equivalent in a cubic crystal under the symmetry of the crystal and are collectively specified as the {100} planes.

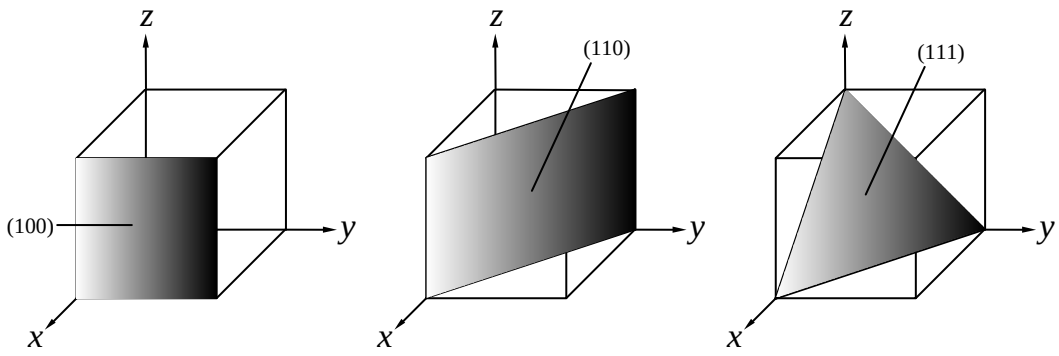

Figure 2.3: Miller indices of some lattice planes in a cubic crystal.

CHAPTER 3

TECHNICAL SPECIFICATIONS OF REAL-SPACE GREEN'S FUNCTION METHOD

3.1 Interaction Potentials

Atoms in model systems are allowed to interact via embedded atom potential (EAM) developed by Foiles, Baskes, and Daw. This is of many-body and has been proven to be reliable for many of the characteristics of the bulk and surface systems for the six fcc metals Ag, Au, Cu, Ni, Pd, and Pt, and their alloys. [7]

The idea of EAM is that the energy of each atom is given by the energy needed to embed the atom in the local-electron density as provided by the other atoms of the metal. The electron density in the vicinity of each atom can be expressed as a sum of the density contributed by the i^{th} atom plus the electron density from all the surrounding atoms. Thus, the energy of atomic site i is given by

$$E_i = F_i(\rho_h, i) + \frac{1}{2}\sum_j \phi_{ij}(R_{ij}) \tag{3.1}$$

where $\rho_{h,i}$ is the host electron density at atomic site i and defined as a superposition of ρ_j^a, the electron density contributed by the atom j to the lattice site i as a function of distance from its center

$$\rho_{h,i} = \sum_{j(\neq i)} \rho_j^a(R_{ij}). \tag{3.2}$$

In this expression, R_{ij} is the distance between the atoms and $\rho_{ij}(R_{ij})$ is the core-core pair repulsion of i and j atoms. F_i in Eqn. 3.1 represents the energy to embed atom i in electron density $\rho_{h,j}$. Then the total energy of a solid is simply the sum over all atom's energies.

$$E_{tot} = \sum_i F_i(\rho_{h,i}) + \frac{1}{2}\sum_i \sum_{j(\neq i)} \phi_{ij}(R_{ij}) \tag{3.3}$$

3.2 Method for Calculation of Vibrational Density of States in Real-Space

3.2.1 Continued Fraction Method

The continued fraction method using real-space Green's function is proposed by Haydock *et al.* [1] In this method the surface matrix is treated as a perturbation to bulk and then the surface Green's function is obtained by projecting the bulk Green's function into the subspace defined by the perturbation matrix. The method proposes the Green's function representation associated with the Eqn. 3.4 (the equation of motion for an atom at lattice site n with an atomic displacement $u_\alpha(\alpha = x, y, z)$ from its equilibrium position).

$$w^2 U_\alpha(n;k) = \sum_{n'\beta} D_{\alpha\beta}(n;n';k) U_\beta(n';k) \tag{3.4}$$

Considering a quantum system and a number N of orthonormal basis states $|\phi_i\rangle (i = 1, 2, \ldots, N)$. Starting from any given state $|f_0\rangle$ belonging to the space spanned by $|\phi_i\rangle$, and operating with H, *the recursion method provides a one-dimensional chain representation of the original quantum system.* When the operator H is applied to the initial state and subtract from $H|f_0\rangle$ its projection on the initial state, new state $|F_1\rangle$ orthogonal to $|f_0\rangle$ is obtained. In the method, diagonal elements of the Green's function $\langle f_0|H|f_0\rangle$ are expressed in terms of a continuous fraction whose coefficients are calculated from a chain of orthonormal states recursively generated from $|f_0\rangle$, [21]

$$|F_{n+1}\rangle = H|f_n\rangle - a_n|f_n\rangle - b_n|f_{n-1}\rangle \tag{3.5}$$

where

$$a_{n+1} = \langle f_{n+1}|f_{n+1}\rangle, \qquad {b^2}_{n+1} = \langle F_{n+1}|F_{n+1}\rangle. \tag{3.6}$$

Then $G_{00}(w^2)$ of the Green's function is then given by the continuous fraction expansion,

$$G_{00}(n_z, n_{z'}, w) = \langle f_0|\frac{1}{w^2 - H}|f_0\rangle = \cfrac{1}{w^2 - a_0 - \cfrac{{b^2}_1}{w^2 - a_1 - \cfrac{{b^2}_2}{w^2 - a_2 - \ldots}}} \tag{3.7}$$

where w is the frequency of the normal mode of the system and n_z is the layer number. Since the number of atoms increases very rapidly as the level of coefficients increases, one has to make a truncation for the level of coefficients of continued fraction which in turn leads to the inaccuracies in the calculations. Moreover, one has to use some method to extrapolate the coefficients for $n \ll N$, where n is the level of coefficient of continued fraction and N is the number of atoms in the crystal.

3.2.2 Real-Space Green's Function Method

In 1979 Dy *et al.* developed a technique, resolvent matrix method, to determine the Green's function for a block-tridiagonal matrix.[8] Wu *et al.* showed that the method of resolvent matrix provides a competent approach to simplify the calculation of the Green's function for large systems.[9] In a separate study, Wu *et al.* developed a convergence procedure to calculate the local Green's function associated with any specified locality in the system without any size affect related to the arbitrary truncation of the system.[10] However the efficiency of the convergence scheme was dependent on the existence of the recursive relation which links the calculations at a given step to those at previous step. To address this problem, a general recursive relation for the calculation of the local Green's function in the resolvent-matrix approach [11] was developed in the subsequent study by Wu *et al.* In this method the system under construction is basically divided into several subsystems as then it is possible to write the matrix describing the interactions in the system in block-tridiagonal form. Therefore the method allows one to work with much smaller matrices, whose dimensions are defined by the subsystems, than the interaction matrices for entire system.

3.2.2.1 A Brief Summary of the Resolvent Matrix Method

Consider a block-tridiagonal matrix as following,

$$H = \begin{pmatrix} \ddots & & & & \\ v_{i,i-1} & h_i & v_{i,i+1} & & \\ & v_{i+1,i} & h_{i+1} & v_{i+1,i+2} & \\ & & & \ddots & \end{pmatrix}, \tag{3.8}$$

where the sub-matrices h_i along the diagonals are $(n_i \times n_i)$ square matrices and the sub-matrices $v_{i,i+1}$ along the "off-diagonals" are matrices of dimension $3n_i \times 3n_{i+1}$ with n being the number of particles in the locality . The Green's function associated with the matrix H is given by

$$G(z) = (zI - H)^{-1} \tag{3.9}$$

where $z = w^2 + i\varepsilon$, ε being the width of the Lorentzian representing the delta function at w^2 and I is a unit matrix of the same size as that of H.

The diagonal elements of the Green's function matrix which lead to the local Green's function matrix corresponding to any chosen locality is given by [2],

$$G_{ii} = [(zI - h_i) - v_{i,i+1}\Delta^{+}_{i+1}v_{i+1,i} - v_{i,i-1}\Delta^{-}_{i-1}v_{i-1,i}]^{-1}. \tag{3.10}$$

(Δ_i^+) and (Δ_i^-) are defined as forward and backward partial Green's functions and described by

$$\Delta_i^+ = (zI - h_i - v_{i,i+1}\Delta_{i+1}^+ v_{i+1,i})^{-1}, \tag{3.11}$$

$$\Delta_i^- = (zI - h_i - v_{i,i-1}\Delta_{i-1}^- v_{i-1,i})^{-1}. \tag{3.12}$$

The relation between the diagonal elements of the Green's function matrix G is [2]

$$G_{ii} = \Delta_i^\pm + \Delta_i^\pm v_{i,i\mp1} G_{i\mp1,i\mp1} v_{i\mp1,i} \Delta_i^\pm. \tag{3.13}$$

As seen in the above equations, in the method of resolvent matrix, the calculation of Green's functions is reduced to the series of inversion and multiplication of matrices whose dimensions are usually much smaller than the total degree of the system under consideration. Another feature of the method is that one can focus on any specified locality in the system and calculate the corresponding Green's function, a diagonal element of the Green's function representing the entire system. This can be achieved through the forward and backward partial Green's functions matrix $\Delta_i^\pm$ as in (3.13).

3.2.2.2 Convergence Procedure

The convergence scheme to be outlined below is for the calculation of the diagonal elements of the Green's function which lead to the local vibrational density of states.

If the locality of interest is denoted by zeroth locality, then, from (3.10), the corresponding Green's function is

$$G_{00} = \{(zI_0 - h_0) - v_{0,1}\Delta_i^+ v_{1,0} - v_{0,1}\Delta_1^- v_{-1,0}\}^{-1}, \tag{3.14}$$

where h_0 is the force constant matrix for the zeroth locality while $v_{0\pm1}$ is the interaction matrix between the zeroth and (±1) localities. From recursive relations in (3.11),

$$\begin{aligned}\Delta_1 = \{&(zI_0 - h_0) - v_{0,1}[\ldots[\ldots - v_{m-1,m} \\ &(zI_m - h_m)^{-1}v_{m,m-1}]^{-1}\ldots]^{-1}v_{1,0}\}^{-1}\end{aligned} \tag{3.15}$$

where m goes to infinity. One can get a similar expression for Δ_1^- using (3.12).

From (3.15) it is clear that for an infinite system the calculation of $\Delta_1^\pm$ involves infinite series of matrix inversions and multiplications. To circumvent the infinite series, the following convergence procedure was introduced.

$$\Delta_1^+ = \lim_{m\to\infty} \Delta_1^{(+m)}(z), \tag{3.16}$$

where

$$\Delta_1^{+(m)} = \{(zI_0 - h_0) - v_{0,1}[\ldots[\ldots - v_{m-1,m} (zI_m - h_m)^{-1} v_{m,m-1}]^{-1} \ldots]^{-1} v_{1,0}\}^{-1}. \tag{3.17}$$

For a given ε at a certain w^2, a series of $\Delta_1^{+(m)}(z)$ are calculated until $\{\Delta_1^{+(m)}\}$ approaches its limit within a preset tolerance. The process is then repeated for a series of decreasing ε until a final limit, imposed by predefined criterion, is reached. One must, however, remember that there is no artificial boundary effects such as truncation of the system imposed from the beginning, since the accuracy of the results from the convergence procedure is dictated by preset convergence criterion rather than truncation of the system as it is in the method of continued fraction, another technique to determine local Green's functions.

As seen from the convergence procedure outlined in (3.16) and (3.17), the calculation of $\Delta_1^{+(m)}$ at present step is independent of $\Delta_1^{+(m-1)}$ at the previous step. Thus Δ_1^{+} at each step must be calculated separately which in turn results in excessive computing time. To eliminate the redundancy in the computation of $\{\Delta_1^{+(m)}\}$, a general recursive relation to link the calculation of $\Delta_1^{+(m)}$ to that of $\Delta_1^{+(m-1)}$ was introduced by Wu *et al* [2]. The recursive relation is general as it is applicable to all situations regardless whether the dimensions of the diagonal blocks forming the block-tridiagonal matrix are the same or different. A brief summary of the recursive relation will be summarized in the next section.

3.2.2.3 Formalism for the Recursive Relation

Consider a block-tridiagonal matrix in the following form

$$H = \begin{pmatrix} h_1 & v_{12} & & \\ v_{12} & h_2 & v_{23} & \\ & v_{32} & h_3 & v_{34} \\ & & & \ddots \end{pmatrix}. \tag{3.18}$$

The H matrix can be used to describe a spherically growing crystal or a solid with a surface. In the first case the dimension of h_i increases with increasing i. Contrary to the case in a spherically growing crystal, the size of h_i for a solid with a surface does not change with increasing i, since the semi-infinite crystal is divided into equally-sized localities. For a semi-infinite crystal, the matrix h_1 contains the top layers of the solid and describes the interactions between the particles within the locality. The sub-matrices h_i and $v_{i,i\pm1}$ for $(i > 1)$ are, respectively, the interaction matrices within and between consecutive localities (or layers) in the bulk.

From (3.10) - (3.12), the local Green's function associated with h_i is the partial forward and backward Green's functions $\Delta_1^{\pm}$. (3.16) at stages from $m = 1$

to $m = m$ yields

$$\begin{aligned}
\Delta_1^+(1) &= (zI_1 - h_1)^{-1} \equiv \Delta_1^- \equiv G_{11}^{(1)}, && (3.19)\\
\Delta_1^+(2) &\equiv G_{11}^{(2)}, && (3.20)\\
\Delta_1^+(3) &\equiv G_{11}^{(3)}, && (3.21)\\
&\vdots && (3.22)\\
\Delta_1^+(m) &\equiv G_{11}^{(m)} && (3.23)
\end{aligned}$$

where $G_{11}^{(m)}$ is the first diagonal block matrix of the Green's function corresponding to the $H^{(m)}$ which is the matrix resulting from truncation of the matrix H at mth step. The procedure to obtain a recursive relation linking $\Delta_1^{(m)}$ to $\Delta_1^{(m-1)}$ is as following.

Step 1: $m = 1$, the matrix H is truncated after first locality (**i.e.** there is no other localities but locality 1),

$$H^{(1)} = h_1, \tag{3.24}$$

then the corresponding Green's function matrix at step 1 is

$$G^{(1)} = G_{11}^{(1)}. \tag{3.25}$$

From (3.10)

$$G_{11} \equiv (zI - h_1)^{-1} \equiv \Delta_1^- \equiv \Delta_1^+ \equiv \Delta_1^{+(1)}. \tag{3.26}$$

Step 2: add the consecutive locality. Then the corresponding H and G matrices are

$$H^{(2)} = \begin{pmatrix} h_1 & v_{12} \\ v_{21} & h_2 \end{pmatrix}, G^{(2)} = \begin{pmatrix} G_{11}^{(2)} & G_{12}^{(2)} \\ G_{21}^{(2)} & G_{22}^{(2)} \end{pmatrix}. \tag{3.27}$$

Using (3.13), we have

$$\Delta_1^{+(2)} \equiv G_{11}^{(2)} = \Delta_1^- + \Delta_1^+ v_{1,2} G_{22}^{(2)} v_{2,1} \Delta_1^-. \tag{3.28}$$

From (3.10)

$$G_{22}^{(2)} \equiv \Delta_2^- = \{(zI - h_2) - v_{2,1}\Delta_1^- v_{1,2}\}^{-1}. \tag{3.29}$$

Inserting (3.29) into (3.28) yields

$$\Delta_1^{+(2)} \equiv \Delta_1^{+(1)} + \Delta_1^- v_{1,2} \Delta_1^- v_{2,1} \Delta_1^-. \tag{3.30}$$

Let

$$A_1 = \Delta_1^- v_{1,2} \tag{3.31}$$

and

$$B_1 = v_{2,1}\Delta_1^-. \tag{3.32}$$

Then (3.30) takes following form

$$\Delta_1^{+(2)} \equiv G_{11}^{(2)} \equiv \Delta_1^{+(1)} + A_1 \Delta_2^- B_1. \tag{3.33}$$

Step 3: add the succeeding locality. Then the matrix H truncated at step 3 and the Green's function matrix associated with it are

$$H^{(3)} = \begin{pmatrix} h_1 & v_{1,2} & 0 \\ v_{2,1} & h_2 & v_{2,3} \\ 0 & v_{3,2} & h_3 \end{pmatrix}, \tag{3.34}$$

$$G^3 = \begin{pmatrix} G_{11}^{(3)} & G_{12}^{(3)} & G_{13}^{(3)} \\ G_{21}^{(3)} & G_{22}^{(3)} & G_{23}^{(3)} \\ G_{31}^{(3)} & G_{32}^{(3)} & G_{33}^{(3)} \end{pmatrix} \tag{3.35}$$

Using (3.13), we have Δ_1 at this step as following

$$\Delta_1^{+(3)} \equiv G_{11}^{(3)} = \Delta_1^- + \Delta_1^- v_{1,2} G_{22}^{(3)} v_{2,1} \Delta_1^-. \tag{3.36}$$

An expression for $G_{22}^{(3)}$ in the above equation can be obtained from (3.10). Thus,

$$G_{22}^{(3)} = \Delta_2^- + \Delta_2 v_{2,3} G_{33}^{(3)} v_{3,2} \Delta_2^-. \tag{3.37}$$

Substituting (3.37) into (3.36) gives

$$\begin{aligned} \Delta_1^{+(3)} \equiv G_{11}^{(3)} = \Delta_1^- + \Delta_1^- v_{1,2} \Delta_2^- v_{2,1} \\ + \Delta_1^- v_{1,2} \Delta_2^- v_{2,3} G_{33}^{(3)} v_{3,2} \Delta_2^- v_{1,2} \Delta_1^-. \end{aligned} \tag{3.38}$$

The first term in (3.38) is equivalent to $\Delta_1^{+(2)}$. Letting $A_2 = \Delta_1^- v_{1,2} \Delta_2^- v_{2,3}$ and $B_2 = v_{3,2} \Delta_2^- v_{1,2} \Delta_1^-$ leads to the next equation

$$\Delta_1^{+(3)} \equiv G_{11}^{(3)} = \Delta_1^{+(2)} + A_2 \Delta_3^- B_2 \tag{3.39}$$

with

$$A_2 = A_1 \Delta_2^- v_{2,3}, B_2 = v_{2,3} \Delta_2^- B_1.$$

By adding more localities, one eventually reaches the following recursive relation

$$\Delta_1^{+(m)} = \Delta_1^{+(m+1)} + A_{m-1} \Delta_m^- B_{m-1} \tag{3.40}$$

where

$$A_{m-1} = A_{m-2} \Delta_{m-1}^- v_{m-1,m} \tag{3.41}$$

and

$$B_{m-1} = v_{m,m-1} \Delta_{m-1}^- B_{m-2}. \tag{3.42}$$

The steps to calculate the terms in the sequence $\{\Delta_1^{(m)}\}$ can be summarized as following

(1) using (3.20) calculate Δ_1^-

(2) from (3.31) and (3.32) evaluate A_1 and B_1

(3) from (3.11) and (3.12) calculate the subsequent Δ_m^-

(4) using recursive relation (3.40) compute the successive terms $\Delta_1^{(m)}$

(5) determine A_{m-1} and B_{m-1} through (3.41) and (3.42). Thus, using the results obtained in the previous step, one can evaluate the the terms in the present step.

In short, once the force constant matrix is built in block-tridiagonal form, the Green's function matrix corresponding to the local region of interest is constructed following the procedure described in the above section. Then using the diagonal elements of the local Green's function, one can determine the normalized vibrational density of states associated with locality l as a function of w^2, where w is vibrational frequency and equal to $2\pi\nu$.

3.2.3 A Brief Review of Density of States and Green's Function

Once the normalized eigenfunctions ψ_m and eigenvalues E_m ,are known ,corresponding to the system with Hamiltonian H, then total density of states of the system is defined as following.

$$N(E) = \sum_m \delta(E - E_m). \tag{3.43}$$

The projected density of states associated with the chosen state of interest on a basis of local normalized orbital $|f_0\rangle$ (normalized to unity) is defined as

$$n_0(E) = \sum_m |\langle f_0|\psi_m\rangle|^2 \delta(E - E_m), \tag{3.44}$$

where ψ_m and E_m indicate normalized eigenfunctions and eigenvalues of H.[21] The total density-of-states can be expressed as the sum of the local density-of-states on any complete orthonormal set $\{f_n\}$.

The Green function corresponding to the operator H is defined by

$$G(E + i\epsilon) = \frac{1}{E + i\epsilon - H}. \tag{3.45}$$

The diagonal matrix element of the Green's function is

$$G_{00}(E + i\epsilon) = \langle f_0|\frac{1}{E + i\epsilon - H}|f_0\rangle. \tag{3.46}$$

Then, inserting the unit operator $1 = \sum |\psi_m\rangle\langle\psi_m|$ into Eqn. 3.46 yield

$$G_{00}(w^2 + i\epsilon) = \langle f_0| \sum_m |\psi_m\rangle\langle\psi_m| \frac{1}{w^2 + i\epsilon - H} |f_0\rangle \tag{3.47}$$

$$= \sum_m |\langle f_0|\psi_m\rangle|^2 \frac{1}{w^2 + i\epsilon - w_m^2} = \sum_m |\langle f_0|\psi_m\rangle|^2 \frac{w^2 - w_m^2 - i\epsilon}{(w^2 - w_m^2)^2 + \epsilon^2}$$

From Eq.3.47, for any $\epsilon > 0$, $G_{00}(E + i\epsilon)$ is analytic and its imaginary part is negative. On the real energy axis, the real part of $G_{00}(E)$ exhibits poles which constitutes the discrete eigenvalues of H, while the imaginary part exhibits δ-like singularities; this can be seen keeping the limit $\epsilon \to 0^+$ in Eq.3.47 and using the result

$$\lim_{\epsilon \to 0^+} \frac{1}{\pi} \frac{\epsilon}{(E - E_m)^2 + \epsilon^2} = \delta(E - E_m). \tag{3.48}$$

Using Eq.3.44 and Eq.3.47, the standard spectral theorem can be obtained as follows

$$n_0(E) = -\frac{1}{\pi} \lim_{\epsilon \to 0^+} Im\{G_{00}(E + i\epsilon)\}. \tag{3.49}$$

Then the total density of states is

$$N(E) = -\frac{1}{\pi} \lim_{\epsilon \to 0^+} Im[Tr\{G_{ii}(E + i\epsilon)\}]. \tag{3.50}$$

Once the Green's function corresponding to a locality of interest is calculated, then the normalized total density of states is obtained through

$$N_i = 2w g_i(w^2) \tag{3.51}$$

where the function $g_i(w^2)$ satisfies the equation

$$g_i(w^2) = -\frac{1}{3n_i\pi} \lim_{\epsilon \to 0^+} \{Im[Tr(G_{ii}(w^2 + i\epsilon))]\} \tag{3.52}$$

In the above equations, G_{ii} is the Green's function sub-matrix associated with locality 'i' and n_i is the number of atoms in this locality. [21]

CHAPTER 4

PARALLEL COMPUTATION METHODS AND DETAILS

In an attempt to develop a parallel version of a code, one needs to first optimize the serial code and than examine the runtime performance of the code. Therefore, an analyzer is an inevitable tool in examining the optimizations for program statements involving cache missing operations, time consuming calculations such as input/output operations and matrix multiplications. Even after the optimizations are carried out, there can still be a need for a parallel version of the program to decrease the total computational time of the serial program. At this stage, the programmer encounters with the choice of parallel programming tools or environments.

Today, the commonly used parallel programming tools are divided into two categories, depending upon the architecture of the computing systems. These are the shared memory based parallel programming tools and the cluster based parallel programming tools. In any of these categories the tools are composed of parallel programming libraries or environments. Nowadays, the popular shared memory parallel programming tool is OpenMP and the popular cluster based parallel programming tool is Message Passing Interface (MPI). However, one has to keep in mind that classifying MPI as a cluster based parallel programming tool is not a restriction on running the programs, developed using MPI, on the NUMA (Non-Uniform Memory Access) shared memory computing systems. (further reading for details about NUMA systems; [12])

A thread is an execution stream in the context of a thread state. The difference between processes and threads is that multiple threads share parts of their state. Typically, multiple threads can read and write same memory, however, processes can not directly access memory of another process. In shared-memory programs the program begins execution as a single thread which is called master thread. At the costly calculation code portions where parallel operations are required, the master thread forks additional threads and at the end of parallel code the created threads join to the single master thread.

In using OpenMP to parallelize a serial code, the most commonly

followed technique is just compute the costly calculation code portions with parallel threads using OpenMP clauses called 'pragma'.In many cases, writing an OpenMP specified parallelization clauses before a simple loop can handle the calculations with parallel processes within the loop. However, OpenMP can not handle the loop nests containing loop-break statements and pointer operations which can generally be mishandled by OpenMP environment or OpenMP threads.[22] Since RSGF algorithm contains several loop nests and the main loop contains loop-break, MPI parallelization tool is chosen to achieve the parallelism of RSGF method. In other words, OpenMP can only handle loops or code portions that are specified in OpenMP environment. Although OpenMP standardized loops are automatically parallelized, the manually parallelized code portions have to be defined to OpenMP environment in 'pragma' clauses with special statements. These special statements must define the global variables and thread dependent variables to acquire reliable results for calculations to accomplish the parallel tasks. Another shortcoming of OpenMP is that the threads' initializations take some time in the operating system side, which may increase the total calculation time. [13]

In MPI the performance of the parallel program is up to the programmer's skills as MPI is considered as a library rather than an environment like OpenMP. Hence, every single step in parallel computation depends on the correct use of MPI libraries. The right location of functions such as send and receive routines for the distribution in the code can also optimize the parallel program.[14] However, one has to keep in mind that it takes more time to initialize the parallel processes in MPI than in OpenMP. The initializing time depends on the secure connection times of cluster machines and the distribution time depends on the data size transferred between processes. So, these time costs are musts in distributed computations with cluster-based parallel programming tools.

Although MPI has a disadvantage in data retrieve, the super computer architecture is heading towards to cluster systems, since nowadays the issue of the ratio of performances to prices of super computers becomes a decisive factor more than ever.[15] While shared memory systems cost more than million dollars, the cluster based systems cost between thousands and hundred thousands of dollars. In addition to this cost wise advantage of MPI, both shared-memory machines and cluster machines can even have same performances for computations such as thousands of floating point operations in a second.

Since RSGF algorithm contains several loop-nests and the main loop contains loop-break statement, which are the major drawbacks of OpenMP, MPI parallelization tool is chosen to achieve the parallelization of the RSGF method.

CHAPTER 5

APPLICATION DETAILS

5.1 Simulation Model

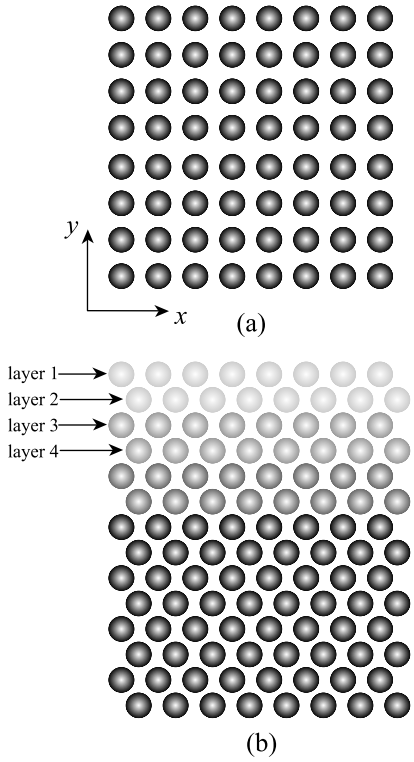

Figure 5.1: The figure shows the Cu crystal formed in fcc structure. (a) the top view of the simulation model (b) the side-view of the Cu crystal which the lighter atoms are the surface atoms, the darker ones indicate the atoms at deeper layers towards the bulk.

The (100) flat surface of Cu crystal is chosen as a prototype system to test the parallel version of the implementation of the real-space Green's function technique, developed in this Thesis. The Cu(100) surface is a flat surface created by cutting the Cu crystal along one of the members of the family of <100> directions. The surface atoms posses the same symmetry along the directions parallel to the surface is given in Fig 5.1. However, the symmetry along the

direction normal to the surface is broken due to the existence of the surface. As illustrated in Fig.5.1, while the x and y axes defines the surface plane, the z axis is chosen to be along the surface normal.

In this Thesis, the RSGF method for calculating the local vibrational density of states is implemented through layer-by-layer approach in which an infinite system with some periodicities is viewed as two-dimensional atomic layers stacked one upon the other along some crystal axis. The number of layers to be included in a particular locality is dictated by the range of the interatomic potential. For the EAM potential for Cu, atoms in layer 1 on Cu (100) have negligible interaction with those in layer 4, hence the locality neighboring the first one need only extend to the fourth layer. There are two prototype test systems. The first prototype system is chosen relatively small to obtain faster test results. In each layer there are 12 chains for the test system and each chain contains 8 atoms as shown in Fig. 5.1, layer 1 and layer 2 is designated as the first locality, layer 3 and layer 4 is designated as the second locality and following layers are designated in same way. The second prototype test systems is chosen relatively large to perform a test of complex simulation model. In this large test system, there are also 12 chains however this time each one contains 12 atoms. Instead of containing 2 layers, this model contains 6 layer in each locality and it is also constructed with 4 localities. The aim to choose such a second test system is that there is a necessity to test the performance of parallel algorithm with bigger sized sub-matrices. On the other hand, for both test simulation models the computational cell contains 20-layers of atoms, which is tick enough to prevent the interaction of top and bottom surfaces' atoms.

In the simulation, periodic boundary conditions are applied along the x and y directions, while no such constraint is imposed along the z-direction. Atoms are initially placed in the bulk-terminated positions. The $0K^0$ equilibrium configuration is determined by minimizing the total energy of the system using the conjugate gradient method. [20]

5.2 Test Environment Details

The test environment is made up of two cluster systems: one is a Solaris v9 cluster of SUN FIRE v210 systems with 48 UltraSPARC III processors grouped into 24 nodes. In each node the system contains two processors communicating over a faster bandwidth and each processors has 4MB cache. The traditional libraries such as LAPACK, LINPACK, BLAS, and BLAS3 are available and packed under SUN's performance libraries and several parallel computation tools like OpenMP and MPI are provided in SUN parallel computation tools.[17]

The second cluster system is an HP supercomputer made up of 42 nodes, of which 21 node contain two 3.4 GHz Intel Xeon with EM64T module and rest of the nodes contain two 3 GHz Intel Xeon with EM64T module. The operation system of these nodes is Red Hat Linux whose cluster development tools are provided with PGI Cluster Development Kit(CDK).[18] In CDK, there are also traditional libraries such as LAPACK, BLAS, SCALAPACK and all are optimized for Intel Xeon processors. Nonetheless, there are parallel programming libraries such as MPI in PGI CDK.

5.3 Parallel Implementation of RSGF Method

5.3.1 Pseudocode of the Algorithm

```
STEP 1: Read initial settings (iteration limit,
        chain number, number of atoms)
STEP 2: Read H1, H2, H3, H4, V12, V23, V34 sub-matrices
STEP 3: Start the clock.
--PARALLEL COMPUTATION STEPS START HERE--
STEP 4: Do steps until step number 22 for frequencies (w)
        STEP 5: Compute delta at stage 1 (Make all elements
                of delta matrix zero. Fill the matrix with
                complex h1. Inverse delta matrix)
        STEP 6: Prepare the ingredients for computing delta
                at stage 2. (Make all elements of v matrix
                zero and Fill the matrix with complex v12.)
        STEP 7: Compute A1 and B1.
                Do matrix operation: Trans(v)*delta*v
        STEP 8: Compute forward Green's function at stage 1.
                Make elements of delta matrix zero and fill
                delta matrix with complex h2 minus temp.
                Inverse delta. Do matrix operation:
```

$$A1\Delta A1^t$$

```
        STEP 9: Compute A2 and B2 for the next iteration.
                A2: make all elements of v matrix zero and
                fill the matrix with v23.
                Do matrix operations:
```

$$A1\Delta V$$

```
        STEP 10: Compute forward Green's at stage 2.
                 Do matrix operations:
```

$$A1\Delta A1^t$$

```
        STEP 11: Compute A3 and B3 for the next iteration.
                 A3: make all elements of V matrix zero and
                 fill the matrix with V34. Do operations:
```

$$A1\Delta V$$

```
        STEP 12: Compute forward Green's function at stage 3.
                 Make elements of delta matrix zero and fill
                 Delta matrix with H4. Do following operations:
```

$$\Delta - V^t \Delta V$$

$$\Delta'$$

```
        STEP 13: Start the iteration for convergence.
                 Do until maximum iteration number reached.
            STEP 14: Compute Delta at present stage.
            STEP 15: If the ratio of
```

$$A1\Delta A1^t$$

```
                 to Stage 3 Delta is equal to the
                 convergence limit then converged.
                 STEP 16: Compute the vibrational density of
                          states. Compute chain density of
                          states for each chain.
                 STEP 17: Write the results to the output file.
            STEP 18: Else
                 STEP 19: Compute A and B for the next iteration.
```

$$A1\Delta V$$

```
                 STEP 20: Compute forward Green's function at
                          present stage. Do operations:
```

$$(H_{Bulk} - V^t \Delta V)'$$

```
            STEP 21: Take action for the next iteration.
STEP 22: Take action for the next frequency calculations.
--PARALLEL COMPUTATION ENDS HERE--
STEP 23: Print out the calculation and total
         time consumed by the program
STEP 24: End program
```

5.3.2 Implementation Details

The above pseudocode shows the steps taken in the serial code (when "parallel starts here" and "ends here" statements are removed) during a run. The programming language of the serial and parallel versions of the code is Fortran 77. Before attempting parallelization, the serial code is carefully studied

for optimization purposes. The steps' statements were investigated for cache misses and unnecessary operations such as bad memory usages. The carried out optimizations in the code are: (1) some of the loops used in IO operations and matrix assignments containing cache misses are optimized for correct cache usage regarding column-by-column storage of two-dimensional arrays in Fortran. (2) matrix operation subroutines are replaced by LAPACK and BLAS3 routines or equivalently corresponding routines in SUN Performance Library which are tested for several years and are known reliable with regarding to the optimization purposes.[16],[17],[18] After the serial code optimizations,

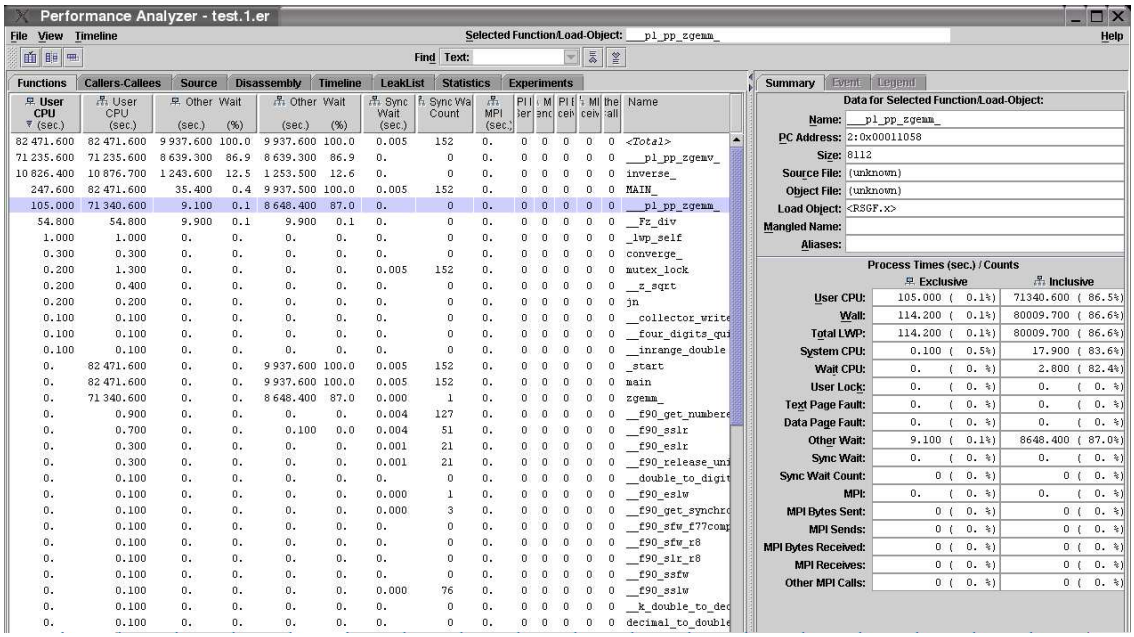

Figure 5.2: A screen shot of the SUN's Performance Analyzer.

SUN's performance analyzer is used to examine the runtime performance. In Fig. 5.2, a snapshot of the SUN's Performance Analyzer is presented, showing how much cpu time is used by different parts of the serial program. Here, 'zgemm' shows the matrix multiplication routine of Lapack 2.0 in the SUN Performance Library, and 'inverse' shows the optimized matrix inversion routine. In fact, both are the most costly calculations in this algorithm. Right-hand side of the snapshot shows the percentage of the time consumed by 'zgemm' sub-routine; %86.5 in user CPU time. The snapshots of the analyzer showing the statistics and time-line of the serial code can be seen in Fig. 5.3, 5.4 and 5.5. In general, for an optimized program code, most of the time must be consumed in 'User CPU' time. However, the analyzed code shows that %12.8 of the time consumed with 'Other Wait' (see Fig. 5.4). In fact 'Other Wait' shows the time consumed for the cpu of other system resources.

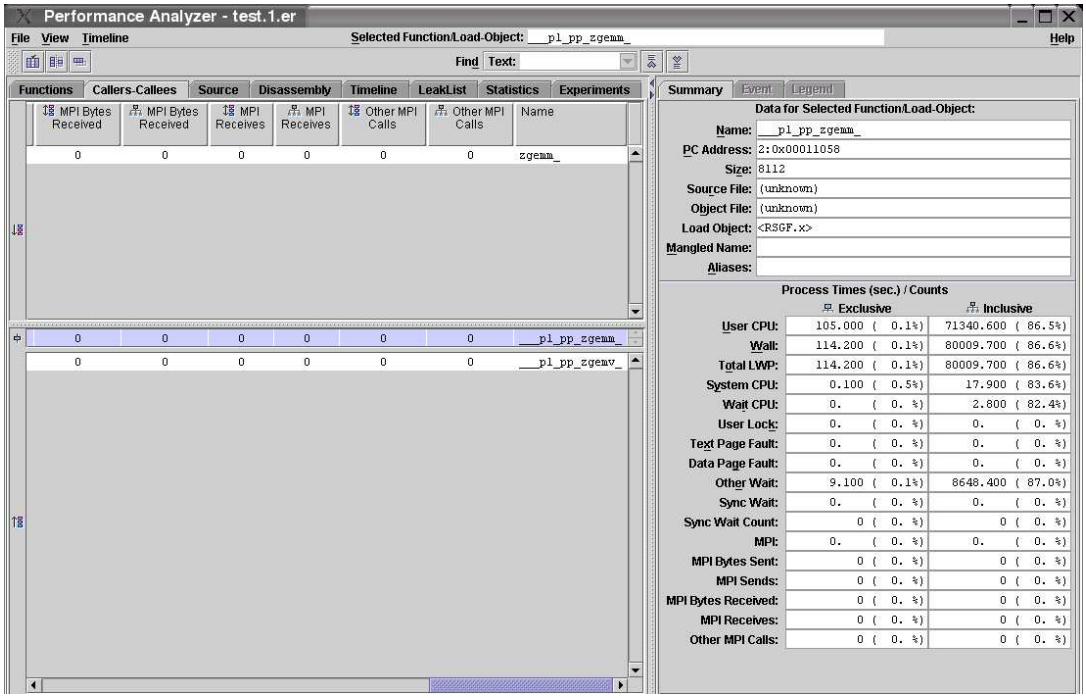

Figure 5.3: A screen shot of the SUN's Performance Analyzer showing the 'zgemm' sub-routine caller and its callee 'zgemv' matrix-vector multiplication. SUN Performance Library matrix multiplication sub-routine uses 'zgemv' matrix-vector multiplication for an optimal calculation.

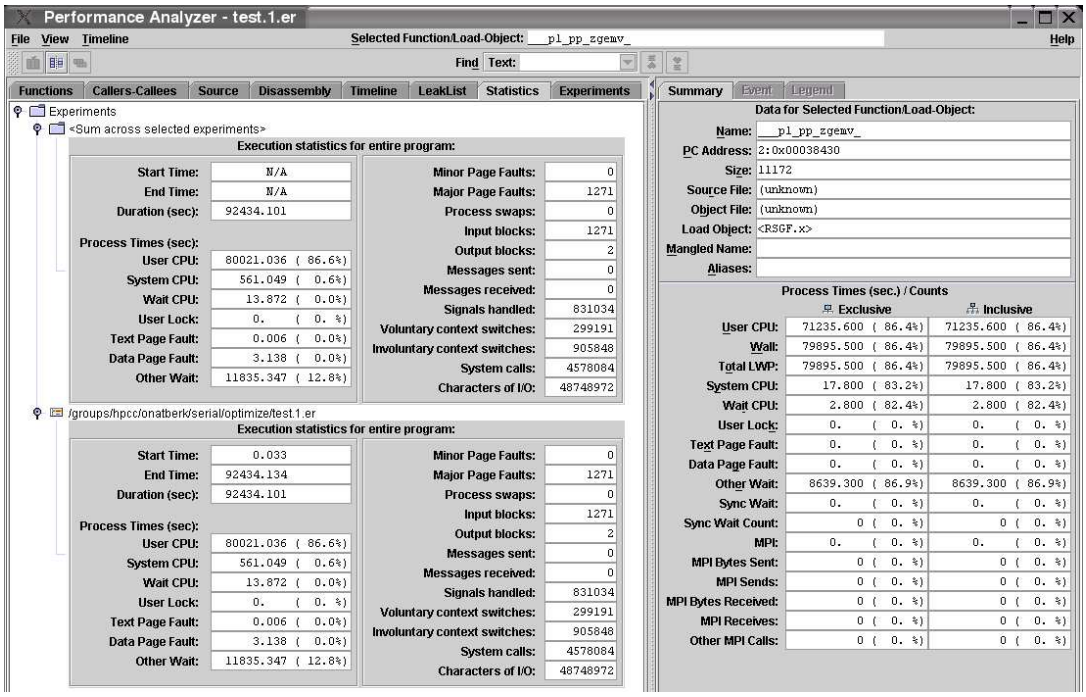

Figure 5.4: A screen shot of the SUN's Performance Analyzer showing the process time statistics for entire program in percentage with respect to the total consumed time.

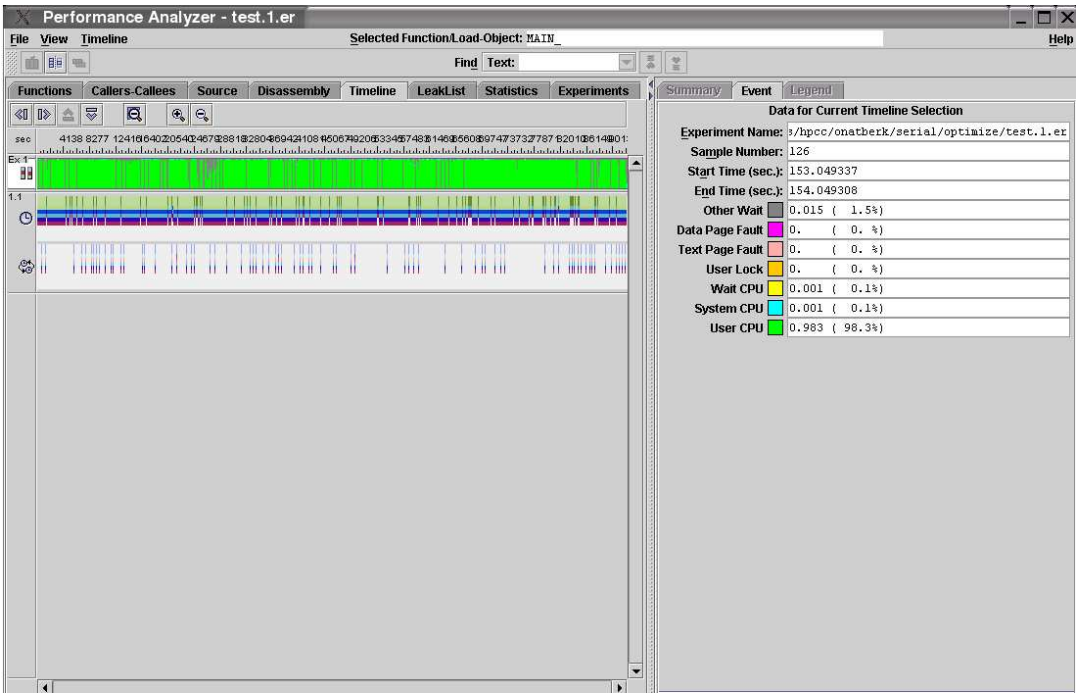

Figure 5.5: A screen shot of the SUN's Performance Analyzer showing how much cpu time is used by the serial program with a time-line diagram. The percentage of the 'User CPU' time (in green) is much more than the other process as expected for a well written optimized code.

In the analyzer output of the algorithm, the results are written to a file with NFS file system, which causes waiting times when NFS server is heavily loaded with other jobs' requests or the network to the NFS server is in a heavy traffic. Even for the relatively small prototype system specified in the previous chapter, the serial code runs consume over 10 hours on SUN test environment. The snapshot reveals that the most time consuming parts for the serial run are the matrix operations: matrix multiplication and inversion. One way to reduce these timely costs is to directly parallelize the subroutines that perform the matrix operations. However, in RSGF technique, the calculations of local density of states (LDOS) for a single frequency with a single iteration for the convergence requires 20 matrix multiplications. This yields at least 4000 matrix multiplications to complete calculations for the whole frequency spectrum with 200 frequency points. If the program involve so many multiplications of small matrices then it wouldn't be wise to parallelize the matrix operations as the send and receive routines for the distribution of matrix elements would slow down the calculations. Another approach would be to distribute the computation of LDOS for each frequency to the processors used since the calculations of density of states for each frequency are independent of one another. Thus yielding no data dependency between each process. Fig. 5.6 shows the flowchart of the program. There one can see the flow independency for the frequency calculations. Since

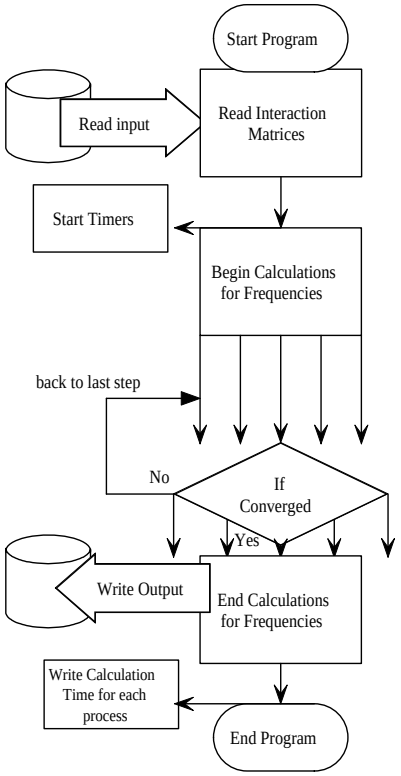

Figure 5.6: The flowchart of the program showing the main operations taken by each process. The parallel part is represented with several parallel arrows and the convergence procedure is indicated with an 'if' statement.

this type of parallel programming is called embarrassingly parallel programming, the algorithm of RSGF calculation for frequency spectrum is an embarrassingly parallel algorithm. The independent computations for each process are the steps numbered between 3 and 22 in the pseudocode. Although in many cases the flow of the parallel and serial codes are different, in this Thesis specific problem the serial code becomes the same as the parallel one when parallel initialization and distribution statements are removed. On the other hand, for this method of parallelization there could be no need for send and receive routines for the distribution of frequencies if several statements are written in the parallel code to define the frequency range for each processes with an algorithm. However, there are several ways to distribute the frequencies to the processes and but some may cause unbalanced load distributions due to the convergence procedure that may take more time or iteration for different frequencies. To circumvent the unbalanced load distributions a random distribution approach could be an innovative solution. However, with no knowledge of which frequencies take more iteration and time, the distribution of frequencies with an interval can also be a random distribution action. With this in mind, first in the frequency range is divided into the number of processors and the resultant sub-ranges are distributed to the processors as shown in Fig 5.7. This also causes an unbalanced

load distribution. Secondly, the frequencies are distributed to the processors with cyclic distribution by assigning each processor almost the same number of frequencies to calculate. That is to say, if processor 0 gets first frequency, processor 2 gets the second and this goes on with consecutive frequencies through the frequency range. After these parallelization attempts, the final parallel code is tested for the scalability, efficiency and speed-up with several simulation models.

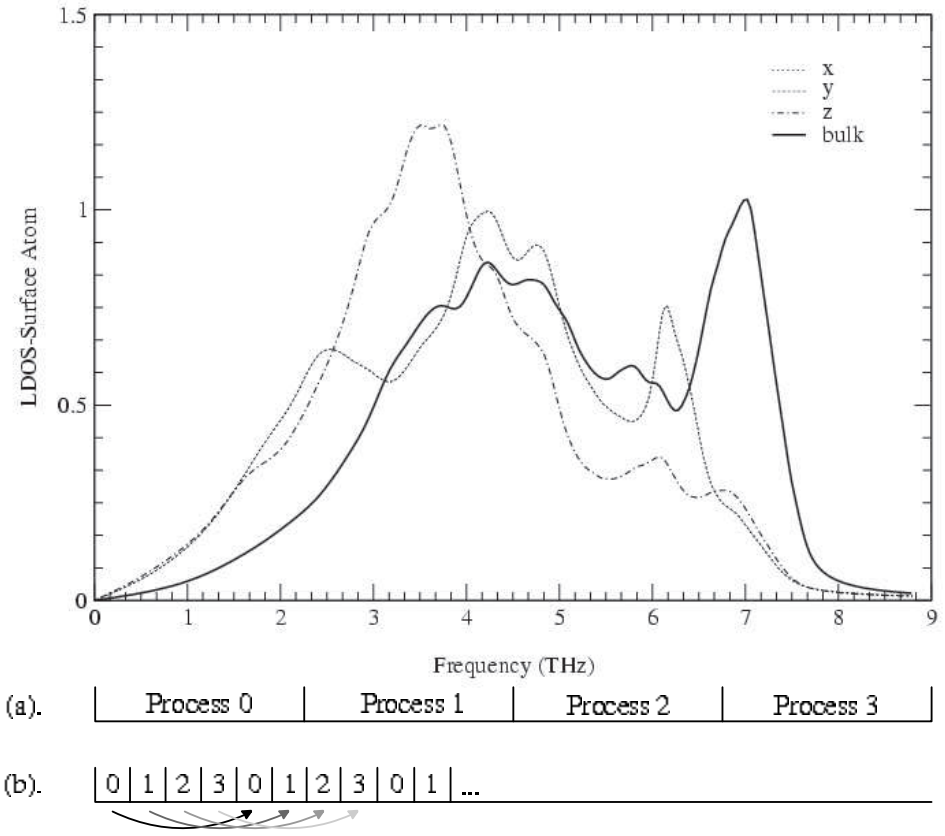

Figure 5.7: The two different distribution algorithms are tested in parallel runs (a) the frequencies distributed with a range to the processors (b) the frequencies distributed with a cyclic (interlived) distribution

CHAPTER 6

RESULTS AND DISCUSSIONS

6.1 Vibrational Density of States for a Surface atom of Cu(100)

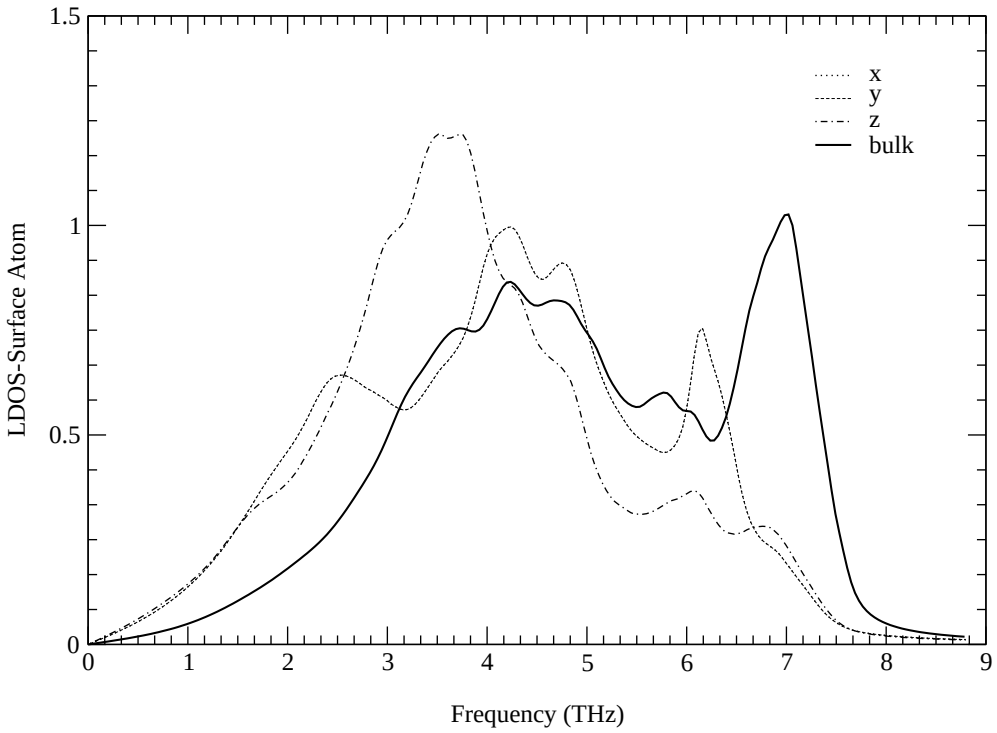

Figure 6.1: The calculated local vibrational density of states (LDOS) for a surface atom of Cu(100) along the x-, y-, and z- directions for the first test model whose locality contains 24 chains.

The LDOS along the x-, y-, and z- direction for a surface atom of first test model is presented in Fig. 6.1 using the parallel algorithm developed in this Thesis. On the other hand, Fig. 6.2 shows the LDOS along the x-, y-, and z-directions for a surface atom for the second test model. According to the figures

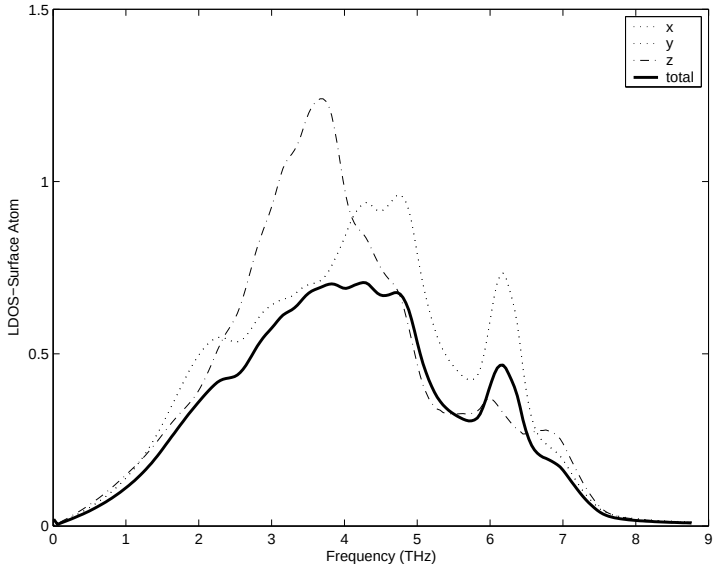

Figure 6.2: The calculated local vibrational density of states (LDOS) for a surface atom of Cu(100) along the x-, y-, and z- directions for the second test model whose locality contains 72 chains.

6.1 and 6.2 the LDOS curves are the same for each x-, y-, and z- directions. These results are expected because both of the test models simulate the same surface atoms of Cu(100) and both test models simulate the same bulk which is constructed with Cu atoms. Let me remind that ϵ in Eqn. 3.52 is the width of the Lorentzian representing the delta function at w^2 and it determines the sharpness of the peaks in LDOS. The smaller ϵ, the sharper the peaks are. Here ϵ is taken to be 0.4 for the calculations of LDOS for both a surface atom and the bulk-like atom. The curve for LDOS, therefore, are smoother. Note that the LDOS of a surface atom shows completely different characteristics, reflecting the different nature of the force fields between the atoms at the surface with respect to the ones in the bulk. The LDOS's along the x- and y- direction are strictly degenerate for symmetry reasons shown in Fig. 6.1. While the major weight in LDOS along these symmetry directions is in the low frequency region, it is in the relatively higher frequency region along the z- direction. Also, the low frequency modes of surface atom are shifted overall toward lower frequencies. These shifts are more pronounced along the x and y directions than those along the z- direction.

6.2 Benchmark Results for Parallel Program

The aim of a parallel computation is to use p number of processors to execute a sequential program p times faster than it executes on a single processor. Such 100 percent speedups is a rare case in reality. In most cases benchmarking a parallel program may not result as expected because of parallel overheads (such as communication operations and redundant computations) and execution

environment overheads (such as other running program's overheads). Therefore, in analyzing a parallel program one needs to examine the scalability, speedup, and efficiency of a parallel program.[14]

Speedup is defined as the ratio of the sequential execution time to parallel execution time. Efficiency of a parallel computation, on the other hand, is the speedup divided by the number of processors used. The scalability of a parallel program is a measure of its ability to increase performance as the number of processors increases.

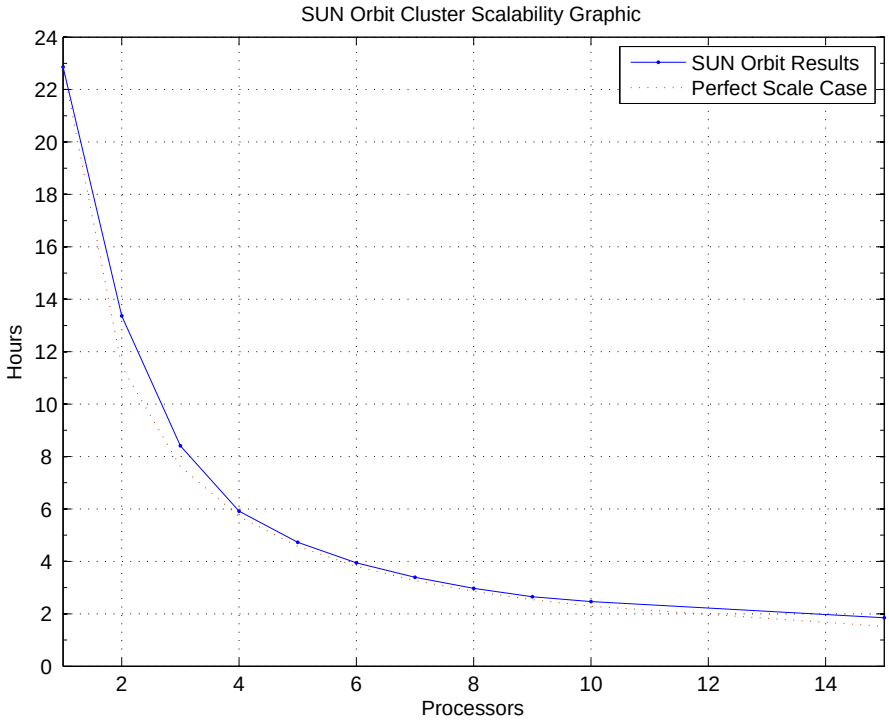

Figure 6.3: The scalability graphic of first test model for the SUN cluster test environment.

The serial version of the code runs more than 22 hours for the first prototype model using the first test environment. These results are confirmed by several runs. When the parallel version of code is run on parallel environment using different number of processors, the computation time decreases remarkably. Fig 6.3 illustrates the scalability of the parallel implementation. As seen in the figure, the initial drops in computation time are close to the ideal case. For example, the computation time with six processors is $\frac{1}{6}$ that of the serial code, indicating that the problem at hand is close to an embarrassingly parallel algorithm as discussed in previous chapter.

The second test environment results of the first test model for the

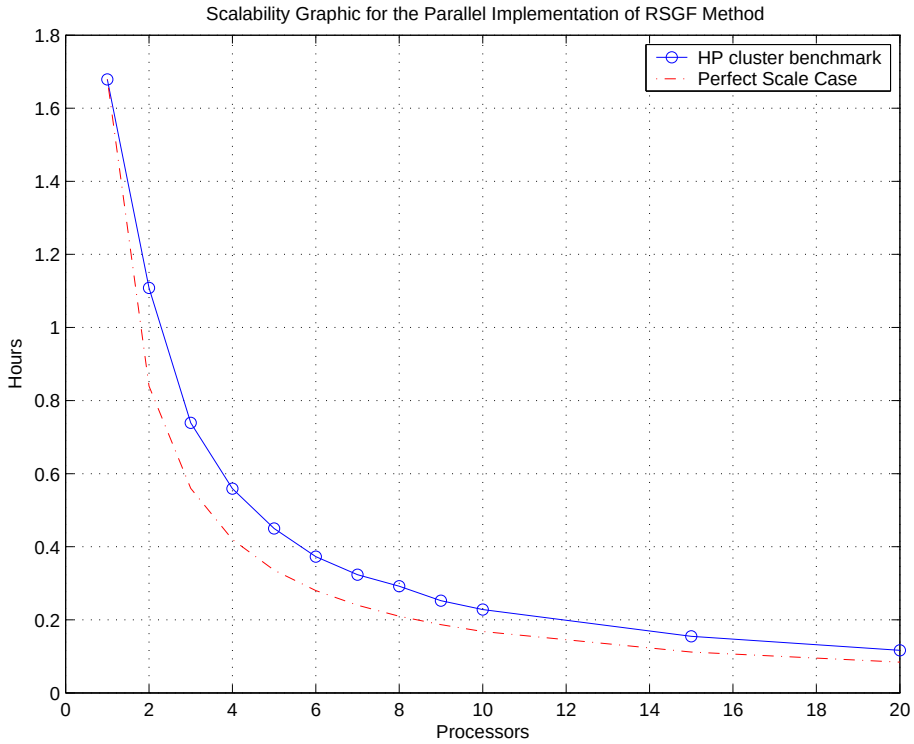

Figure 6.4: The scalability graphic of first test model for the HP cluster test environment.

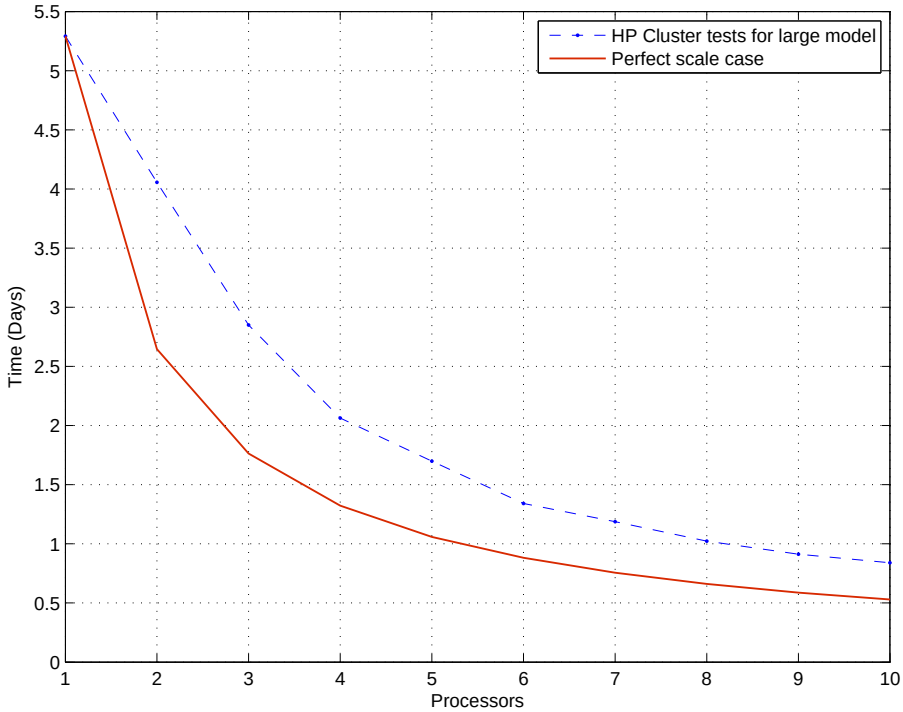

Figure 6.5: The scalability graphic of second test model for the HP cluster test environment.

scalability

is presented in Fig. 6.4. The serial version of the code runs more than 100 minutes, which is over 12 times faster than the run on the first test environment. This is because, the CPUs of the two test environment are different in speed and performance. The scalability of the parallel program run on the SUN cluster system scales better than that on HP cluster system. However, one may worry about the difference between the perfect scale assumptions and the parallel run results on the HP cluster compared to the SUN cluster scale results. SUN cluster results seem to scale almost perfectly for parallel runs. To understand the difference between two test system runs, the analysis of the speed-ups graphic for both systems could help. Fig. 6.5 shows the speedup graphics for the two test systems. As seen in the Figure, the speedup ratio of the SUN cluster appears to be better than that of HP cluster. The reason that the SUN machines passes high speed-up ratios is the cache performance of SUN CPUs. While HP Intel CPUs have a 2MB cache, SUN UltraSPARC III CPUs have 8MB caches. That means, SUN CPU's may not have cache misses for matrix operations, thus resulting in faster calculations. The results confirms that SUN CPUs have a better cache performance than the Intel CPUs.[17],[19] The efficiency of two test system runs for parallel implementation is presented in Fig. 6.6. The figure shows the efficiency of two test systems for the parallel run on different number of processors. The efficiency of the HP cluster runs is 0.75 in average. In fact, the efficiency is almost 0.9 in average for SUN clusters. However, the efficiency of the SUN cluster for lesser number of processors decreases compared to higher number of processors, revealing that for these particular runs the load on the processors were more than the usual. Because the SUN test system is usually a heavily loaded environment, these test results are the average cpu time consumed by each number of processors.

After first prototype tests, the second prototype model tests, involving much larger matrices, are performed on HP cluster test system. The serial version of the second prototype model simulation takes more than 5 days of run. Since it is 12 times slower to perform a test on SUN test environment and there is limited time to take test results for this Thesis, the second model is only tested on HP test environment. For the second test simulation model the scalability of parallel algorithm is presented in Fig. 6.7. According to the test results the parallel algorithm seems to be scaled worse than the small matrix scale results for HP cluster. Since HP cpus have 2MB caches, larger matrices don't fit to the cache. In fact, with larger matrices there are more cache misses than small matrices cases. This can also be seen in speed-up and efficency graphics. However, there might be another issue that the load on the processors for sequential run and the

runs with lesser number of processors were more than the load of a system with one user task. The runs with less than 10 CPUs take more than 3 days for this test model, and in such a period of time overheads of an operating system and other system based tasks (back-ups,file merging scripts) can take more time than couple of hours. Therefore, one needs to perform more test runs to circumvent the system load effects for both SUN test environment and for HP test environment. Although the HP cluster machines are not heavily loaded systems, for long runs the operating system load is expected to affect the runtime performance of the code.

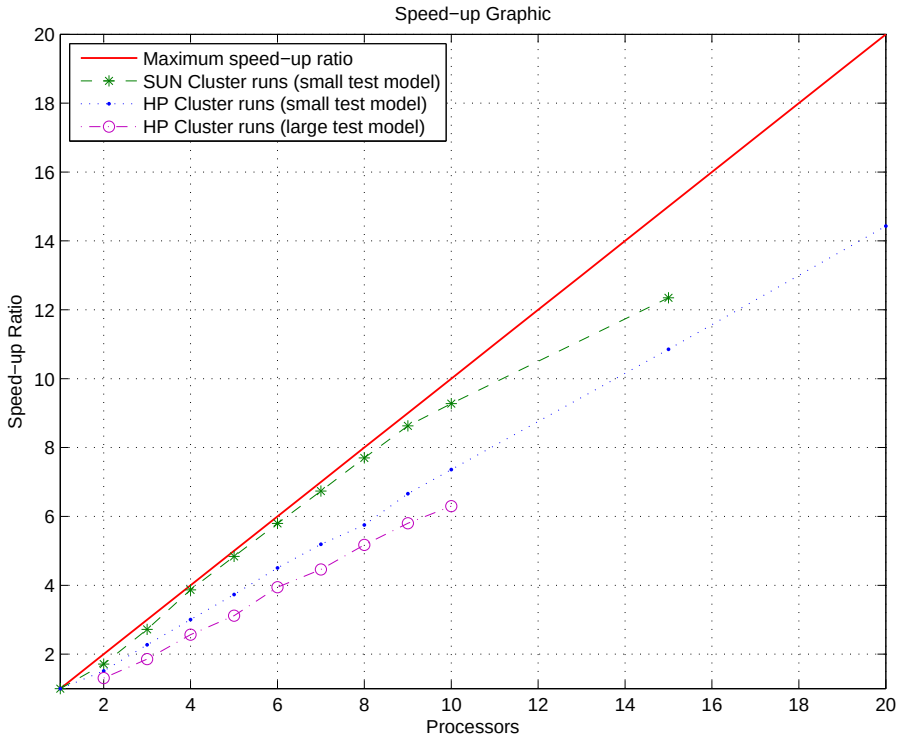

Figure 6.6: The speedup graphic showing the speedups of the test machines or environments

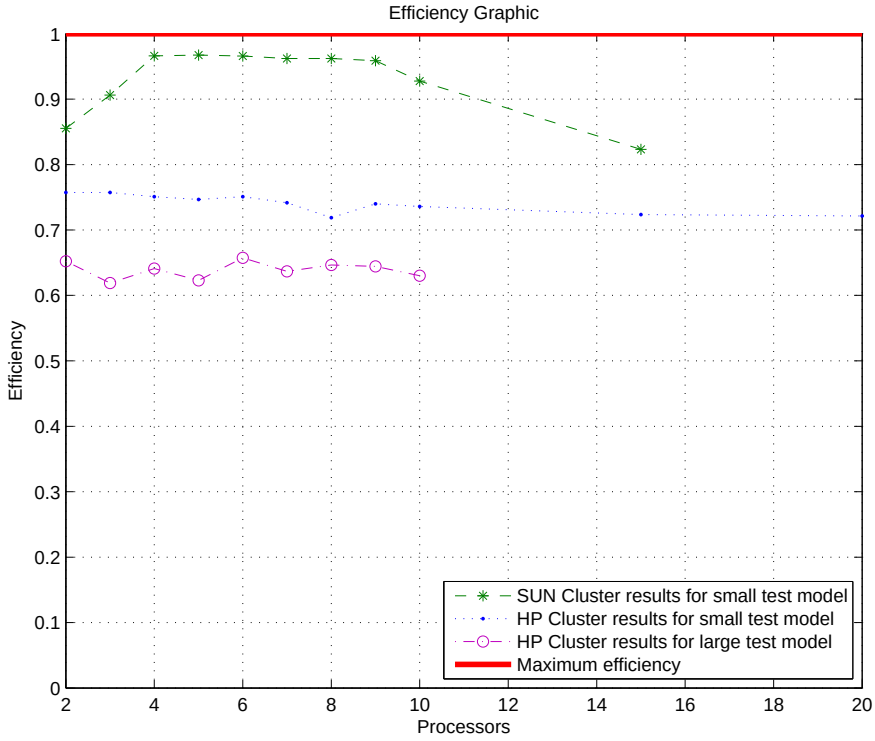

Figure 6.7: The efficiency graphic showing the speedups of test machines or environments

CHAPTER 7

CONCLUSIONS

I have developed an MPI based parallel algorithm to implement the RSGF technique calculating the vibrational density of states of a solid. The (100) flat surface of Cu is chosen as the prototype system to test the parallel implementation. The interactions between the atoms within the model system are defined through the potentials derived from the Embedded Atom Method. Since in RSGF calculation of LDOS for each frequency is sequential and independent of one another, I follow an embarrassingly parallel implementation through distributing the frequencies to the processors rather than parallelizing the matrix operations involved in the technique. MPI is chosen as the cluster based parallel computation tool. The parallel version of the code is tested on the SUN and HP cluster machines and performance on both test environment is examined. The results indicate that the parallel runs on large number of processors are at least 10 times (an order of magnitude) faster than the optimized serial code. This, thus, enables the user to work with much larger matrices representing much more realistic systems with defects and reduced symmetries.

REFERENCES

[1] R. Haydock, V. Heine, and M.J. Kelly, J. Phys. C 5, 2845 (1972).

[2] S.Y. Wu, J. Cocks, and C.S. Jayanthi, Phys. Rev. B 49,7957 (1994).

[3] A. Yamamoto, "*Crystallography of Quasiperiodic Crystals*" , Acta. Cryst. **A52**, 509-560, (1996).

[4] For a review, see K. Wandelt, "*Properties and Influence of Surface Defects*", Surf. Sci. **251/252**, 387 (1991).

[5] Charles Kittel, "*Introduction to Solid State Physics*", sixth edition, (John Wiley & Sons, Inc), (1986).

[6] Neil W. Ashcroft and N. David Mermin, "*Solid State Physics*", (Saunder College), 1976.

[7] S.M. Foiles, M.I. Baskes, and M.S. Daw, "*Embedded-Atom-Method Functions for the FCC Metals Cu, Ag, Au, Ni, Pd, Pt, and their Alloys*", Phys. Rev. B , **33**, 7983 (1996); M.S. Daw, S.M. Foiles, and M.I. Baskes, "*The Embedded-Atom Theory and Applications*", Mater. Sci. Rep. **9**, 251 (1993).

[8] K.S. Dy, Shi-Yu Wu, and T. Spratlin, "*Exact Solution for the Resolvent Matrix of a Generalized Tridiagonal Hamiltonian*", Phys. Rev. B **20**, 4237 (1979).

[9] S. Y. Wu, J. A. Cocks, and C. S. Jayanthi, "*An Accelerated Inversion Algorithm using the Resolvent Matrix Method*", Comput. Phys. Commun. **71**, 125 (1992).

[10] S. Y. Wu, Z. L. Xie, and N. Potoczak, "*Feasibility Study on the Application of the Method of the Resolvent Matrix to Complex Systems*", Phys. Rev. B **48**, 14826 (1993).

[11] S. Y. Wu, J. Cocks, and C.S. Jayanthi, "*General Recursive Relation for the Calculation of the Local Green's Fucntion in the Resolvent-Matrix Approach*", Phys. Rev. B **49**, 7957 (1994).

[12] "*NUMA Non-uniform Memory Access*", http://www.lsbu.ac.uk/oracle /oracle7/server/doc/SPS73/chap3.htm ,(2006).

[13] R. Chandra, L. Dagum, D. Kohr, D. Maydan, J. McDonald, R. Menon, "*Parallel Programming in OpenMP* ", (Morgan Kaufmann Publishers) Chp. 3. "*OpenMP Organization*", http://www.openmp.org (2006).

[14] M. J. Quinn, "*Parallel Programming in C with MPI and OpenMP*" International Edition, (McGraw Hill Publish), 159-175.

[15] Bell, G. and J. Gray; "*What's Next in High Performance Computing*", Communications of the ACM, Vol. 45, No. 2, (February 2002), pp 91-95.

[16] E. Anderson, J. J. Dongarra, "*Performance of LAPACK: A Portable Library of Numerical Linear Algebra Routines*", UT, CS-92-156, (May 1992). E. Anderson, J. J. Dongarra, "*Implementation Guide for LAPACK*", UT,CS-90-101, (April 1990).

[17] "*SUN High-End Servers*", http://www.sun.com/servers/index.jps?cat=Sun%20Fire%20High-end%20Servers&tab=3 , (January 2006).

[18] "*PGI The Portland Group*", http://www.pgroup.com , (Academic Press), (2000), Page: 20-22,185-188.

[19] "*HP Cluster Platform Systems*", http://h20311.www2.hp.com/HPC/cache/276360-0-0-0-121.html , (January 2006).

[20] H. Yıldırım, MSc Thesis, İstanbul Technical University, (2003).

[21] G. Grosso and G. P. Parravicini; "*Solid State Physics*" , (2000).

[22] "*If You Find Multiple Loops*", http://www.openmp.org/presentations/sc98/mod3_loops/sld021.htm , (January 2006)

[23] B. Onat, S. Durukanoğlu, H. Dağ; "*A Parallel Implementation of Real Space Green's Function Method for Calculations of Vibrational Density of States for a Solid*", Special Issue for Journal of Parallel Computing (2006), (accepted for publishing).

BIOGRAPHY

He was born in İstanbul in 1980. He was completed his high school education at Beşiktaş Atatürk Anadolu High School. Between 1998 and 2003, he had been in Physics Engineering Program at Department of Physics at College of Science and Letters in İstanbul Technical University. Since 2003, he has been a graduate student in Computational Science and Engineering at Informatics Institute founded in the İstanbul Technical University.