A PARALLEL COMPUTING-VISUALIZATION FRAMEWORK FOR POLYCRYSTALLINE MINERALS

A Thesis
Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Master of Science in Systems Science
In
The Interdepartmental Program in Systems Science

By
Venkatasrirama PavanKumar Yerraguntla
B.E. Computer Science & Engineering, Andhra University 2002
August, 2005
Acknowledgements

I want to begin by thanking my parents for their persistent understanding and their support. I wish to express my gratitude to my parents and younger brother for providing both the means and encouragement for me to pursue the cutting edge education in US.

Also I would like to express my most sincere appreciation to Dr. Bijaya B. Karki, my advisor. He has been guiding my research since I came to LSU. Thank you for introducing to me those brilliant ideas proposed in this thesis. Your unique and profound insight into problems has helped me to explore new horizons of knowledge in Inter-Disciplinary Research area. The weekly meetings with you have been keeping me on the right track during the past two years. I can say without exaggeration that this thesis wouldn’t be possible without your support.

I would like to express my appreciation to my committee members Dr. John. M. Tyler and Dr. Donald H Kraft for their support and suggestions. Thanks are extended to Dr. Bijaya B. Karki for honing my skills in Parallel Algorithms.

Last, but not the least, I would like to gratefully acknowledge Department of Computer Science, Louisiana State University for providing the resources and need during the project.
# Table of Contents

Acknowledgements..................................................................................................................... ii

List of Tables .............................................................................................................................. v

List of Figures ............................................................................................................................ vi

Abstract .................................................................................................................................... viii

Chapter 1: Introduction ............................................................................................................... 1

Chapter 2: Input Configuration ................................................................................................... 4
  2.1 Voronoi Diagram .............................................................................................................. 5
    2.1.1 Definition ............................................................................................................... 5
    2.1.2 Voronoin Function ................................................................................................. 6
  2.2 Bounded 3D Voronoi Diagrams ....................................................................................... 7
  2.3 Generation of Irregular Polycrystalline Structure Using 3D Voronoi Diagrams ............. 9
    2.3.1 Lattice Grid Mapping ............................................................................................. 9
    2.3.2 Inside / Outside test .............................................................................................. 11
    2.3.3 Addition of Atomic Basis .................................................................................... 11
    2.3.4 Distance Test ........................................................................................................ 13

Chapter 3: Molecular Dynamics ............................................................................................... 17
  3.1 Basic Physical Model ...................................................................................................... 17
  3.2 Implementation ............................................................................................................... 18
    3.2.1 Linked-cell Lists .................................................................................................. 19
    3.2.2 Spatial Decomposition ......................................................................................... 20
    3.2.3 M P I - Message Passing Interface ....................................................................... 21
    3.2.3.1 Basic Structure of MPI Program ............................................................... 22
    3.2.3.2 Communicators and Groups ..................................................................... 22
    3.2.3.3 MPI Programming Constructs ................................................................ 23
    3.2.4 Parallelization ...................................................................................................... 25
  3.3 Performance Tests ........................................................................................................... 28

Chapter 4: Visualization of Polycrystalline Grain Structure .................................................... 32
  4.1 Visualization Model ........................................................................................................ 32
  4.2 Implementation ............................................................................................................... 33
    4.2.1 OpenGL & GLUT Library ................................................................................... 33
    4.2.2 Different Modules in the Software ...................................................................... 34
    4.2.2.1 ReadConf .................................................................................................. 34
    4.2.2.2 InitView .................................................................................................... 35
    4.2.2.3 MakeAtoms ............................................................................................... 35
    4.2.2.4 Display and Reshape ................................................................................. 35
    4.2.3 User-Interface ...................................................................................................... 36
4.3 Visualization of Grain Boundaries.................................................................................. 36

Chapter 5: Conclusion............................................................................................................... 43

References.................................................................................................................................. 44

Vita............................................................................................................................................ 47
List of Tables

Table 2.1 Represents the Lattice structure of the MgSiO3 molecule ........................................ 12
Table 2.2 Represents the Lattice structure of the MgO molecule............................................. 13
Table 2.3 Shows the Ideal minimum distance for the MgSiO3 molecule ..................................... 14
Table 2.4 Shows the Ideal minimum distance for the MgO molecule........................................ 15
Table 3.1 Specification of Linux clusters .................................................................................. 29
List of Figures

Figure 2.1 Depicts points chosen randomly on a 2D surface ..................................................... 7
Figure 2.2 Depicts the formation of mirror images of the points in 2 dimensional space........... 8
Figure 2.3 Depicts the formation of bounded 2 dimensional Voronoi regions ......................... 8
Figure 2.4 Calculation of the parameter which is difference between X coordinates .............. 9
Figure 2.5 Shaded region represents the 2D view of the cuboid at the center of the cube...... 10
Figure 2.6 Final pictures showing a particular Voronoi region filled with tilted grid............. 11
Figure 2.7 Represents the lattice structure of MgSiO3 molecule ............................................. 12
Figure 2.8 Represents the lattice structure of MgO molecule .................................................. 13
Figure 2.9: Polycrystal of Magnesium Silicate (perovskite) containing eight grains .......... 16
Figure 2.10 Eight grain structure of MgSi03 molecule. .......................................................... 16
Figure 3.1 Splitting the physical system in subsystems along X, Y, Z axes ......................... 19
Figure 3.2 Linked-cell-list construction algorithm ................................................................. 19
Figure 3.3 Partitioning of a 2-d finite difference grid amongst four processors ................. 20
Figure 3.4 Communications pattern for four processor domain decomposition .................. 21
Figure 3.5 Schematic of spatial decomposition in PCMD algorithm .................................. 26
Figure 3.6 Outline of PCMD algorithm using MPI ............................................................... 27
Figure 3.7 Total execution time and communication time of MD on SuperMike ................. 29
Figure 3.8 Total execution time and communication time of MD on SuperHelix ............... 30
Figure 3.9 Computation time per MD step for Linux Cluster and Colombia19 .................. 31
Figure 4.1 Initial equilibrium position for the atoms (side view) ......................................... 37
Figure 4.2 Initial equilibrium position for the atoms (front view) ..................................... 38
Figure 4.3 Three Dimensional view of atoms in initial equilibrium position ................... 39
Figure 4.4 Displacement of atoms after 3000 time steps (top view). ....................................... 39
Figure 4.5 Represents two grain structure at initial time .......................................................... 40
Figure 4.6 Three Dimensional view of atoms in initial equilibrium position.......................... 40
Figure 4.7 Represents two grain structures at initial time (front view). .................................... 41
Figure 4.8 Represents two grain structures at initial time (side view)....................................... 41
Figure 4.9 Three Dimensional view of two grain structures ..................................................... 42
Abstract

In this report, we have reported some preliminary results in the development of a parallel computing-visualization framework for large-scale molecular dynamics simulations of polycrystals of minerals, which are geo-physically relevant for Earth’s mantle. First, we have generated the input configurations of atoms belonging to various grains distributed in the space in a way, which resembles the polycrystalline structure of the minerals. The Input configuration is developed using Voronoi geometry. Thus generated polycrystalline system is simulated using the PolyCrystal Molecular Dynamics algorithm. Performance tests conducted using up to 256 processors and a couple of millions of atoms have shown that the computation time per MD step remains under 20 seconds. The other important part is the development of an efficient visualization system to interactively explore the massive three-dimensional and time-dependent datasets produced by MD simulations. Some results are presented for the simulation of two-grain structure. The proposed framework is expected to be useful in simulations of more realistic and complex rheological (mechanical) properties of important Earth forming mineral phases under different conditions of stresses and temperatures.
Chapter 1: Introduction

Rapid advances in scalable parallel-computing framework based on molecular dynamics (MD) method have made it possible to carry out large-scale atomistic simulations of a wide range of materials problems [1,2,3]. For example, successful benchmark test has recently been performed for a 6.44-billion-atom simulation on a multi-Teraflop machine [4]. Continuous efforts are being made towards adapting the MD approach to simulate ever more realistic and complex materials problems (in both space and time) of important nano-technological, biotechnological and planetary applications. We are particularly interested to develop an efficient MD-based computing framework for the modeling of rheological (mechanical) properties of materials. Due to its direct relevance across so many disciplines, rheological modeling represents a fundamental cutting-edge problem.

Polycrystalline phases of relatively simple systems such as metals have previously been simulated using the MD technique [5, 6]. However, to the best of our knowledge no such studies have been carried out for relatively more complex systems such as geophysically relevant oxide and silicate mineral phases. It still remains a major challenge to develop a firm basis to understanding the rheological properties of the Earth forming materials at prevailing temperature and stress conditions of deep interior that are crucial in gaining insight into the complicated dynamics of Earth’s interior implied by seismological observations and other sources. For instance, the different mechanisms (diffusion or dislocation creep) by which the mantle may deform in the solid state follow profoundly different pictures of mantle dynamics. Whether lattice preferred orientations that could be the sources of the seismic anisotropy relating to mantle flow may develop or not depend on deformation mode. Large-scale simulations performed on massively parallel supercomputers are expected to predict the key rheological
properties under the experimentally inaccessible extreme conditions. This report is developed based on the preliminary work towards development and implementation of parallel MD simulation and visualization algorithms for atomic-level studies of Earth forming polycrystalline mineral phases. The required input configurations for polycrystals are generated using Voronoi geometry. The individual grains are represented by Voronoi cells bounded in the simulation box, which are filled with atoms on specific lattice arrangements.

Our simulation algorithm, known as PCMD (Polycrystal Molecular Dynamics) employs a spatial decomposition approach and dynamic management of distributed data structures for force computations [3, 4, 5]. As usual, the algorithm involves the $O(N)$ operations to compute van der Waals interactions and the $O(N^{3/2})$ operations to compute Coulomb interactions, where $N$ is the number of atoms. Moreover, we are particularly interested in commodity-based Linux clusters constructed from off-the shelf components. For example, Louisiana State University (LSU) has recently acquired/upgraded a very powerful Linux clusters (named SuperMike) consisting of 512 dual Intel Xeon 3.0 GHz nodes (i.e., 1024 processors) connected by Myricom’s Myrinet interconnect. The SuperMike was ranked among the 11th fastest supercomputer in the world in August 2002 [7]. Although the performance of low-cost multi-Teraflop Linux clusters has thus been confirmed by standard benchmark tests, there is a continuing concern regarding the scalability of such architecture for real high-end scientific/engineering applications. This work is expected to represent a successful example of such high-end computational environment for realistic simulations. The final part is to develop the visualization software which is used to visualize the polycrystalline grain structure containing multi-million atoms comprising individual grains oriented in random/preferred orientations. The dynamics of the atoms present in the polycrystalline grains during the simulation is traced using the visualization software.
The atomic positional data generated from the simulation are fed as input to the Visualization software at regular time intervals to animate the dynamics of atoms and grains. We discuss about various problems encountered like voluminous data, which affect the efficiency of the software and increase the demand for more hardware requirements. Further the methods that are employed to improve the efficiency of the software and at the same time reduce the hardware requirements are described in detail. This thesis report is organized as follows. Chapter 2 describes the generation of input configuration to represent a polycrystalline phase. Chapter 3 describes the design and analysis of the MD algorithm. Results of performance tests are given here. Chapter 4 explains about the visualization of massive three-dimensional and time-dependent simulation datasets. Finally, Chapter 5 contains conclusions.
Chapter 2: Input Configuration

Basically to perform the atomistic simulations of polycrystals of Earth forming mineral phases we need to generate a polycrystalline structure containing molecular configuration. Generally in the polycrystalline structure the region is divided into sub regions called Grains. Each individual Grain is filled with atoms that belong to particular molecule. Also the grains are formed in such a way that any molecule that falls in a particular grain is always closest to that particular grain than any other grain surrounding that molecule. Though the molecules that belong to one particular grain are oriented in same direction, the molecules that belong to different grains are oriented in different directions.

From the above facts it is evident that the input configuration needed for the simulation should contain atoms belonging to particular mineral distributed in the space in such a way that it resembles the polycrystalline structure of the minerals. To achieve the above mentioned, the input configuration is developed using Voronoi geometry which emulates the grain Structure of the polycrystalline minerals which are present in the earth’s mantle. Basically the Voronoi regions and the grains in the polycrystalline structure share the same properties and hence Voronoi geometry is most suitable for the present input configuration generation.

Each Voronoi region needs to be filled with atoms based on a particular molecular lattice structure so as to emulate a grain structure of a polycrystalline mineral. The orientation of the molecules in each Voronoi region is chosen randomly but made sure that all molecules with in the same grain have same orientation. In this manner each Voronoi region is filled with atoms so as to develop a full fledged polycrystalline grain structure containing atoms of a particular mineral which is ready for real time simulation. Each Voronoi region contains atoms oriented in different directions so as resemble grain structure of the polycrystal minerals.
2.1 Voronoi Diagram

The Voronoi diagram is a fundamental structure in the computational geometry and arises naturally in many different fields. The Voronoi Diagram is closely linked to another important geometric structure, the so called Delaunay Triangulation. This following section gives brief description of the Voronoi Diagram and how they are generated.

2.1.1 Definition

A Voronoi diagram of a set of "sites" (points) is a collection of regions that divide up the plane. Basically a Voronoi diagram is defined as follows: Let \( P = \{p_1, p_2, \ldots, p_n\} \) be a set of \( n \) distinct points in the plane; these points are the sites. We define the Voronoi Diagram of \( P \) as the subdivision of the plane into \( n \) cells, one for each site in \( P \), with the property that a point \( q \) lies in the cell corresponding to a site ‘\( p_i \)’ if \( \text{dist} (q, p_i) < \text{dist} (q, p_j) \) for each \( p_j \in P \) with \( j \neq i \).

Geometrically speaking the boundary of the Voronoi diagram is formed by the plane that forms the perpendicular bisector between the two adjacent points representing two sites. We denote the vertices of a Voronoi Diagram for site ‘\( p_i \)’ by \( V (p_i) \).

An ordinary Voronoi diagram is formed by a set of points in the plane called the generators or generating points. Each region corresponds to one of the sites, and all the points in one region are closer to the corresponding site than to any other site. A Voronoi region in a 3-Dimensional field is enclosed or represented by convex polyhedron. The faces or facets of convex polyhedron are formed by the intersection of the perpendicular bisectors of the points in the associated set. Because of the basic principle of generation of Voronoi diagram, some of Voronoi regions are infinite and some of Voronoi regions are finite. Currently, there are different software modules which implement several algorithms to generate Voronoi diagram. In our case, the generation of the Voronoi diagram is done using MATLAB Software. MATLAB also called
Mathematics Laboratory is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. MATLAB provides an in-built function named Voronoin in the Interpolation and Computational Geometry section. This Voronoin function is capable of generating Voronoi diagram in N-Dimensional space where n greater than equal to 2.

2.1.2 Voronoin Function

\[
[V,C] = \text{Voronoin}(X)
\]

returns Voronoi vertices \( V \) and the Voronoi cells \( C \) of the Voronoi diagram of \( X \). \( V \) is a \((\text{numv-by-n})\) array of the \text{numv} (number of generator points) Voronoi vertices in \( n \)-dimensional space, each row corresponds to a Voronoi vertex. \( C \) is a vector cell array where each element contains the indices into \( V \) of the vertices of the corresponding Voronoi cell. \( X \) is an \( m \)-by-\( n \) array, representing \( m \) \( n \)-dimensional points, where \( n \) greater than 1 and \( m \) greater than or equal to \( n+1 \). The first row of \( V \) is a point at infinity. If any index in a cell of the cell array is 1, then the corresponding Voronoi cell contains the first point in \( V \), a point at infinity. This means the Voronoi cell is unbounded.

Basically Voronoin uses Qhull algorithm. Qhull computes convex hulls, Delaunay triangulations, and half space intersections about a point, Voronoi diagrams, furthest-site Delaunay triangulations, and furthest-site Voronoi diagrams. It runs in 2-d, 3-d, 4-d, and higher dimensions. It implements the Quickhull algorithm for computing the convex hull. Qhull handles round off errors from floating point arithmetic. It computes volumes, surface areas, and approximations to the convex hull. Since the sites which are on the boundary of the Voronoi diagram are semi-bounded that is semi-infinite. To obtain a bounded cube containing Voronoi regions which is supposed to be the simulation box, we make the mirror images of the random
points (generated initially) along positive and negative directions of X, Y, Z axes so that the boundaries of the bounded cube from the edges of Voronoi regions.

2.2 Bounded 3D Voronoi Diagrams

The Voronoi diagrams are semi-bounded in nature that is in reference to those regions that are formed around the points which are towards the corners. But the present application demands a simulation box divided into grains. That is, we need a bounded cubic region which is divided into Voronoi regions. In order to generate bounded Voronoi regions we have used Mirror Imaging technique to develop bounded Voronoi regions and the process is described as followed.

Step1: Initially Random points are generated in the 3D Plane using the random number generator where ‘m’ represents the total number of random points in the plane.

Step2: The mirror images of these points are taken along the maximum and minimum bounds of the simulation box for each of X, Y, Z axes.

Step3: This set containing random points is sent as the input to the Voronoin(X) function which returns two sets of arrays. One set of array represents the Voronoi points on the plane and other array represents the vertices of each individual Voronoi polyhedron formed by these points.

Step4: Now we retain the Voronoi polyhedrons for those points only which lie inside the simulation box.
Step 5: Discard each Voronoi polyhedron which fall outside the bounds of the cube so as to obtain a bounded cube containing the region divided into Voronoi polyhedra.

Figure 2.2 Depicts the formation of mirror images of the points in 2 dimensional space

Figure 2.3 Depicts the formation of bounded 2 dimensional Voronoi regions[29]
2.3 Generation of Irregular Polycrystalline Structure Using 3D Voronoi Diagrams

Basically for generating the irregular polycrystalline structure, each of the Voronoi regions should be filled with Grid points with random orientation. This is achieved by circumscribing each Voronoi polyhedron tilted 3D grid whose tilt angle is selected randomly for each of the Voronoi region so that each of the Voronoi region is filled with tilted 3D grid whose tilt angle is different from the that of neighboring regions. Now the Grid points are used as the Basis for placing the molecules and hence with in a polyhedron all the molecules are oriented in one direction. A test is performed which is based on the Inter-Molecular distances to make that the whole system stable and does not collapse during the process of simulation.

2.3.1 Lattice Grid Mapping

The following process is followed to map a tilted grid on each Voronoi region.

Step1: Consider a particular Voronoi region which is to be filled by the tilted 3D grid. The difference of the X-co-ordinate values between the points which have maximum X-co-ordinate value and minimum X-co-ordinate value is found (parameter1). Similarly difference of values of the points is found for Y-co-ordinate values (parameter2), Z values (parameter3).

![Figure 2.4 Calculation of the parameter1 which is difference between X coordinates](image.png)
Step2: Among the three parameters which ever parameter is the maximum is considered and a Cube having side twice the value of that parameter is formed with center at the Origin.

Step3: A regular 3D grid is built with in this cube where two neighboring points differ by distance equal to scale. Scale is defined as the value obtained by dividing the side of the simulation box with number of molecules along each dimension of the simulation box.

Step4: The cube is rotated along a particular axis with a random angle (generated from random function). At the center of this cube that is at the origin, a cuboid whose dimensions are equal to (parameter1, parameter2, parameter3) is formed.

Figure 2.5 Shaded region represents 2D view of the cuboid placed at the center of the cube

Step5: Then the grid points of the cube that fall with in this cuboid are retained and remaining points are discarded.

Step6: The vertices of cuboid are translated by appropriate distance such that cuboid circumscribe Voronoi region a lies on inside or outside a Voronoi region following test is performed.
Step 7: An Inside/Outside test (described as follows) is performed for each grid point and the grid points that fall within the region are retained and remaining points that fall outside are discarded.

2.3.2 Inside / Outside Test

To test whether a particular point is inside a convex region or not and is described as followed. The normal (say Normal) to each facet of the polyhedron enclosing the Voronoi region is calculated. The vector between the point (center point) that is strictly inside the Voronoi region and a point which is on the facet is considered (say Invec). The vector joining the test point and point on the facet is calculated (say Testvec). Now dot product of the Normal and Invec is calculated (say result1). The dot product of the Normal and Testvec is calculated (say result2). If the algebraic signs of result1 and result2 are equal, the two points are supposed to be on the same side of the facet. Otherwise they are on the opposite side of the facet. Since one point is always inside the Voronoi region, the location of other points can be easily determined. This is how the Voronoi region is filled with 3D grid that is tilted with a random angle is obtained.

2.3.3 Addition of Atomic Basis

Now the molecules are placed at the grid points which are taken as basis. In the case of Magnesium Silicate (MgSiO3), molecule is placed as explained. The Magnesium (Mg) atom is placed at each Grid point and the corresponding Silicon (Si), Oxygen (O) atoms are placed at
appropriate distances from their corresponding Mg atom which are defined as follows. If Mg atom is placed at x1, y1, z1 then the atoms are placed as follows:

**Table 2.1 Represents the Lattice structure of the MgSiO3 molecule**

<table>
<thead>
<tr>
<th>Atom</th>
<th>X-coordinate</th>
<th>Y-coordinate</th>
<th>Z-coordinate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mg</td>
<td>x1</td>
<td>y1</td>
<td>Z1</td>
</tr>
<tr>
<td>Si</td>
<td>x1 + scale/2</td>
<td>y1 + scale/2</td>
<td>z1 + scale/2</td>
</tr>
<tr>
<td>O</td>
<td>x1 + scale/2</td>
<td>y1 + scale/2</td>
<td>z1</td>
</tr>
<tr>
<td>O</td>
<td>x1 + scale/2</td>
<td>y1</td>
<td>z1 + scale/2</td>
</tr>
<tr>
<td>O</td>
<td>x1</td>
<td>y1 + scale/2</td>
<td>z1 + scale/2</td>
</tr>
</tbody>
</table>

**Figure 2.7 Represents the lattice structure of MgSiO3 molecule**

In the above figure 2.7 magnesium is represented by red sphere, oxygen by blue and silicon is represented by green color. In the case of Magnesium Oxide (MgO), molecule is placed as explained. The Mg atom is placed at each Grid point and the corresponding Si, O atoms are placed at appropriate distances from their corresponding Mg atom which are defined as follows. The figure 2.8 is used to represent the structure of Magnesium molecule lattice which contains only two atoms namely magnesium and oxygen. In figure 2.8, magnesium is represented by
green sphere (light green for atom at basis point), oxygen is represented by dark blue. If Mg atom is placed at $x_1, y_1, z_1$ then the atoms are placed as shown in Table 2.2.

Table 2.2 Represents the Lattice structure of the MgO molecule

<table>
<thead>
<tr>
<th>Atom</th>
<th>X-coordinate</th>
<th>Y-coordinate</th>
<th>Z-coordinate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mg</td>
<td>$x_1$</td>
<td>$y_1$</td>
<td>$z_1$</td>
</tr>
<tr>
<td>Mg</td>
<td>$x_1 + \text{scale/2}$</td>
<td>$y_1 + \text{scale/2}$</td>
<td>$z_1$</td>
</tr>
<tr>
<td>Mg</td>
<td>$x_1 + \text{scale/2}$</td>
<td>$y_1$</td>
<td>$z_1 + \text{scale/2}$</td>
</tr>
<tr>
<td>Mg</td>
<td>$x_1$</td>
<td>$y_1 + \text{scale/2}$</td>
<td>$z_1 + \text{scale/2}$</td>
</tr>
<tr>
<td>O</td>
<td>$x_1 + \text{scale/2}$</td>
<td>$y_1 + \text{scale/2}$</td>
<td>$z_1 + \text{scale/2}$</td>
</tr>
<tr>
<td>O</td>
<td>$x_1$</td>
<td>$y_1$</td>
<td>$z_1 + \text{scale/2}$</td>
</tr>
<tr>
<td>O</td>
<td>$x_1$</td>
<td>$y_1 + \text{scale/2}$</td>
<td>$z_1$</td>
</tr>
<tr>
<td>O</td>
<td>$x_1 + \text{scale/2}$</td>
<td>$y_1$</td>
<td>$z_1$</td>
</tr>
</tbody>
</table>

Figure 2.8 Represents the lattice structure of MgO molecule

2.3.4 Distance Test

In general if any two atoms come close than a particular cut-off distance between the two atoms then the two atoms may collapse into each other. This happens due to imbalance that
arises in force factors between the two atoms. This would make the system unstable and would lead to the crash of the whole system. Normally the atoms that belong to molecules from different regions are in different orientations. Thus there is good probability that atoms of the molecules that are present at the boundaries of the neighboring regions may come closer than the expected ideal minimum distance. So in order to avoid this situation, a preventive measure is taken which is based on the distance between any two atoms. That is, for a system to be wholly stable there are specific minimum values for distances between any two atoms, beyond which the atoms are not expected to come closer. The Distance test is described as in the following part. The distances between the individual atoms of the neighboring molecules is measured and compared with the expected ideal minimum distance between the corresponding atoms. If the distance is less than expected ideal minimum distance between the atoms then one of the molecules is discarded irrespective of the type or number of atoms within a molecule violating the condition. At the end of the test the system is survived with only those molecules whose atoms satisfy the ideal distance condition. This is how it is made sure that the whole system is made stable and ready for the real time simulation.

Table 2.3 Shows the Ideal minimum distance for the MgSiO3 molecule

<table>
<thead>
<tr>
<th>Atomic combination</th>
<th>Ideal minimum Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mg—Mg, Si—Si</td>
<td>Scale</td>
</tr>
<tr>
<td>Mg—Si</td>
<td>1.732 * scale</td>
</tr>
<tr>
<td>Mg—O</td>
<td>0.707 * scale</td>
</tr>
<tr>
<td>Si—O</td>
<td>0.5 * scale</td>
</tr>
<tr>
<td>O--O (diff point-diff mol)</td>
<td>scale</td>
</tr>
<tr>
<td>O--O (diff point-diff mol)</td>
<td>scale * 0.5</td>
</tr>
</tbody>
</table>
Table 2.4 Shows the Ideal minimum distance for the MgO molecule

<table>
<thead>
<tr>
<th>Atomic combination</th>
<th>Ideal minimum Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mg—Mg (same point-diff mol)</td>
<td>scale * 0.5 * 1.414</td>
</tr>
<tr>
<td>Mg—Mg (diff point-diff mol)</td>
<td>scale * 0.5 * 1.414</td>
</tr>
<tr>
<td>Mg—O</td>
<td>0.5 * scale</td>
</tr>
<tr>
<td>O—O (same point-diff mol)</td>
<td>scale * 0.5 * 1.414</td>
</tr>
<tr>
<td>O--O (diff point-diff mol)</td>
<td>scale * 0.5 * 1.414</td>
</tr>
</tbody>
</table>

In the tables shown the term scale is defined as the distance between principal atoms belonging two different adjacent molecules and is used as a parameter to fix the minimum ideal distance between the molecules. The following parameters are used in case of MgSiO3 molecule. Now at this stage we have simulation box which is being tessellated into Voronoi regions. Also, each Voronoi region is filled with Magnesium Silicate or Magnesium Oxide molecule oriented in particular direction so that all the molecules with in the same cell are oriented in same direction but different from that of neighboring Voronoi regions. Instead of Magnesium Silicate or Magnesium Oxide molecules, any other molecule can also be placed at the Grid point according to the lattice structure of that molecule and hence another molecular system can be developed using the same software. A polycrystal of Magnesium Silicate (perovskite) containing eight grains that were generated with Voronoi method is shown in the figure 2.9.
Figure 2.9: Polycrystal of Magnesium Silicate (perovskite) containing eight grains

Figure 2.10 Eight grain structure of MgSi03 molecule

The figure displays eight grain structure containing $25 \times 25 \times 25$ molecule that is 78125 atoms. This above discussed procedure is implemented using C software language and MATLAB C and C++ math library. In next section we discuss how the molecular dynamics algorithm is implemented.
Chapter 3: Molecular Dynamics

In this section we discuss the theory behind the Molecular dynamics and also about the methods that are used to implement them successfully. The algorithm that is used and its related data structures used are defined and explained in detail.

3.1 Basic Physical Model

The Molecular Dynamics method is described for a physical system of set of $N$ atoms [9]. The positions $\{r_i\} \forall i=1,...,N$, and velocities, $\{v_i\} \forall i=1,...,N$, are calculated at specific time intervals by numerically integrating the Newton’s equations of motion,

$$m_i \frac{d^2 r_i}{dt^2} = F_i^e,$$  \hspace{5pt} (3.1)

$$\vec{F}_i = -\frac{\partial V(\{r_i\})}{\partial r_i}.$$  \hspace{5pt} (3.2)

where $m_i$ is the mass of atom $i$ and the force on atom $i$ as $F_i$.

The above Newton’s equations of motion in Eq.3.2 are integrated using the velocity Verlet algorithm [10]:

$$r_i(t+\Delta t) = r_i(t) + v_i(t) \Delta t + 1/2 \left( F_i(t)/m_i \right) \Delta t^2 + O(\Delta t^3) \hspace{5pt} \left( i = 1, \ldots, N \right)$$  \hspace{5pt} (3.6)

$$v_i(t+\Delta t) = v_i(t) + 1/2 \left( (F_i(t+\Delta t) + F_i(t))/m_i \right) \Delta t^2 + O(\Delta t^3) \hspace{5pt} \left( i = 1, \ldots, N \right)$$  \hspace{5pt} (3.7)

The above described equations calculate the displacement and velocity of the atoms at regular time intervals. The accuracy of the calculations directly depends on the magnitude of time steps used for calculation. To compute the long-range Coulomb interactions, Ewald sum method is used which allows a replacement of the infinite Coulomb summation by two rapidly convergent
summations, one in real space which is short-range in nature and the other in reciprocal space which is long-range in nature. Thus the electrostatic potential energy of a system of N charges, with self-energy correction term, can be expressed as:

\[
V_{\text{col}} = \frac{1}{4\pi\varepsilon_0} \sum_{i<j}^N \frac{q_i q_j}{r_{ij}} \operatorname{erfc}(4r_{ij}/\gamma) + \frac{1}{\varepsilon_0} \sum_{k \neq 0} \frac{1}{k^2} \sum_i^N q_i \exp(-ikr_i) \right|^2 - \frac{1}{\pi\varepsilon_0} \sum_{i=1}^N q_i^2,
\]

(3.5)

where \(r_{ij}\) is the distance vector between \(i^{th}\) and \(j^{th}\) particles with charges \(q_i\) and \(q_j\), respectively and \(k\) is the reciprocal lattice vector. In practice, the convergence of the Ewald sum is controlled by three variables: the real space cutoff \(r_c\); the convergence parameter \(\gamma\) and the largest reciprocal space vector \(k_{\text{max}}\) used in the reciprocal sum. The van der Waals and Columbic contributions to atomic interactions are given by

\[
V_{\text{vdw}} = \sum_{i<j} V_{\text{vdw}}(r_{ij}) = \sum_{i<j} [A_{ij} \exp(-B_{ij}r_{ij}) - C_{ij}r_{ij}^6],
\]

(3.3)

\[
V_{\text{col}} = \sum_{i<j} q_i q_j \varepsilon r_{ij},
\]

(3.4)

where \(q_i\) is the partial atomic charge of atom \(i\), \(r_{ij} = |\vec{r}_{ij}|\), \(\vec{r}_{ij} = \vec{r}_i - \vec{r}_j\), and \(\varepsilon\) is the dielectric constant. A cutoff distance \(r_c\) is introduced for van der Waals contribution, beyond which the potential and force are set to zero.

3.2 Implementation

This section describes each and every parallel algorithm and their corresponding supporting features that have been used in the implementation of parallel molecular dynamics algorithm for polycrystalline minerals. The pseudo code for each algorithm is also shown which is followed by explanation which is given about the way the algorithms work. Also the parallelization scheme involved is explained in lucid manner.
3.2.1 Linked-cell Lists

The computation time for calculating van der Waals and real-space Ewald contribution is done in O(n) time using a data structure namely, linked-cell lists. Basically the physical system is divided into $M_x \times M_y \times M_z$ cells of cell edge $> r_c$ and on average $N_c = N/(M_x M_y M_z)$ atoms in each cell to construct the linked-cell-list [5]. An atom in a particular cell $m$ has to search only the 26 neighboring cell in addition to cell $m$ to find all the atoms within the distance $r_c$ as the cell edge is at least $r_c$. This computation scales as $27NN_c$, compared with the $O(N^2)$ brute force search. As shown in Figure 3.2., the linked-cell-list construction algorithm uses two data structures, $lscl$ and $lshd$, where $lscl(m)$ is the linked list of atoms in cell $m$ and $lshd(m)$ points to the first element of $lscl(m)$.

![Figure 3.1 Splitting the physical system in subsystems along X, Y, Z axes]({})

| Input: | number of atoms $N$, positions $\{ \hat{r}_i \}$, cutoff radius $r_c$, number of cells $M$. |
| Output: | linked list $lscl$, header list $lshd$ |
| MAKE_LINKED_CELL($N$, $\{ \hat{r}_i \}$, $r_c$, $M$) |
| $\forall m$, $lshd(m) = 0$ |
| for $j \leftarrow 1$, $N$ |
| $m_{i\alpha} = \lfloor (r_{i\alpha} + r_c)/r_c \rfloor$ ( $\alpha = x, y, z$; $r_{i\alpha}$ is the $\alpha$th component of $\hat{r}_i$) |
| $m = m_x \times M_y \times M_z + m_y \times M_z + m_z + 1$ |
| $lscl(j) \leftarrow lshd(m)$ |
| $lshd(m) \leftarrow j$ |

![Figure 3.2 Linked-cell-list construction algorithm]({})
3.2.2 Spatial Decomposition

Sometimes the geometry of a problem clearly prescribes decomposition. This is particularly true of cases where spatial systems are being modeled. Take a finite difference scheme for example. In these problems large grids are commonly encountered. For the sake of simplicity let us say that we wish to model a two dimensional system. The idea then is to distribute parts of the grid amongst the processors. The work entailed across the system will be fairly evenly distributed so giving each processor an equal chunk of grid should prove to be adequate to load balance the system. Suppose we start off with four processors as in Figure below.

![Partitioning of a 2-d finite difference grid amongst four processors](image)

Figure 3.3 Partitioning of a 2-d finite difference grid amongst four processors\(^{28}\)

The easiest way to perform this decomposition (other than to have adopted a data parallel paradigm for which this problem would be ideal) is to allocate each processor a subsection of the system. All the data is still stored at each processor but each processor is only responsible for updating the data in its own sector. At the end of each update processors retrieve the updated data from the other processors and are then ready to move on to the next update phase. The problem with this decomposition is that the problem size is constrained by the processor memory the largest problem you will ever be able to model is one that will fit in the memory of a single processor. However extending the decomposition a little bit further so that processors only look after the data that it has been assigned then the maximal problem size is dependent on the total memory distributed across all processors. There are complications. The algorithm must now
become a little more complex to deal with the data interdependency between the processors. For the simplest finite difference scheme, where you only need to access data points above, below to the right and left of each point, this means that at the processor boundaries data located in some other processor will be required. Before we begin the update phase then we copy this extra data into our own processor as in Figure below which also has periodic boundary conditions. The processors can then proceed with the computational phase by purely examining local data in each processor. We thus preserve the data locality of the system.

![Communications pattern for four processor domain decomposition](image)

Figure 3.4 Communications pattern for four processor domain decomposition

For a little extra work then we can now potentially tackle problems much larger in size than would have been possible by storing the entire data set at each processor. The communications pattern required is straightforward and should not be difficult to implement.

**3.2.3 MPI - Message Passing Interface**

MPI is a specification for the developers and users of message passing libraries. By itself, it is not a library but rather the specification of what such a library should be. Simply stated, the goal of the Message Passing Interface is to provide a widely used standard for writing message
passing programs. The interface attempts to be practical, portable, efficient and flexible. MPI lends itself to most (if not all) distributed memory parallel programming models. In addition, MPI is commonly used to implement (behind the scenes) some shared memory models, such as Data Parallel, on distributed memory architectures. MPI can be used on various Hardware platforms like Distributed Memory models, Shared Memory models and Hybrid memory models belonging to any parallel architecture including massively parallel machines.

3.2.3.1 Basic Structure of MPI Program

Basically any MPI program consists of four major steps which are outlined as below in the same specific order:

Step1: Initially all the header files which are needed to develop an MPI program relevant to the current environment are included.

Step2: The MPI environment is initialized and all the variables which are required are initialized. The first MPI routine called in any MPI program must be the initialization routine. This routine establishes the MPI environment, returning an error code if there is a problem.

Step3: Now the message passing is done between the processors belonging to a particular group by calling appropriate MPI routines in a specified sequence and specific mode.

Step4: Finally after completion of the whole message passing process, the current MPI environment is terminated by calling the specific finalizing routines of MPI. These routines clean up all MPI data structures, cancels operations that never completed, etc.

3.2.3.2 Communicators and Groups

MPI uses objects called communicators and groups to define which collection of processes may communicate with each other. Most MPI routines require you to specify a communicator as an argument. A group is an ordered set of processes. Each process in a group is
associated with a unique integer rank. Rank values start at zero and go to N-1, where N is the number of processes in the group. In MPI, a group is represented within system memory as an object. It is accessible to the programmer only by a "handle". A group is always associated with a communicator object. A communicator is a handle representing a group of processors that can communicate with one another. The communicator name is required as an argument to all point-to-point and collective operations. The communicator specified in the send and receive calls must agree for communication to take place. Processors can communicate only if they share a communicator.

A communicator encompasses a group of processes that may communicate with each other. All MPI messages must specify a communicator. In the simplest sense, the communicator is an extra "tag" that must be included with MPI calls. Like groups, communicators are represented within system memory as objects and are accessible to the programmer only by "handles". For example, the handle for the communicator that comprises all processes is MPI_COMM_WORLD. There can be many communicators, and a given processor can be a member of a number of different communicators. Within each communicator, processors are numbered consecutively (starting at 0). This identifying number is known as the rank of the processor in that communicator. A processor can also determine the size, or number of processors, of any communicator to which it belongs.

3.2.3.3 MPI Programming Constructs

MPI programming also involves certain constructs which are devised according to specific rules. MPI programming has its own Data Types and Routines which are to be used for efficient programming. MPI provides its own reference data types corresponding to the various elementary data type in other programming languages. MPI allows automatic translation between
representations in a heterogeneous environment. As a general rule, the MPI data type at a given receiving end must match the MPI data type specified at the corresponding sending end. In addition, MPI allows you to define arbitrary data types built from the basic types.

MPI provides routines for sending and receiving blocking and nonblocking messages. A blocking send does not return until it is safe for the application to alter the message buffer on the sending process without corrupting or changing the message sent. A nonblocking send may return while the message buffer on the sending process is still volatile, and it should not be changed until it is guaranteed that this will not corrupt the message. This may be done by either calling a routine that blocks until the message buffer may be safely reused, or by calling a routine that performs a nonblocking check on the message status. A blocking receive suspends execution on the receiving process until the incoming message has been placed in the specified application buffer. A nonblocking receive may return before the message has been received into the specified application buffer, and a subsequent call must be made to ensure that this has occurred before the application uses the data in the message.

There are large number of MPI routines which can be used for implementing parallel algorithms programmatically. But practically almost any parallel algorithm can be implemented using the following few commands.

MPI_Init: The first MPI call must be MPI_INIT, which initializes the message passing routines.

MPI_Finalize: When you are finished with the message passing routines, you must close out the MPI routines. The MPI_Finalize routine is used for doing this finalization.

MPI_Comm_Size: This routine is used to find the number of processes in a specified MPI communicator. In MPI, you can divide your total number of processes into groups, called
communicators. Generally used within the communicator MPI_COMM_WORLD to determine the number of processes being used by your application.

MPI_COMM_RANK: This routine is used to find the rank of the processes running the code. Each process in a communicator is assigned an identifying number starting from 0.

MPI_Send: Basic blocking Send operation. Routine returns only after the application buffer in the sending process is free for reuse. Note that this routine may be implemented differently on different systems. The MPI standard permits the use of a system buffer but does not require it. Some implementations may actually use a synchronous Send to implement the basic blocking send.

MPI_Recv: Receive a message and block until the requested data is available in the application buffer in the receiving process.

MPI_Bcast: Broadcasts (sends) a message from the process with rank "root" to all other processes in the group.

MPI_Allreduce: Applies a reduction operation on all processes in the group and then places the result in all processes in the group. This is equivalent to an MPI_Reduce followed by an MPI_Bcast.

3.2.4 Parallelization

The PCMD algorithm has been parallelized using spatial decomposition [5, 10], in which the physical system is divided into \( P \) subsystems, \( P_x \times P_y \times P_z \), in the x, y, and z directions, and each subsystem is assigned to a processor. Each processor \( p \) thus stores arrays containing the positions \( \mathbf{r}_i(i = 1, \ldots, N_p) \), velocities \( \mathbf{v}_i(i = 1, \ldots, N_p) \), species \( i \) \( (i = 1, \ldots, N_p) \) of atoms and the identification number of the Grain \( g_i \), where \( N_p \) is the number of atoms residing in the \( p \)th subsystem. When time-stepping procedure updates the atomic positions, some atom \( i \) in
subsystem \( p \) may have moved out to a neighboring subsystem \( p' \). This atom is ‘migrated’ to processor \( p' \), i.e., the information, \( \dot{r}_i, \dot{v}_i \) and \( i \), is removed from processor \( p \) and sent to processor \( p' \), where it is appended to position, velocity and species arrays (see Figure 3.8). To calculate bonded and van der Waals interactions as well as the near-field contribution to Columbic interaction in processor \( p \), all atoms, which are in the neighboring processors but are close to \( p \), are ‘cached’. Namely the positions and species of these atoms must be received from the neighboring processors, and then appended to local arrays in \( p \). More precisely, we define a skin of thickness \( r_c \) (the cutoff radius of the van der Waals potential function, \( v_{vdw}(r) \)) around each processor \( p \), and cache the information about the atoms within this skin from neighboring processors to \( p \).

![Figure 3.5 Schematic of spatial decomposition in PCMD algorithm](image)

In the figure 3.5 the atoms that fall between red borders that is, atoms at the boundary of the processor are given special attention while performing the force and energy calculations between atoms. For example atom \( i \) has migrated from processor 3 to processor 4 due to time-stepping procedure. For bonded and non-bonded pair interactions, the force on each atom is
computed by the processor that has the atom. The same strategy is used to calculate near-field contribution to Coulomb interaction and for any bonded pair \( ij \) crossing a processor boundary (see figure 3.5).

```plaintext
--Header file containing all variable declarations
#include 'pcmd.h'

--Header file for MPI environment
#include 'mpif.h'

--Initialize the MPI environment
call MPI_INIT(ierr)

--Keep my node ID in a common block
call MPI_COMM_RANK(MPI_COMM_WORLD,myid,ierr)

--Read atomic coordinates, velocities and box dimensions from input
filename='inp000'

==========Main MDloop==============================

cpu1=MPI_WTIME()

Calculate Initial energies and tilt structure
call e_kinetic(ekin)
call MPI_ALLREDUCE(vepot,epot,1,..., MPI_SUM, MPI_COMM_WORLD,ierr)
call MPI_ALLREDUCE(energy,denergy ,.. , MPI_SUM,MPI_COMM_WORLD,ierr)

-- Start Outer loop from here -----------------------------
do iouter = 1, nstp

--First half-step long-range kick, v(t+mmts*Dt/2)
call vkick(mmts)

c--Migrate moved-out atoms to neighbor nodes
call bamove(comt)

c--Second half-step long-range kick, v(t+Dt)
call vkick(mmts)

c—Calculate Energies and temperature at every iouter step---------
call e_kinetic(ekin)
call MPI_ALLREDUCE(vepot,epot,.., MPI_SUM,MPI_COMM_WORLD,ierr)
etot=ekin+epot
temp=ekin*temp_fac

c—Calculate Energy at every nmes'th step------------------
call MPI_ALLREDUCE(energy,denergy, MPI_SUM,MPI_COMM_WORLD,ierr)
c--End of outer loop---------------------------------
c—Write output files
cpu2=MPI_WTIME()

c==Main MDloop ends here=================================

c--Write the CPU time from node 0
cpu=cpu2-cpu1

c--Finalize the MPI environment
call MPI_FINALIZE(ierr)

Figure 3.6 Outline of PCMD algorithm using MPI
```
We use single-program multiple-data (SPMD) programming paradigm, in which all the processors execute the same program on different datasets. Figure 3.6 shows the parallelized PCMD algorithm for each processor. With the spatial decomposition, the computation scales as $N/P$ while communication scales as $(N/P)^{2/3}$. This is how the time complexity of the whole molecular dynamics algorithm is reduced to the minimum possible extent in order to improve the efficiency of the algorithm.

### 3.3 Performance Tests

In this section we describe the scalability tests performed with our MD code. To study the effect of granularity we use $N/P = 1000$ and 8000. The number of processors $P$ varies from 1 to 256 and the maximum number of atoms is 1 million. The tests are performed multi Tera-flop machines Intel Xeon-based Linux cluster SuperMike at Louisiana State University, 512-processors Columbia19 (NUMALINK) at NASA and Intel Pentium IV Linux cluster SuperHelix at Louisiana State University (see Table 3.1). In SuperMike 512 dual-processor nodes are connected by Myrinet network, which consists of 24 switch units and 64 nodes are connected. On Linux cluster as shown in figure 3.7, the execution time increases only slightly with the number of processors, but we see a sudden jump in the execution time for 1024 processors. This is due to memory-sharing effect as two processors on a motherboard share the main memory.

Initially the performance measurements were conducted on SuperHelix Linux Cluster by varying $P$ up to 64 processors with $N/P$ fixed at 8640 atoms as shown in figure 3.8. As observed from the figure the computation time is in coherence with the ideal computation time expected. However, there is steady trend of increase in the case of communication time per each MD step with the increase in the number of atoms. The results from the test runs on SuperHelix were
useful to confirm the validity of the input configuration and the molecular dynamics algorithm to proceed further to large scale simulation.

Table 3.1 Specification of Linux clusters.

<table>
<thead>
<tr>
<th>Label</th>
<th>SuperHelix</th>
<th>SuperMike</th>
<th>Columbia</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>Intel Pentium IV 3.0GHz</td>
<td>3.0 GHz Intel Xeon</td>
<td>1.5GHz Itanium2</td>
</tr>
<tr>
<td>Number of processors</td>
<td>256</td>
<td>1024</td>
<td>512</td>
</tr>
<tr>
<td>Peak speed</td>
<td>3.7 Tflops</td>
<td>3.7 Tflops</td>
<td>3.1 Tflops</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>512 KB/processor</td>
<td>512 KB/processor</td>
<td>6 MB/processor</td>
</tr>
<tr>
<td>Memory</td>
<td>2GB/2-processor-node</td>
<td>2GB/2-processor-node</td>
<td>2GB/1-processor-node</td>
</tr>
<tr>
<td>Band width</td>
<td>250MB/proc (bidirectional)</td>
<td>250MB/proc (bidirectional)</td>
<td>125MB/proc (bidirectional)</td>
</tr>
</tbody>
</table>

Figure 3.7 Total execution time and communication time of MD on SuperMike
Figure 3.8 Total Execution time and communication time of MD on SuperHelix

The following figure 3.9 depicts the Computation time per MD step for SuperMike and Colombia19 as a function of the system size, N (expressed in the units of 8,000 atoms) and the number of the processors used P. A perfect linear scalability (shown by dashed horizontal line) means that the communication time does not vary with system size if N/P is kept fixed. The performance measurements conducted on SuperMike Linux Cluster by varying P up to 256 processors with N/P fixed at 8,000 and 1,000 atoms are shown in Figure 3.9. The computational cost for N/P = 8000 is somewhat better than that for Colombia19, consistent with the difference in the processors’ speeds between two machines. By decreasing the number of atoms per processor, we have found a substantial improvement in the computation time. For instance, when the total number of atoms in the system is equal to 256,000, the execution time with 256 processors improves by a factor of 5.2, compared to that with 32 processors (figure 3.9). In SuperMike cluster, two processors on a motherboard share the main memory. We have also studied the effects of main memory sharing processors on performance by using one processor per node versus both processors per node and found that the effects are small but not negligible. Using single processor per node actually improves the performance by about 10% for our PCMD program. This is consistent with previous study, which showed a similar level of effects of
memory sharing in the case of the parallel algorithms based on MD and density functional theory [5].

Figure 3.9 Computation time per MD step for Linux Cluster and Colombia19

The communication time shows some increasing trend with increasing number of processors for both machines (Figure 3.9). The increase is relatively more rapid for Colombia19, compared to SuperMike. The calculated ratios between communication and execution times for the maximum number of processors (256) used are 0.52 and 0.1%, respectively, for Colombia19 and SuperMike. Also, the communication time significantly improves when the number of atoms per processor is decreased (see, the dashed lines in Figure 3.9).
Chapter 4: Visualization of Polycrystalline Grain Structure

In this chapter, we discuss about the program which is used to visualize the polycrystalline grain structure containing multimillion atoms belonging to different grains with random/preferred orientations. The 3D time-dependent positional data that are generated from the simulation output are fed as input to the Visualization program. Substantial data processing and specific OpenGL features are used so as to make the program more efficient and simultaneously reduce the need for hardware requirements without losing the quality of the animated pictures that are rendered on the screen. Basically the input data is time dependent. That is, this dataset consists of several subsets of data where each subset of data represents positions or atomic coordinates of the atoms at a particular instant of time.

4.1 Visualization Model

Basically the simulation box is represented by a bounded 3D cube containing the atoms. Each atom is represented by a sphere. The arrangement of the spheres (atoms) within the grain reflects the orientation of the grain. Atoms within the same grain have the same orientation whereas atoms in the neighboring grains have different orientations with respect to each other. Thus the boundary of each individual grain structure is reflected from the orientation of atoms in that individual grain.

In case of Magnesium Oxide the atoms can be viewed as layered structures, that is, the atoms are grouped together to form 2D layered structure. These layers are lined up along a particular plane at regular intervals to form a 3D structure. Initially the atoms belonging to each molecule are supposed to be along the layer to which they belong. After simulation the atoms tend to displace from their equilibrium position. Depending upon the direction in which a particular atom is displaced from the equilibrium, the atom is given a specific color (say red).
and the atoms that are displaced in opposite direction from the equilibrium is given another specific color (say blue). The above process gives a clear picture about how many atoms are displaced in each direction from the equilibrium.

4.2 Implementation

The basic issue of concern is how to handle the large dataset of the atomic configuration on computer systems having normal hardware specifications without losing the quality of the picture. In general the structure of the atoms that are in front of the viewer or viewpoint block the view of those which are behind them. Therefore the data related to these atoms may not prove to be of much use during visualizing them from a particular viewpoint. This data can be safely discarded that is this data need not be loaded into the memory during the run time. In this way the dataset can be pruned to considerable extent in order to reduce computational complexity and improve the efficiency of the program.

4.2.1 OpenGL & GLUT Library

The software for visualization is built using C language and OpenGL GLUT library. OpenGL is the premier environment for developing portable, interactive 2D and 3D graphics applications. Since its introduction in 1992, OpenGL has become the industry's most widely used and supported 2D and 3D graphics application programming interface (API), bringing thousands of applications to a wide variety of computer platforms. OpenGL fosters innovation and speeds application development by incorporating a broad set of rendering, texture mapping, special effects and other powerful visualization functions. Developers can leverage the power of OpenGL across all popular desktop and workstation platforms, ensuring wide application deployment. The capabilities of the OpenGL library allow developers to use it in diverse fields such as in Broadcasting, CAD/CAM/CAE, entertainment, medical imaging, and virtual reality to
Produce and display incredibly compelling 2D and 3D graphics. GLUT stands for OpenGL Utility Toolkit which is an API for OpenGL. GLUT is very easy to set up and use compared to traditional Win32 OpenGL. Because GLUT handles input and window setup, GLUT code is not platform specific, meaning any application you write will run on just about any platform. Some specific features of the OpenGL which are prominent to above software are discussed in the following sections. There are also specialized functions that are useful handling control over input and output devices. There are mouse functions which helps the user to deal with the mouse events like right button click, left button click, etc. Also there are specific functions that are devised to handle various keyboard events. This feature for event handling in GLUT makes it a fully powerful tool to develop scientific graphics with good user interface.

4.2.2 Different Modules in the Software

The above discussed process is implemented as program which contains the following modules. Each module incorporates individual functionality which reduces the communication overhead between each of the modules. Thus the dependency between the functional modules is reduced to optimal extent.

4.2.2.1 ReadConf

This module is responsible for reading the data from the input data from the specified input file which contains the information about the dimension of the simulation box, total number of atoms present in the simulation box and the positional co-ordinates of each atom after each simulation along with the name of each atom. Generally this dataset is voluminous (containing up to 400,000 or more atoms on the whole) and hence the data related to these atoms are stored in the heap memory. The data from this dataset is retrieved on “Need To Use” basis while the program is getting executed.
4.2.2 InitView

In this module basic environment for visualizing the molecular dynamics is established. That is Lighting conditions like position of the Light source, intensity of light, etc are established. Also the position of the View point, center of the view port and positional coordinates of the Viewer’s eye are ascertained to a fixed value. The characteristics of the material used for building graphics objects like spheres and their properties like shining, opacity are also determined in this module.

4.2.2.3 MakeAtoms

This module is used for plotting molecules and their layered structure. The basic properties of the atoms like radius of the atom, color of the atom are determined in this module based on certain conditions. That is, the user is prompted to select the criteria to display the picture. For example the radius of atoms, the color of the atom can be determined depending on the user’s choice. Basically the coordinate system origin is translated from the center of the simulation box to the position of a specific atom and the atom is plotted. After the atom is plotted, the coordinate system origin is shifted back to the center of the simulation box. This process is followed for plotting every atom in the simulation box.

4.2.2.4 Display and Reshape

The Display module plots the simulation box in which the molecules are placed. An important task of the display function is to call or invoke the Display list which is used in this case to plot the atoms. More information about the Display list is given in following sections. The Reshape module is invoked when ever the view port is resized. The Reshape module takes care of the aspect ratio of the view port when ever the view port is resized and also set the projection to either orthographic projection or perspective projection.
4.2.3 User-Interface

Several features have been explicitly implemented in this software to make it more User friendly. The user can control the picture displayed on the screen through input devices like mouse, keyboard, etc. The software incorporates special mouse functions which enable the user to control the rotation of the simulation box. That is, the simulation box can be rotated in all possible directions in the 3 dimensional space using the mouse that is attached to the system. This helps the user to look at the picture of atoms from any point in any direction and get the required information. Also, certain functionality is included in the program which enables the user to control the picture through the I/O devices and specify certain features for display in an interactive manner. The users can choose the option that atoms belonging to each grain are colored differently. Also, the radius of atoms is determined by name of the atom where as the color of the atom is determined by the direction of its displacement from its ideal equilibrium position or vice versa depending on the user’s choice. All the above functionality is provided with the software so that the user can interact more dynamically during the runtime.

4.3 Visualization of Grain Boundaries

Since all the molecules that belong to a particular Voronoi region or grain are oriented in the same fashion, thus the orientation of all the molecules with in a particular grain follows a uniform pattern. At the boundary of the grains the molecules are from two or more different neighboring grains and hence the orientations of the molecules differ from each other. This non uniformity leads to clear cut demarcation. Thus the shape of a particular Voronoi region or grain is implicitly delineated due to the difference in the orientations of the molecules from different grains. By rendering the molecules with in the same grain with same color and simultaneously by making sure that molecules belonging to different grains have different colors.
Hence the representation of the grain boundaries is made more elegant and clear. In the case of visualization of a particular molecule the basic structure of that molecule is implicitly reflected due to the distribution of atoms within each grain. From the basic structure of the molecule some important scientific inferences can be made which will prove to be of good use while analyzing the results of simulation pertaining to that particular molecule.

Figure 4.1 Initial equilibrium positions of the atoms (side view)

For instance, in the case of two grain structure, the simulation box is divided into two grains by boundary surface which are perpendicular to the X-Y plane. The molecules are so placed that they form layers perpendicular along X-Y plane along the Z-axis. Initially all the molecules are at their equilibrium positions which happens to be the exact position of the layer to which these molecules belong to. After the certain period of time during simulation these molecules tend to move away from their equilibrium positions. These displacements can be tracked by the present
software by giving a characteristic color to the direction of the displacement of the atom from its equilibrium position (layer position). That is, atoms that are displaced in same direction from the equilibrium are given the same color. The displacement of the atoms and the layered structured along X-Y planes for different Z-values can be visualized from the top or side view of the simulation box as shown in the figures 4.1 and figures 4.5.

Figure 4.2 Initial equilibrium positions for the atoms (front view)
Figure 4.3 Three Dimensional view of atoms in initial equilibrium position

Figure 4.4 Displacement of atoms after 3000 time steps (top view)
Figure 4.5 Displacement of atoms after 3000 time steps (front view)

Figure 4.6 Three Dimensional view of atoms in initial equilibrium position
Figure 4.7 Represents two grain structures at initial time (front view)

Figure 4.8 Represents two grain structures at initial time (side view)
Figure 4.9 Three Dimensional view of two grain structures

In the all of the previous figures green color indicates the displaced atoms along upwards direction and red indicates the displaced atoms along downward direction. But in the following figures magnesium atoms are represented by red color and oxygen atoms are by green color. The orientation of molecules with in each grain is clearly evident from above figure.
Chapter 5: Conclusion

We have developed a Parallel Computing and Visualization Framework for Polycrystalline Minerals. This framework is a comprehensive package which will prove to be a powerful and useful tool in the study of geological properties of minerals from earth’s mantle. Basically Voronoi geometry is used to generate the Input configuration which emulates the polycrystalline structure of the minerals from Earth’s mantle. We have developed/implemented a parallel PCMD (Polycrystal Molecular Dynamics) algorithm based on the spatial decomposition approach and dynamic management of distributed linked cell lists. Its performance tests have successfully been demonstrated on 1024-processor Linux cluster (SuperMike) and 512-processor Colombia19 using up to 256 processors and a few millions of atoms. The calculated timings of less than 20 seconds per MD step suggest that large-scale simulations with Ewald Coulomb are now feasible on moderate computing resources. We plan to extend these tests to the larger number of processors and larger system sizes.

The visualization software is designed to deal with voluminous datasets in a very effective manner. The software provides good quality of images with minimum hardware requirements. The software incorporates good number of interactive features which help the user to analyze the simulation results efficiently. It is expected that this framework will be useful to simulate mechanical properties (deformations, dislocations) of major earth forming silicate and oxide mineral phases at extreme conditions of the deep interior.
References


[22] www.Qhull.org. The site which provides comprehensive information on Qhull.

[23] Computational Geometry in C by Joseph O’Rourke.

[24] www.OpenGl.org The official site which provides comprehensive information on OpenGl.


Vita

Venkatasrirama PavanKumar Yerraguntla was born in Tanuku, Andhra Pradesh, India, on December 5, 1979. He is the son of Mrs. Venkata Sundaramma and Mr. Satyanarayana Yerraguntla. He received his bachelor’s degree in computer science from Andhra University, India, in 2002. He is currently a master’s student in computer science Department at Louisiana State University. He joined Louisiana State University, Baton Rouge, to pursue his master’s degree in systems science in August 2003. His research areas include molecular dynamics, parallel algorithms and computational geometry.