

DEVELOPMENT OF A USER-FRIENDLY
MOLECULAR DYNAMICS (MD)
SIMULATION SYSTEM FOR
NANOMETRIC CUTTING
AND TRIBOLOGY

By

MATHEW S. LEE

Bachelor of Science

Oklahoma State University


Stillwater, Oklahoma

2000

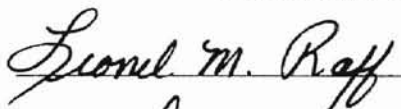
Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 2002

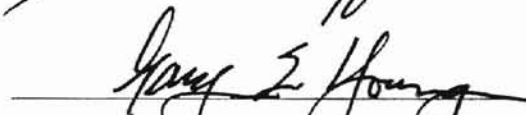
DEVELOPMENT OF A USER-FRIENDLY
MOLECULAR DYNAMICS (MD)
SIMULATION SYSTEM FOR
NANOMETRIC CUTTING
AND TRIBOLOGY

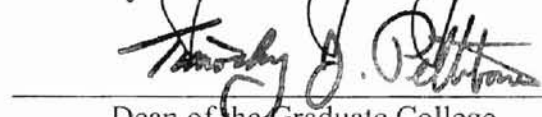
Thesis Approved:



Thesis Advisor







Dean of the Graduate College

ACKNOWLEDGEMENTS

First and foremost, I would like to thank God for giving me the ability to do this work. My family deserves my sincere thanks for the help and support they have provided and continue to provide throughout my career. I would like to give a special shout out to my girlfriend, Erika Nevin. God bless our relationship.

A special thanks also goes to my graduate advisor and mentor, Dr. Ranga Komanduri. His friendship, encouragement, and discussions have helped guide my work to be the best of my abilities. Thanks are also due to Dr. Lionel Raff for the many meetings and ideas for solving problems. His expertise has been invaluable to my understanding of MD. Thanks also to Dr. Young for serving on my committee.

Much of the work for this project was completed using the freely available operating system, Linux, initiated in 1994 by Linus Torvalds. Thanks are due to his original work and the continuing work of open source developers around the world who are rarely given credit for the powerful tools they have created.

Thanks are also due to my fellow research colleagues who have provided useful discussions with different perspectives on various problems: Milind Malshe, Rutuparna Narulkar, and David Stokes. My humblest gratitude goes to Joe Hershberger, master programmer and friend, for his programming help, expertise, and masterminding the MDbinfmt library.

TABLE OF CONTENTS

Chapter	Page
I. INTRODUCTION.....	1
1.1. Exploring Manufacturing Processes in the Nano Region.....	1
1.2. Molecular Dynamics (MD) Simulations.....	2
1.2.1. Potential Energy Functions.....	3
1.2.2. Moving, Boundary, and Peripheral Atoms.....	5
1.2.3. Numerical Simulations.....	6
1.3. Parallel Processing Approach to Solving MD Simulations.....	7
1.4. Background Information and Review of MD at OSU.....	8
1.5. Thesis Outline.....	9
II. PROBLEM STATEMENT	13
2.1. User-Friendly System.....	13
2.2. Justification.....	14
2.3. Solution Approach.....	15
2.3.1. Administrator's Perspective.....	15
2.3.2. Programmer's Perspective.....	16
2.3.3. End-user's Perspective.....	18
III. PARALLEL COMPUTING VIA BEOWULF CLUSTERS	21
3.1. Introduction.....	21
3.2. Security Issues.....	25
3.3. Hardware Overview.....	28
3.4. Construction.....	32
3.5. Installation and Configuration.....	37
IV. PROGRAMMING APPROACH FOR MD SIMULATION	38
4.1. Introduction.....	38
4.2. MDbinfmt Library.....	40
4.3. Parallel Simulations.....	43

Chapter	Page
V. IMPLEMENTATION OF USER-FRIENDLY SOFTWARE.....	47
5.1. Introduction.....	47
5.2. Overview of Software Operation.....	48
5.3. Pre-Processing Software Implementation.....	50
5.4. Simulation Software Implementation.....	51
5.5. Post-Processing Software Implementation.....	52
VI. APPLICATION OF USER-FRIENDLY SYSTEM TO NANOMETRIC CUTTING.....	55
6.1. Introduction.....	55
6.2. Simulation Process.....	57
6.3. Results.....	59
VII. CONCLUSIONS.....	65
7.1. General Conclusions.....	65
7.2. Specific Conclusions.....	67
7.3. Future Work.....	68
REFERENCES.....	70
APPENDIX A.....	75
APPENDIX B.....	106
APPENDIX C.....	118

LIST OF TABLES

Table	Page
6.1 Parameters used in the oblique simulation investigation.....	57
6.2 Computational parameters used within the simulation software.....	58

LIST OF FIGURES

Figure	Page
1.1 Typical nanometric cutting simulation with atom types labeled.....	6
2.1 MDbinfmt library file operations usage.....	17
2.2 End-user perspective of simulation software.....	19
3.1. Clock time versus number of processors @ 1.7 GHz.....	24
3.2 Schematic of how packets travel iptables chains.....	28
3.3 Overview of MDbeta parallel environment.....	31
3.4 Picture of MDbeta computational node.....	33
3.5 Assembled file-server node.....	35
3.6 Assembled nodes and mounted in rack.....	36
4.1 MDbinfmt library overview.....	42
5.1 End-user view of simulation software.....	48
5.2 Diagram of pre-processing operation.....	49
5.3 Diagram of simulation operation.....	49
5.4 Screenshot of MDii pre-processing software.....	51
5.5 Screenshot of the MDui animation software showing nanometric cutting.....	54
6.1 Orientation of the tool creating the inclination angle.....	55
6.2 Diagram of the oblique machining operation.....	56
6.3 Snapshots showing various stages of oblique nanometric cutting.....	60

6.4	Various orientations of the simulation that can be animated using MDui.....	61
6.5	Chip flow angle vs. angle of inclination.....	62
6.6	Variation of the chip flow angle with the angle of inclination.....	64

NOMENCLATURE

AMD-Advanced Micro Devices, manufacturer of the MDbeta processors

AI-Artificial intelligence

API-Application programming interface, library used for accessing prebuilt user interfaces

Beowulf cluster-Supercomputer built from commodity hardware components

Bpbatch-Application used to remotely boot the nodes in the MDbeta cluster

CAT-5-Twisted pair cable standard used to connect nodes to the network switch

CISC-Complex instruction set computer

COTS-Commodity-off-the-shelf

Communication-Message passing from one node to the next

CVD-Chemical vapor deposition

CPU-Central processing unit, also known as a processor

Data structure-Term used in programming to describe variable layout

DEC-Digital Equipment Corporation, purchased by Compaq and now Hewlett Packard

DHCP-Dynamics host configuration protocol, used to assign network addresses

DirectX-Graphics library used for windows

DVD-Optical disk format capable of handling 4.7 Gb per side

EAM-Embedded atom model

ECC-Error correction control, used to double check errors in memory

ia32-32-bit Intel based processor

ia64-64-bit based processor

FCC-Face centered cubic

FDM-Finite difference method

FEM-Finite element modeling

Function-Term used in programming to describe a piece of code that can be compiled that has defined inputs and outputs

Glui-OpenGL user interface, cross platform API for OpenGL

Glut-OpenGL utility toolkit, API for common OpenGL functions

HP-Hewlett Packard

Kernel-basic piece of code that allows software to interface with hardware

Load Balancing-Distributing of the computational work between nodes

Linux-Freely available operating system used as the operating system of the Beowulf cluster

Library-Term used in programming to describe a precompiled piece of code that can be accessed during runtime

LLNL-Lawrence Livermore National Labs

Mbit-Million bits per second

MC-Monte Carlo

MEAM-Modified embedded atom model

MD-Molecular dynamics

MDalpha-First Beowulf class supercomputer constructed using ia-64 processors

MDbeta-Second Beowulf class supercomputer constructed using ia-32 processors

MDbinfmt-Library created to help with atomistic simulations

MDii-Molecular dynamics input interface, created in this study for pre-processing

MDiso-Custom installation disk created for installing the Slackware Linux distribution

MDui-Molecular dynamics user interface, created in this study for post-processing

MDsetupc-Console version of MDii that works in Linux operating system

MFLOPS-mMllion floating point operations per second

MIMD-Multiple instruction, multiple data

MPI-Message passing interface standard

MPICH-Library implementing MPI standard

NFS-Network file system, provides a common remote file system

OpenGL-Graphics library that allows for simulation animation

PCI-Peripheral component interconnect

PVFS-Parallel virtual file system, uses NFS across multiple nodes to increase file throughput and decrease file access

PXE-Pre execution environment used for booting the machines from the network

RISC-Reduced instruction set computer

Scaling-Increasing parallel simulation size without noticing effects from the communication

SSH-Secure shell protocol used for communications with remote machines

SIMD-Single instruction, multiple data

Tribology-Study of friction, wear, and lubrication

Tru64-Unix operating system, created by DEC

CHAPTER 1

INTRODUCTION

1.1. Exploring Manufacturing Processes in the Nano Region

As technology advances from micro to nano scale, there will be an increase in the need for the development of new manufacturing techniques for ultra small devices. In order to create new manufacturing techniques, an understanding of material properties at the nano level is needed. Two approaches can be used to explore these important areas, namely, experimental and theoretical.

Equipment limitations exist with current experimental techniques at the scale future technology demands. In many cases, experiments performed are difficult to recreate because geometries and crystallographic orientations of the specimen used cannot be replicated easily without orientation characterization by x-ray diffraction measurements. In addition to the difficulties encountered during setup, the specimens used in these tests are expensive and not reusable.

Theoretical approaches to simulating micro, macro, and full-scale phenomenon can be accomplished by continuum methods such as the Finite Element Modeling (FEM) and Finite Difference Methods (FDM). The basics of FEM were developed by Hrenikoff [1] and Courant [2] in the early 1940's. However, the FEM method was not formally proposed until the late 1950's by Argyris and Kelsey [3] and Turner *et al.* [4].

As the scale of the simulation approaches the nano scale, materials must be modeled as discrete points rather than a continuum. A technique, called molecular dynamics (MD) [5,6] is capable of modeling the molecular interactions so that simulations can be performed at the nanometer scale. MD provides a mechanism for studying the molecular interactions by numerically modeling and simulating material interactions. MD provides a tool that can be used to explore areas that are physically difficult, if not impossible with current experimental technologies.

MD simulation techniques can be applied to a large number of engineering problems. The downside to the technique is the computational power required to perform a simulation and overcoming the difficulties associated with the development and software coding of interaction potentials. Also, software created for the end user should be easy to use. To distribute the large computational overhead, Beowulf clusters, the supercomputers of the future, can be implemented to rapidly conduct large-scale simulations, yet allow multiple small simulations to be run simultaneously.

1.2. Molecular Dynamics (MD) Simulations

Molecular Dynamics (MD) simulations are highly coupled systems that follow Newton's equations of motion. Simulations range in size from a few hundred to several thousand atoms. Each atom is described by 6 coupled equations, 3 coordinate and 3 momenta [7]. To calculate new positions of atoms with respect to time, a Runge-Kutta differential integration [8] routine is used. Forces of the interactions on the atoms are needed several times during the differential integration process and are computed as the derivative of the interatomic potential.

1.2.1. Potential Energy Functions

The interatomic potential embodies the governing mathematical model for MD simulations. It aims to mimic or represent the physical and chemical interactions of the atoms such as lattice spacing, thermodynamic properties, bond interactions, molecular weight, equilibrium position, and more. Complexity of the potential ranges from simple pair-wise interactions to complex multiple body interactions with electron embedding energies. These empirical interatomic potentials are implemented to provide a mechanism for calculating an ensemble of atom trajectories in a reasonable amount of time. The accuracy, usually related to the complexity of the potential, dictates the quality of the final simulation. The alternative to the empirical potential is to solve the theoretical interactions, which are extremely complex but have been approximated by means of *ab initio*, or first principle calculations [9]. Unfortunately they are extremely time-consuming to calculate even with the fastest supercomputers to date.

Many empirical interaction potentials have been developed. One common pair-wise potential used in MD simulations is the Morse potential. This potential was originally derived for dimers but has been shown to adequately model the interactions between atoms in some face centered cubic (FCC) systems [10]. The functional form of the Morse potential is $V_r = De^{-2\alpha(r-r_{eq})} - 2De^{-\alpha(r-r_{eq})}$ where the parameters D , α , and r_{eq} are defined by fitting physical and chemical properties of the material. Another simple pair-wise potential is the Leonard-Jones potential, originally defined for inert gasses with van der Waals-type cohesion forces [11, 12]. The functional form of this potential is $V_r = 4\epsilon \left(\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^6 \right)$ where the adjustable parameters are ϵ and σ .

The limitation of the pair-wise potential is that it models the interaction between two distinct atoms and neglects to account for contributions of the remaining atoms. In other words, each pair of atoms is considered to be in isolation from the entire system for each calculation. Tersoff [13] developed a potential aimed at considering local interactions and handling angular effects on neighboring atoms.

Other advanced potentials, described below, can be used to account for and approximate the complexities of nature. Nonetheless, these atomic interactions will still be empirical. The embedded atom model (EAM) and modified embedded atom model (MEAM) are two empirical potentials that have been developed and show promise for more accurately replicating the behavior of atoms in the nature. However, these potentials are extremely complicated in comparison to the pair-wise potentials and require significantly more computational time. For a detailed synopsis of the EAM and MEAM potentials refer to the literature [14-16].

Replacing the trajectory calculation with other methods to speed up the simulation process is a viable alternative. One such method, shown to give good results, is Monte Carlo (MC) simulations [17]. This method neglects the calculation of atom trajectories by randomly moving atoms until the minimum potential has been acquired. These random moves are monitored by criteria such as the number of accepted to rejected random moves. Artificial Intelligence (AI) is another promising alternative to calculating the atomistic interactions during a simulation. This method employs training a neural network [18] from first principle calculations or MD potential calculations, and then using the neural network to predict interactions.

1.2.2. Moving, Boundary, and Peripheral Atoms

In MD simulations of nanometric cutting and tribology, there are three types of atoms [19]. The first is, boundary atoms, which helps to provide stability to the atomic structure of the system by simulating the surrounding bulk structure. Interactions of boundary atoms with other boundary atoms are neglected while interactions between different atom types are computed. The boundary atom layer should be thick enough so that interactions of the second nearest neighbor atoms are not neglected. The second type is the moving atom upon which no restrictions are placed. It is free to move in any direction with any velocity as long as it satisfies the trajectory calculation from the Runge-Kutta differential integration. The third type is the peripheral atom, also called thermostat atom, which facilitates simulating the properties of the bulk material by providing a mechanism for transferring heat generated from the moving atoms in the simulation to the bulk.

These three types of atoms are used to classify and identify different regions within the simulation. When performing MD simulations, there are many energy related calculations for a given atom per integration step. This means that, as the number of atoms involved in the simulation increases, the processing time required increases immensely. Atoms that are not essential to the simulation, such as those lying in the bulk, should be removed or their effect not considered decreasing computational time. However, effects from these removed atoms must not be neglected. In many nanometric cutting and tribology simulations, large crystal deformations occur, which generates large amounts of heat that must be transferred to the bulk and removed away from the process

interaction region. Figure 1.1 shows a typical MD simulation scheme for nanometric cutting.

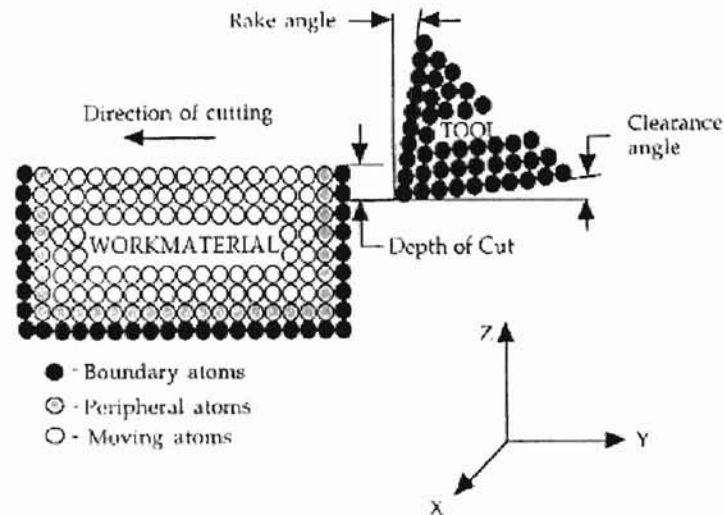


Figure 1.1 – Typical nanometric cutting simulation with atom types labeled

1.2.3. Numerical Simulations

A powerful feature of numerical simulations is the ability of the programmer or user to control every aspect of the simulation. Information can be collected during and after the simulation to help the researcher identify important phenomenon. Modification of input parameters, materials, geometries, velocities, and other important input parameters can be accomplished by using specially designed pre-processing software that is used to generate the input files for the desired MD simulation of nanometric cutting and tribology.

Another powerful feature of numerical simulations is the ability to interpret the results by viewing, rotating, animating, and analyzing the simulation after it has been completed. This post-processing is accomplished by storing simulation information, such as coordinate data, forces, and energies, into a file that can be used by post-processing

software. The animations created by the simulation output files give the researcher a glimpse into areas of molecular interactions that are virtually impossible to explore using experimental techniques.

1.3. Parallel Processing Approach to Solving MD Simulations

One of the drawbacks to MD simulation relates to the time required to compute a simulation. Real time simulation durations currently being performed are on the order of a few pico seconds. This requires simulation speeds to be many orders of magnitude greater than what is commonly used at the macro level. There are several ways to decrease the computational overhead when performing MD simulations of nanometric cutting and tribology. The first is to use parallel processing or distributed computing. The second is to employ new techniques and algorithms for solving problems that operate faster than the current algorithms employed. These new techniques include implementing algorithms based on the linked cell method [20], optimized integration routines [8], and multi-step time based regions [21, 22].

Researchers have demonstrated that the use of parallel processing decreases the amount of time spent solving complex mathematical problems, such as MD simulations [23, 24]. Parallel processing provides a mechanism for distributing different parts of the simulation between multiple processors. However, implementing a code for performing parallel simulations requires an enormous investment of time for developing and testing. The programmer must be knowledgeable with hardware, operating systems, coding, and MD. Implementations that decrease the programming overhead, such as the *adhara* library [25], should be explored before effort is spent creating parallel versions of MD.

Beowulf clustering [26] employs one form of the parallel processing paradigm and allows for the construction of rather inexpensive massively parallel supercomputer that fits budgets for some research groups, universities, colleges, and even some small businesses. This clustering technique can be constructed or purchased from a growing number of computer vendors. Other alternatives to the parallel possessing paradigm can be explored by purchasing supercomputers that are extremely expensive and are rapidly becoming obsolete by the Beowulf clustering technique.

1.4. Background Information and Review of MD at OSU

The initial work on nanometric cutting by means of MD came from the pioneering work of Belak *et al.* [27] at Lawrence Livermore National Labs (LLNL). They simulated the orthogonal metal cutting process of a copper workpiece using a diamond tool. Soon thereafter, Dr. Ranga Komanduri and Dr. Lionel Raff created a cross-disciplinary collaborative research group at Oklahoma State University aimed at studying nanometric cutting and tribology.

The original collaborative work started with attempting to model the formation of diamond coatings by chemical vapor deposition (CVD). Since that time, the MD research group has coded the Morse, Tersoff, EAM, and MEAM potentials while simulating a number of different processes. Some of these processes include the study of atomic scale friction [28], tension [29], indentation and scratching [30], orthogonal cutting of Si [31], length restricted molecular dynamics [32], effect of tool geometry in orthogonal cutting [33], exit failure [34], cutting through grain boundaries [35], crystal orientation and direction of cutting in orthogonal cutting [36], extrusion, grinding, and

milling using multipoint tools. All of these previous studies were conducted in 2D under plain strain conditions. Recently, 3D animation capabilities have been introduced and work is continuing to explore these areas.

Another area of interest is to expand current simulation techniques to new methods of simulation, such as MC and neural networks trained by *ab initio* data to develop potentials. These recent explorations are aimed at decreasing the computational time required to perform a simulation in the former case and develop more accurate potentials in the latter. The MC method replaces the calculation of the trajectories by using random numbers along with minimum potential criterion to calculate the next position of the atoms. With this method, simulating a cutting speed is not intuitive. Thermal gradients or other means must be used to relate the cutting speed to the current simulation. The MC method has been programmed using the recently developed user-friendly system and helped to show the usefulness of the designed system.

1.5. Thesis Outline

To facilitate in the creation of a serial or parallel computational simulation software, a user-friendly system must be created. It should provide the computational backbone, hardware, operating system, user interface (for end-user simulation), and a reference for the simulation software. This implies that the term user-friendly extends beyond the exterior interface that the users utilize for creating and running simulations.

A user-friendly system for MD can be defined at several distinct levels. The most obvious one is that which the user of the software encounters, namely, the end-user tier. This tier encompasses the visible interface and usage of the applications. The

applications should be easy to use and provide for the creation, simulation, and animation of nanometric cutting and tribology MD simulations. Another important level is the software programming tier. This level provides a mechanism for creating and modifying simulation software. The last level is the administration tier. This provides the hardware and computational capabilities that the programmer needs to solve MD simulations in both serial and parallel modes.

In this chapter, a general introduction to MD is given. Various simulation approaches for nanometric cutting and tribology simulations are discussed. Simulation potential selection, alternative simulation methods, and improvements to simulation processes are covered. Introduction to parallel processing, Beowulf clustering, and the need for these tools at decreasing the required simulation clock time are given. A brief discussion on the origin of MD as well as the history of MD simulations within the collaborative research group here at Oklahoma State University is given so that future researcher will have an understanding of the background of the available knowledge as well as a baseline for the previously written software.

In Chapter 2, the problem statement is given. The approach to this thesis is different than the traditional document in that the aim is to transfer working knowledge to the reader. This allows the researchers interested to grasp important issues while providing a manual on using the system, performing MD simulations, and creating new MD simulations of nanometric cutting and tribology. Identification of the problem is given as well as some background information on the multiple approaches used to create the solution.

Chapter 3 approaches the solution from the administrator's perspective and gives an overview of the hardware aspects of parallel processing. Details include information on the different hardware architecture options to the operating system. The goal of this section of the user-friendly system is to provide the parallel processing and hardware capabilities needed to develop and run simulations. Issues involved with implementing the parallel processing as the basis for the hardware aspects of the user-friendly system such as the security issues are discussed. Past implementations are discussed with a focus on the construction of a powerful Beowulf cluster.

Chapter 4 approaches the solution from the programmer's perspective by giving information on the tools created to supplement some of the programming overhead. Further discussion on parallel processing options and techniques are given. The goal of this section of the user-friendly system is to develop the necessary tools for the development of future simulations. The tools, library functions and structures, and functionality provided by the user-friendly system are discussed in a context of furthering the development of nanometric cutting and tribology simulations in the future.

Chapter 5 approaches the solution from the end user's perspective. Overview of the user-friendly system operation is provided. Implementations of the pre-processing and post-processing software are discussed.

Chapter 6 extends the depth of the end-users view of the system by providing an example on the nanometric simulation of oblique machining. Discussion of important phenomenon relating to oblique machining is discussed, namely the variation between the inclination angle and the chip flow angle.

Chapter 3 approaches the solution from the administrator's perspective and gives an overview of the hardware aspects of parallel processing. Details include information on the different hardware architecture options to the operating system. The goal of this section of the user-friendly system is to provide the parallel processing and hardware capabilities needed to develop and run simulations. Issues involved with implementing the parallel processing as the basis for the hardware aspects of the user-friendly system such as the security issues are discussed. Past implementations are discussed with a focus on the construction of a powerful Beowulf cluster.

Chapter 4 approaches the solution from the programmer's perspective by giving information on the tools created to supplement some of the programming overhead. Further discussion on parallel processing options and techniques are given. The goal of this section of the user-friendly system is to develop the necessary tools for the development of future simulations. The tools, library functions and structures, and functionality provided by the user-friendly system are discussed in a context of furthering the development of nanometric cutting and tribology simulations in the future.

Chapter 5 approaches the solution from the end user's perspective. Overview of the user-friendly system operation is provided. Implementations of the pre-processing and post-processing software are discussed.

Chapter 6 extends the depth of the end-users view of the system by providing an example on the nanometric simulation of oblique machining. Discussion of important phenomenon relating to oblique machining is discusses, namely the variation between the inclination angle and the chip flow angle.

Documentation, in the form of a manual for the installation and configuration of the Beowulf cluster, is given in Appendix A. For researchers unfamiliar with the use of the Linux operating system, information is provided in Appendix B. Appendix C gives details of the structures and functions that are provided in the programming library.

CHAPTER 2

PROBLEM STATEMENT

2.1. User-Friendly System

The purpose of this study is to develop a user-friendly molecular dynamics (MD) simulation system for nanometric cutting and tribology. To create and utilize the system developed in this study, the following two objectives are given:

Objective 1:

To design and implement a system that can be used to develop and perform computational simulations of nanometric cutting and tribology. This system should encompass the computational simulation process from hardware to software while being user-friendly. Documentation should be provided so that as new users are introduced to the system, information is available for guidance on topics that include running simulations, creating simulation software, and maintaining the computational environment.

Objective 2:

To utilize the system developed to meet objective 1 and implement the software needed for an example of nanometric simulation (oblique machining). The software should be designed so that it can be easily modified and expanded as needed.

2.2. Justification

What if knowledge transfer on creating and performing computational nanometric cutting and tribology simulations ceased to occur between successive generations of researchers? This would be disastrous at advancing the technology on simulating these processes. Would the rate of technology advancement in the field of MD slow to a crawl? Researchers creating and performing numerical simulations of nanometric cutting and tribology spend an enormous amount of time recreating software that has been previously written by other researchers. What happens when a previous researcher leaves after completing work? The researcher effectively retains all the information and knowledge that had been acquired while working on the project. In many cases, multiple versions of source code with sparse explanations are found. This requires new researchers to effectively start from the ground up because of difficulties encountered continuing where the previous researcher left off.

These are important questions and ultimately problems to tackle as this research begins to encompass more complex potentials and more powerful computers. The solution to these problems is to implement a system, user-friendly in nature, aimed to help researchers create useful and portable code that can be used for perpetuating the transfer of knowledge from one generation of researchers to the next while advancing the complexities of the simulation software for nanometric cutting and tribology. The ultimate goal of this user-friendly system is to provide an easy to use system that students with different backgrounds can use for creating and performing simulations. Further advancement, using this user-friendly system, allows researchers to focus on the defined problem, rather than duplicating what previous students have accomplished in the past.

With the use of a user-friendly system, the time taken to conduct specific research on nanometric cutting and tribological processes using MD will increase.

2.3. Solution Approach

An approach to the solution requires addressing three perspectives. A clear understanding is needed on how the end-user, programmer, and administrator of the user-friendly system utilize the resources available to create a system that can encompass the needs of the different types of users. Using these three perspectives, tools and information to help each accomplish their goals are provided. The following subsections provide each of these important perspectives so that the reader can grasp the importance of approaching this problem with multiple perspectives.

2.3.1. Administrator's Perspective

Looking at the user-friendly system from the administrator's perspective is often overlooked. The administrator acts as the central head and controls the usage, maintenance, and solves problems when they occur. Without a central head for managing the systems and resources, chaos erupts. Users run multiple simulations effectively creating an expensive paperweight out of the computational resources. Managing the available resources for researchers can create some ill feeling between co-workers. Care must be taken so that end-users and programmers that use the computational resources for developing and executing of software have the resources available. The administrator of the machine should be the only person that handles abuse and security of the computational resources. This is required to maintain the overall

integrity of the system. Batch systems should also be explored as the number of researchers utilizing the cluster increases.

Chapter 3, regarding the parallel computing via Beowulf clusters is aimed at covering the implementation of the user-friendly system from the administrator's perspective. In most cases, the administrator of the system will also be one of the principal researchers on the nanometric cutting and tribology projects.

2.3.2. Programmer's perspective

Looking at the user-friendly system from the programmer's perspective requires an understanding of complex programming, software development, and basics of MD simulations. A programmer utilizing the system creates software that must be expandable yet easy enough for the end-user to operate without requiring the user to understand every detail on how the software is written to perform MD simulations. User-friendly features of the system must not be neglected for the programmer. If the programmer is to create efficient and easy to use software, he/she should have a basic idea of how MD simulations are performed, but not be required to understand every aspect of each piece of software created for MD simulations. A basic understanding of the overall scheme is needed, but easy creation and modification of simulation applications is a necessity to ease software creation and reduce development time. To handle these needs, a special library named MDbinfmt was created. This library helps make the programmer's job easier by taking some of the complexities out of the programming. The library contains functions and structures for accessing atom information during the simulation. Structures are also designed to help the programmer

access large amounts of complex data with simplicity. As multiple code contributions from multiple sources are introduced into the current MD software suite, care must be taken so that multiple students may benefit and utilize code written by other students. Using the MDbinfmt library can make this a reality. The basic functionality of the created library is to give an easy to use programming interface for the data and information store in the data file. Figure 2.1 shows the coding scheme used in conjunction with the special library format.

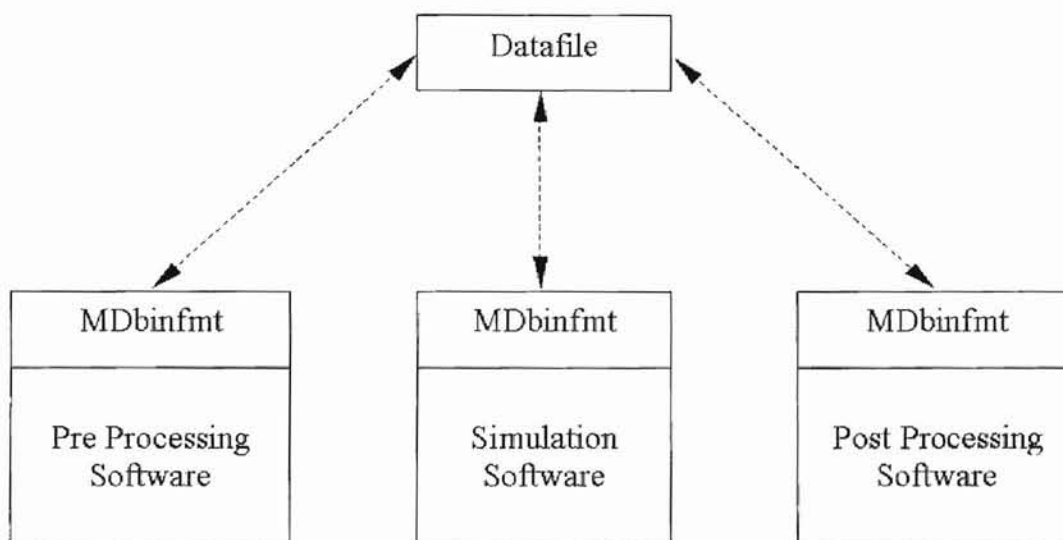


Figure 2.1 – MDbinfmt library file operations usage

To elaborate more on the details that the programmer utilizing the user-friendly system, Chapter 4 has been provided with details on the overall programming scheme for the MD software suite. Chapter 3 gives some discussion on the parallel processing the Beowulf cluster provides. Continuing in further detail, Appendix C provides the programming details such as programming structures, functions, and examples on how to perform some common tasks using the MD library.

2.3.3. End-user's perspective

Most users of the system fall into the end-user category. For this reason, it is easy to identify the important aspects that these users will desire. The first and foremost issue for the end-users is to provide software that is simple to use. Graphics applications should be employed whenever possible. Documentation should be provided on the operation and details of the particular piece of software.

Therefore, software created should provide end-users an application that is capable of completing a multitude of nanometric cutting and tribology MD simulations without requiring them to understand the mathematics behind the simulation. Options for different simulation geometries as well as processes should be provided. The focus of the users should be on creating and exploring desired simulation, performing the simulation, and viewing or analyzing the simulation.

To provide a mechanism for easily creating, performing, and viewing the nanometric cutting and tribology simulations, three applications groups were identified. They are defined as the pre-processing, simulation software, and post-processing groups. Each group provides a separate but powerful piece of software that is capable of performing the desired task. Both the pre-processing and post-processing software runs in the easy to use Windows® environment. The simulation software, because of the need for stability during long simulation durations, runs on the Linux operating system. Fortunately, because these three applications are written using the C programming language, the applications are portable. This means that the applications can be ported to different operating systems so that they will operate in both Linux and Windows.

Nonetheless, users will find the software scheme similar to those found in other computational research software such as a number of FEM and CAD packages. Figure 2-2 shows the MD simulations software suite from the end-users perspective.

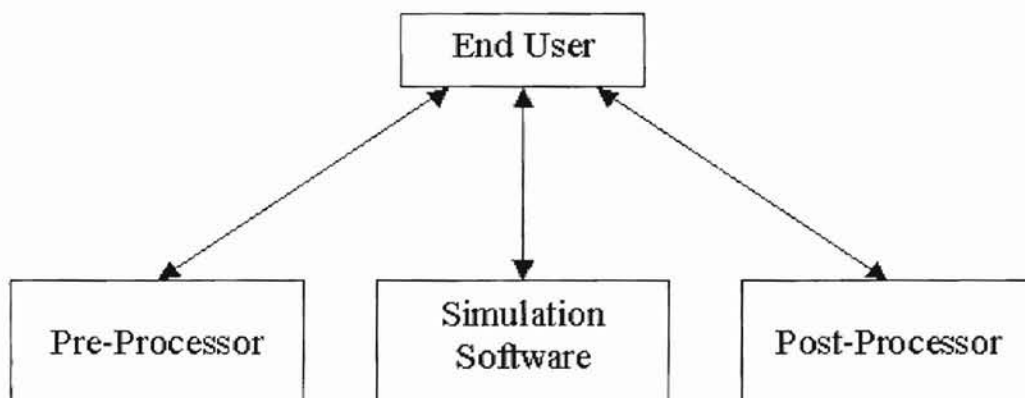


Figure 2.2 – End-user perspective of simulation software

The separation of the applications into three groups allows the programs to exploit the power of either the graphical operating system or the stability of Linux. The pre-processing and post-processing applications were created to run on a graphical operating system. Graphics enhance the user-friendly aspect of the pre-processing application by providing dialog boxes where values can be easily entered and modified for the specific simulation. Graphics also enhance the post-processing application by allowing the user to see the movement of the individual atoms at any given time during the simulation through animation, video creation, and still images. This was accomplished by utilizing the OpenGL graphics library. The simulation software can be exploited to harness the stability of Linux and allow the user to run simulations on multiple computers remotely

without burdening the local computer with large amounts of computational work, which ultimately decreases productivity for the researcher.

Performing MD simulations involves several steps. The first step requires that the simulation parameters be selected, or pre-processing. The second step takes the input parameters and performs the MD simulation. The final step is to view the output of the simulation or do any other necessary types of calculations based on atom positions and other information provided during the simulation process, or post-processing. These are the only steps that the end-user of the software should be required to understand.

The end-user perspective is identified, but the best way to reveal its importance is to provide examples. Chapter 5 provides details on the operating and functionality of the software provided in this study. An example implementation of an oblique machining simulation is also provided to clarify any misconceptions on the operation of the software.

CHAPTER 3

PARALLEL COMPUTING VIA BEOWULF CLUSTERS

3.1 Introduction

MD simulations are inherently computationally intensive. This is because of the small integration time step ($\sim 10^{-14}$ seconds) and large number of equations ($N(N-1)/2$) where N is the number of atoms, roughly in the thousands, that must be simultaneously solved during each time step. Finding adequate computational resources for solving MD simulations is a difficult task. Supercomputers cost per clock time and gaining access to one may be difficult. Unlimited use to such a system for development and testing of software can surmount astronomical costs. A cost effective alternative to the supercomputer is to implement a massively parallel Beowulf cluster that utilizes the commodity-off-the-shelf (COTS) philosophy.

In computer systems, the processors are characterized by the instruction set that the chip implements. There are two common types of instruction sets, the *reduced instruction set computer* (RISC) and the *complex instruction set computer* (CISC). The alpha based processors are 64-bit and implement a RISC instruction set. The Intel or ia32 based systems implement CISC instruction sets. The RISC processor has been glorified as the faster of the two chips. However, recent advances with the CISC chips have surpassed the operating frequency of the RISC processors

and are therefore much faster. While both of these processor types have been used in clustering environments, the CISC processors commonly found in ia32 systems are continually increasing in speed while decreasing in cost. This makes the selection of the ia32 processor an easy task.

There are several types of distributed architectures. The *single instruction, multiple data* (SIMD) architecture allows for multiple data items to be manipulated during a single instruction cycle in a single processing core. *Multiple instructions, multiple data* (MIMD) allows for multiple instructions to occur on multiple data during a single instruction cycle in multiple processing cores. MIMD embodies the most promise for MD simulations.

An alternative to the traditional supercomputer is Beowulf clustering. In the Beowulf cluster, each computer is considered a node consisting of one or more processors while the cluster as a whole adheres to the MIMD architecture. This clustering technique utilizes the COTS philosophy to implement a massive multiprocessor machine that uses a commonly available Fast Ethernet network as the mechanism for communication between each node. Each node is comprised of a standard workstation that is used strictly for computational work or server style computer that runs services. Libraries employing the message passing interface (MPI) standard give the programmer access to the remote processors while hiding the complexities of the specific network communication [36-39]. One important advantage to using a library that implements MPI is the software written is independent of the hardware that makes up the cluster. This feature helps provide portable software capable of running on various MPI enabled clusters.

MD software developed to run on a serial node is limited to run at the maximum speed of a single processor. Using a Beowulf cluster or parallel environment, MD simulation size can be increased dramatically where the limitation is moved from processor speed to the bandwidth and latency of communication between the processors in the cluster. The bandwidth is defined as the maximum amount of information (bits/second) that can be transmitted from one node to the next. The latency of the communication is the duration of time it takes to send the information out the network interface and for the destination node to receive the data.

One drawback in utilizing a Beowulf cluster, as apposed to a commercially available supercomputer, is they are not available with compilers that automatically compile code to utilize the parallel environment. Many hours are required to write and tune software capable of utilizing the constructed clustering hardware. A sample parallel application, employing self-scheduling, to solve the moving heat source problem was performed and shows that using a Beowulf cluster can dramatically increase the computational power available. Figure 3.1 shows the decrease in simulation time as the number of processors increases by using the MDbeta parallel computing system for solving the moving heat source problem with 2000 nodal calculations.

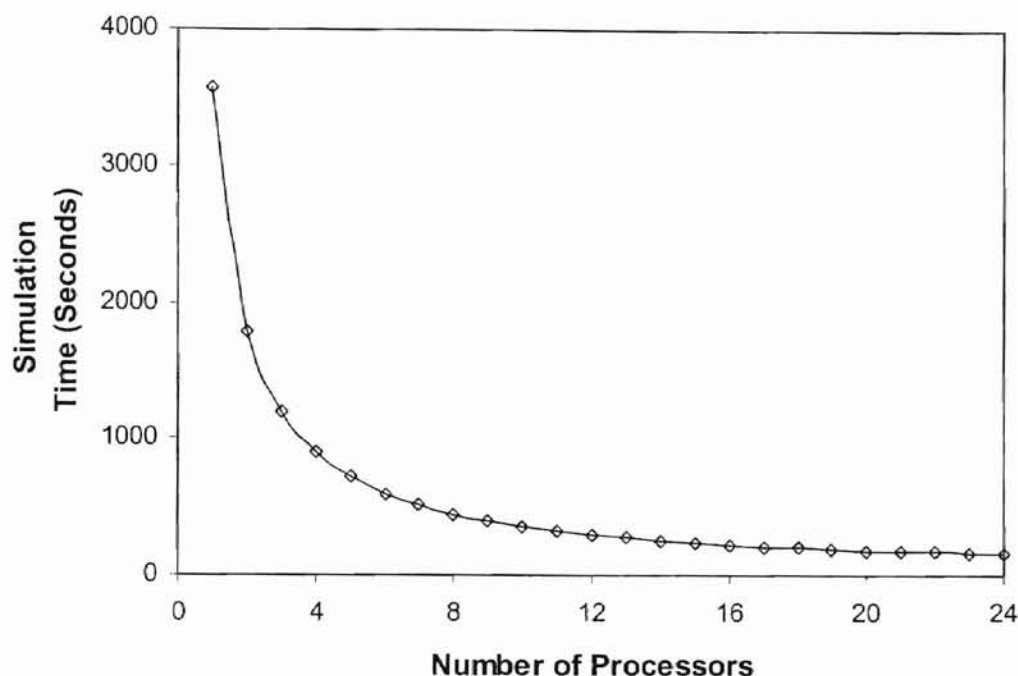


Figure 3.1 – Clock time versus number of processors @ 1.7 GHz

The goal of using the parallel environment is to distribute the computational load of a given simulation across multiple processors to acquire the solution in a decreased amount of wall clock time. The Beowulf cluster may also be used to run multiple serial simulations, simultaneously. Development of parallel enabled applications provides a method to simulate an increased number of atoms allowing for larger, approaching the macro size, systems to be explored. Recent advancements in Beowulf clustering have provided the necessary tools to implement a custom cluster aimed at solving specific problems that the programmer and user encounter. Problems may include difficulties in profiling the parallel enabled software, organizing data during the simulation, and scheduling multiple parallel simulations on a single cluster.

Several operating systems are available for operating a Beowulf cluster. Stability along with minimal software expenses should be sought to get the highest performance-to-cost ratio. The operating system should also be secure and allow for users to access the resources remotely. This allows many users to spawn multiple terminals that access the resources of the cluster. Because of these important features, the Linux operating system along with the freely available message passing interface library (MPICH) was selected. However, any message-passing library designed to operate with the selected hardware or even custom software capable of communicating with the cluster nodes directly through the network interface may be used. Multiple libraries can be installed and selected by the programmer at application compile time. This gives flexibility to the programmer when implementing different parallel algorithms. The main advantage to selecting Linux, as the cluster operating system, is that all of the money associated with the cluster operating system can be used to purchase hardware and not on licensing fees as it is freely available on the Internet. The future administrator of the workstations should use the information provided in this chapter as well as in Appendix A to install, configure, and maintain the hardware aspects of the user-friendly system.

3.2 Security Issues

In order to utilize a Beowulf cluster for effectively simulating large-scale MD simulations, the previous strategies and concerns should be addressed. However, the construction phase of a Beowulf cluster takes an approach that requires an understanding of Linux kernel level software and hardware construction. Security,

both local and remote, is one of the most important areas that require critical attention. This is extremely important because a compromised machine may create an enormous loss of clock time to repair if a malicious attacker destroys important data and software.

Keep in mind that all of the information provided here about security is constantly evolving. It is important to keep up to date with vulnerabilities that are identified. Use of an information system such as Security Focus website (www.securityfocus.com) can become an invaluable resource at identifying weak points in the security of the Beowulf cluster and serial standalone workstations. Any of the information discussed on security can be applied to both the gateway node in the Beowulf cluster as well as serial workstations. There are two types of security to focus on when securing a Linux machine: local security and remote security.

Local security can be enhanced by locating the computer in a safe room while keeping the area secured under lock and key. The reason local access to the machines should be restricted is because information stored on hard drives is not encrypted. Basically a hard disk drive could be removed from the cluster and transplanted into another Linux machine where the root password could be extracted or changed. A malicious user can also wreak havoc by gaining access locally and causing unneeded problems such as rebooting the machines. This could cause a loss in the current simulation running as well as corrupt data stored on the machines.

There are several effective methods that can be used to enhance remote security for Linux machines. One is to identify and eliminate any clear text password authentications. All password authentications and exchanges should be completed

via encrypted communications. This can be accomplished by utilizing the secure shell (SSH) protocol. Several different implementations are available. It is important to note that encryption is not needed within the cluster and can actually hinder the performance by increasing communication latency.

In addition to utilizing communication encryption, a firewall should be implemented to restrict ports that are available to remote users. Since the development of the Linux 2.4 kernel series, a kernel level firewall via iptables should be implemented that filters network communications defined by rules that the administrator can set. Using the kernel level firewall, all incoming ports should be blocked except for the port on which SSH runs (port 22). SSH allows for both terminal access as well as file transfers to and from the machines.

Within the kernel level firewall, a given packet is received from any communication device (such as eth0, eth1, lo, and others) and is passed to the kernel where a routing decision is made to move the packet along a chain where rules are defined that determines the fate of that packet. The rules defined for the input chain are the most important for securing the machine on which the firewall resides. The input chain gives remote users access to local processes and services such as SSH. Using strict rules, all packets except those coming from desirable hosts can be dropped. The same types of rules can be set for the output chain and forwarding chain. Figure 3.2 shows a schematic of how packets traverse the iptables firewall.

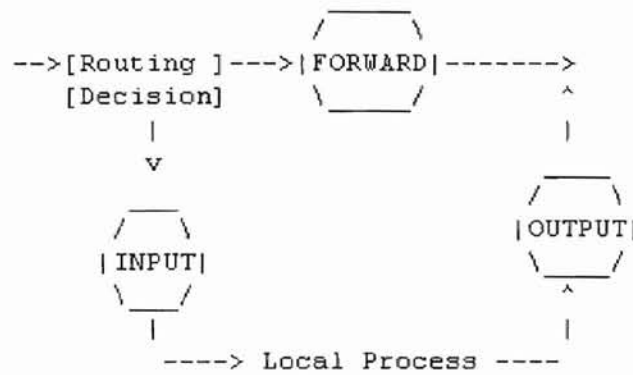


Figure 3.2. Schematic of how packets travel iptables chains

3.3 Hardware Overview

One of the important phases in constructing a Beowulf cluster is the hardware selection. In some cases, such as the MDalpha cluster, the hardware was available prior to cluster assembly. However, when constructing a new cluster, such as MDbeta, care must be exercised in selecting hardware that is both supported by the selected operating system and adequate to perform the desired calculations. Selection of the improper hardware can cause a large amount of increased administration and troubleshooting time.

Selecting hardware to construct a Beowulf cluster can be a daunting task for someone who is unfamiliar with computer hardware. The goal is to build the fastest possible machine while trying to save the most money, which increases the performance-to-cost ratio. This way, more nodes can be purchased. It is possible to purchase a cluster pre-assembled and installed. However, the cost of such clusters is inflated.

The formulation and selection of the cluster was accomplished by considering experience, discussing capabilities with computer parts vendors, and price

comparisons. Price comparisons as a function of computational power were identified for several different machines. After selecting the most powerful machine for the least cost, a test node was purchased to ensure that the available hardware would operate properly on the selected operating system and with the selected MPICH libraries.

Thought of the overall physical layout for the newest MDbeta cluster was a driving factor in the selection of some hardware. Identification of rack mountable hardware was identified as a viable alternative for the ultimate setup for harboring large numbers of nodes. The goal is to increase the size of the cluster to 128 nodes, which would consume an enormous amount of floor space. Rack mountable hardware provides a large density assembly. The Rack mountable hardware also provides an easy way for accessing the nodes for maintenance and repair. Selection of all the hardware and computer parts was done in consideration of the mounting style selected. Two CPUs were selected in order to increase the computational power per node while not increasing the price as much as two separate nodes. Motherboards were selected to have pre execution environment (PXE) while retaining the capability of mounting in the smallest rack mount cases available. Backup power supplies were also implemented to increase the reliability of the machines if brown or black outs occur. For more information regarding the hardware selection process, review the reference provided [26].

Before describing the actual construction phase, background information on the current computational capabilities of the hardware selected is provided. The first, MDalpha, is a DEC alpha 64-bit based Beowulf cluster. Each node in the MDalpha

cluster is running at 500 MHz with access to 512 MB of ECC memory. MDalpha contains 8 processors where each node is capable of 565.88 MFLOPS. These nodes were constructed from Digital Alpha 500au Personal Workstations. The units were originally purchased to run simulations as serial workstations. The original operating system was Tru64 Unix but was moved to Linux as soon as the idea of clustering was explored. The network interconnect is provided by a 24 port HP switch capable of running each port at 100 Mbit full duplex.

The second cluster constructed aimed at fulfilling the user-friendly hardware aspects of this study, MDbeta, is the AMD Athlon based Beowulf cluster with each node running two processors at 1.7 GHz each with access to 1 GB of ECC memory. Each node in the MDbeta cluster is capable of 2108.95 MFLOPS. MDbeta was designed, purchased, and constructed from individual parts that were assembled to form the cluster. Before selecting hardware for the MDbeta cluster, a test machine was purchased and configured to help identify problems with hardware level drivers. All issues with the hardware were worked out and a custom kernel configuration is provided and discussed in the installation section. The network interconnect is provided by a 24 port HP switch capable of running each port at 100 Mbit full duplex.

In both clusters, only one gateway is externally accessible whereas multiple computational nodes, server nodes, and network switches may exist. Note in Figure 3.3, only one computational node, server node, and Intranet switch is shown. However, in the implemented MDbeta cluster, there are multiple computational and server nodes. Another important feature that is shown in the diagram is the console switch. This piece of hardware allows remote logins via the network to any local

console of any device that accepts serial communication. This is extremely useful for remotely troubleshooting problems that occur on the cluster.

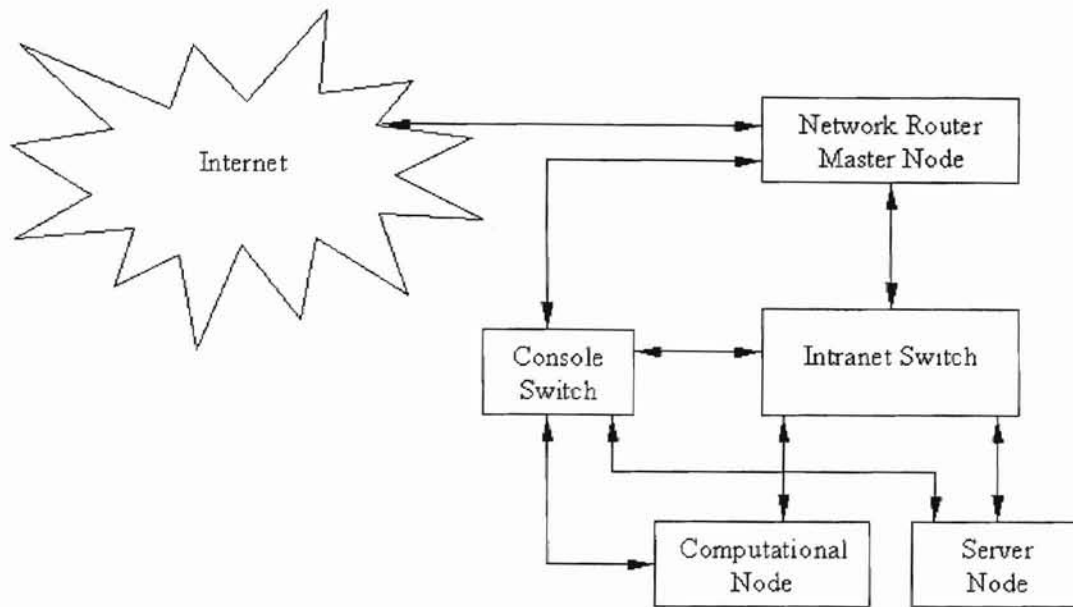


Figure 3.3 – Overview of MDbeta parallel environment

As MDbeta is scaled to include more computational nodes, latency will become an issue. However, preparations have been made to ensure that the 1U cases are capable of accepting an additional 32 or 64-bit PCI card for a fiber or gigabit network upgrade. Interconnection networks may also be explored in conjunction with the current Fast Ethernet network hardware to decrease the network latency between nodes in the cluster.

Another important feature of the MDbeta cluster is that local hard disk drives have been eliminated from each of the computational nodes. This decreases the cost per node as well as facilitates in easing of the administration and configuration. Utilizing this scheme, additional computational nodes can be inserted into the cluster

by simply plugging them into the network switch and editing a few configuration files. This is an extremely important feature that provides scalability to the design.

Booting diskless nodes is accomplished by utilizing PXE, remote-boot application (bpbach), and a Dynamic Host Configuration Protocol (DHCP) server. The nodes are configured to boot up without intervention and mount remote network file system (NFS) shares across the network switch from the file server node. As the number of computational nodes increase, implementing the parallel virtual file system (PVFS) across multiple file-servers may become a requirement to help distribute the network saturation to the file serving node and help increase the file I/O capabilities and decrease file I/O latency.

3.4 Construction

There are two ways of acquiring a Beowulf cluster. The first is to find a vendor and purchase the unit fully assembled with operating system pre-configured. The second option is to assemble the pieces from separate vendors. The second method was chosen because the increased understanding of the inner workings of the cluster coupled with the large savings in the overall cost. Although assembling the cluster from pieces is more difficult and takes more time, details are learned that become pertinent to maintaining and optimizing the cluster at a later date. In addition to the educational experience, the Beowulf cluster designer has complete control over each and every aspect of the cluster nodes.

Assembly of the Beowulf cluster is a repetitive task. All computational nodes are assembled in the same manner and therefore can be done in an assembly line

format. The following outlines the assembly of the cluster nodes. First, all the cases were inspected and motherboards were placed into the cases. CPUs and memory were then placed into the motherboards. Power cables and fans were then connected to the motherboard. Wires were then neatly fastened with zip-ties and organized to allow for easy maintenance and access to the hardware at a later date. Figure 3.4 shows a single computational node assembled. As the picture shows, the computational nodes assembled contained the fewest possible parts. Note that there is no hard disk or floppy drive. This helps to decrease the amount of hardware failures and alleviates hardware maintenance. The height of each computational node is 1U, which is a standard rack size.

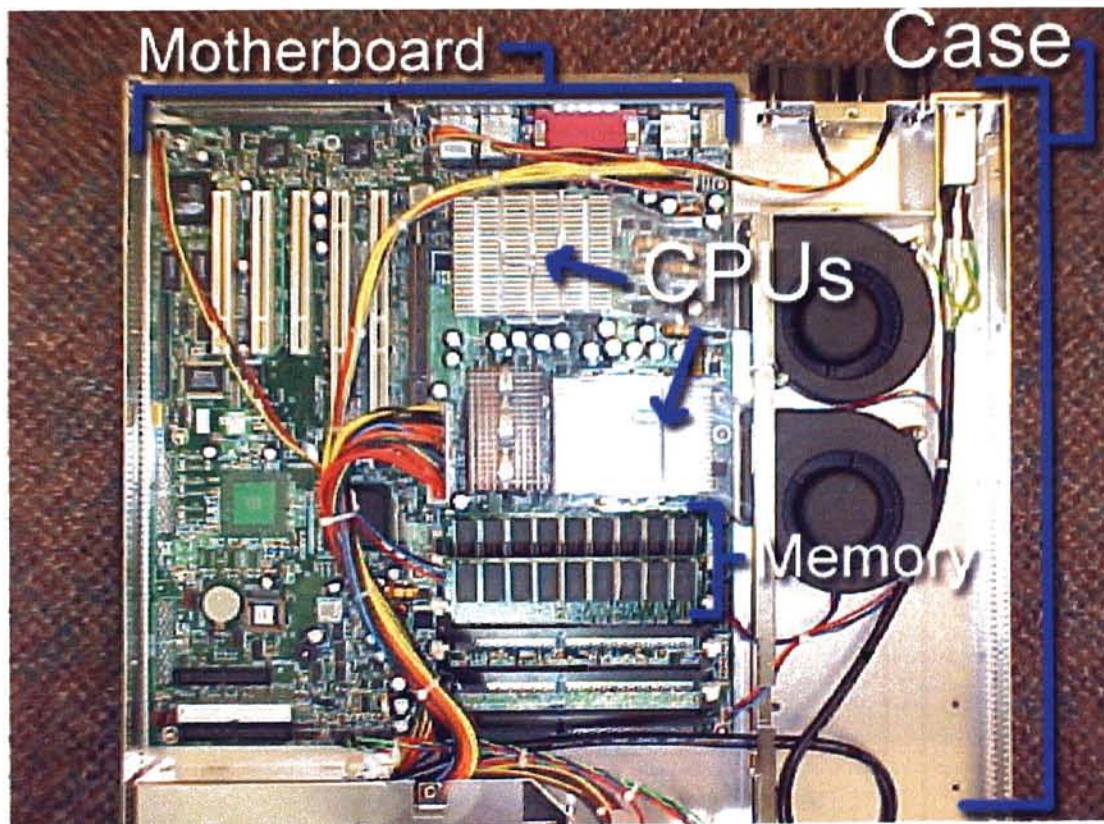


Figure 3.4 – Picture of MDbeta computational node

After assembling the twelve computational nodes, the file-serving node was assembled. This node is identical to a computational node except a 64-bit raid controller card was added along with 800 GB of user drive space. A DVD burner was also installed to provide a cost effective way to archive large simulation files. To accommodate the extra hardware that was installed, a 2U case was selected for this server node. Figure 3.5 is a photograph of a file server node showing the orientation of the motherboard, CPUs, case, raid controller card, hard disk drives, and DVD-R/RW drive in the server node.

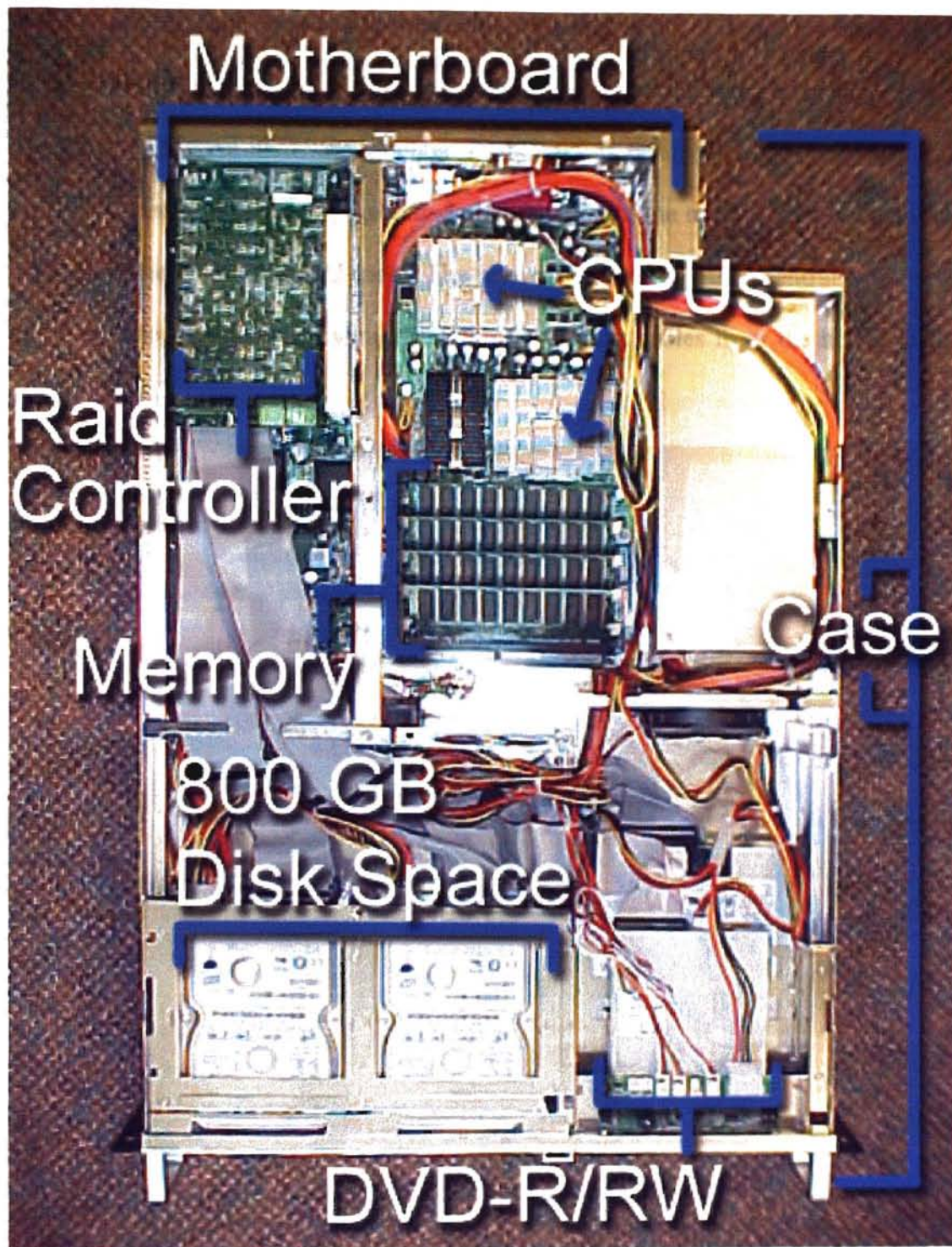


Figure 3.5 – Assembled file-server node

After assembling the hardware in the cases, each of the nodes was then mounted into the rack. The total rack space for the installed nodes and backup power supplies is 20U. All hardware mounted in the rack is fastened to the rack rails with 12-24x1/2 hex cap screws for quick and easy removal of the nodes. Figure 3.6 shows the nodes placed into the 45U rack. The network switch was then centrally mounted in the rack to decrease the average length of the patch cables from the nodes to the switch.

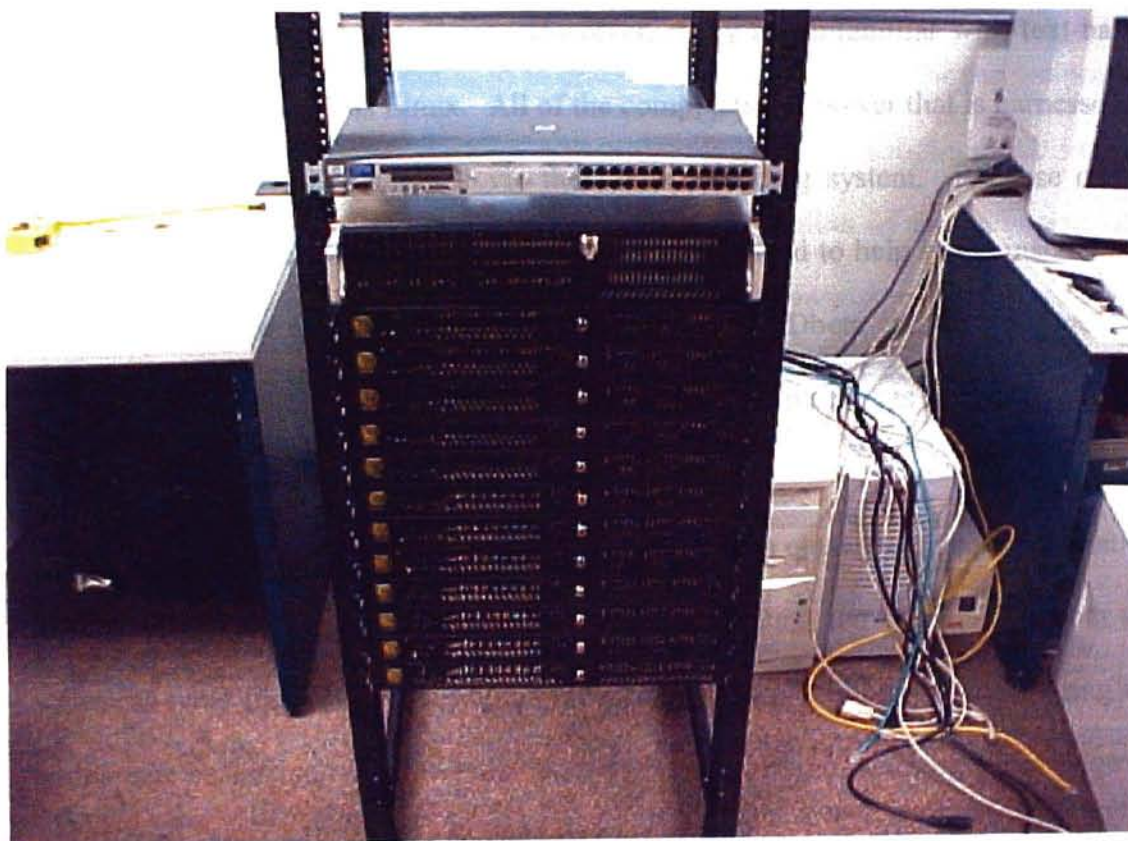


Figure 3.6 – Assembled nodes and mounted in rack

After assembling the computational hardware and placing them into the racks, the back-up power supplies should be installed. Twisted pair, CAT-5 cables were then assembled and each node was connected to the switch. An additional cable was

placed into the second Ethernet port of the gateway node, which doubles as the file serving and DHCP server node. This second network cable is then plugged into the building network jack. The cluster is now completely assembled but is useless until the operating system and libraries are installed.

3.5 Installation and Configuration

Many engineering students are knowledgeable with graphics based applications and operating systems. However, many are unfamiliar with text-based operating systems, such as Linux. All of the computational power that is harnessed in the MDbeta cluster is harnessed via the Linux operating system. Because of the selection of this operating system, Appendix A is provided to help in the installation, configuration, and ultimately the maintenance of the MDbeta cluster. The goal of Appendix A is to give the reader an overview for the main services needed by each of the different types of nodes in the cluster.

In order to facilitate configuration of the MDbeta cluster, a custom installation and maintenance CD was created, named MDiso. MDiso can be found on both the web server and file server. Furthermore, describing the operations for configuring the nodes can be difficult if the reader does not have a basic working knowledge of Linux. Because the reader's knowledge about this subject may vary, a series of screenshots with captions have been provided to guide the reader through the installation process. Appendix A provides a detailed cluster installation and configuration manual.

CHAPTER 4

PROGRAMMING APPROACH FOR MD SIMULATION

4.1. Introduction

Approaching a problem such as creating the software used for the pre-processing, simulations, and post-processing of nanometric cutting and tribology requires careful thought to the needed data structures, format, and operation of the software. A large programming effort is required to undertake even small changes to software that is poorly designed. Therefore a mechanism for maintaining simulation data should be provided. Testing and verification of each piece of software created must be performed in order to ensure that simulations created from the software are valid. Actual development time for the software programming of MD requires many hours of planning, programming, and troubleshooting.

To distribute the programming load over time through different projects and between multiple programmers, modular software should be employed. This approach allows for simple tasks to be organized in a way such that more complicated tasks can be performed. The modular pieces of the MD software suite are divided into three distinct groups: pre-processing, simulation, and post-processing applications. Linking the different modular pieces of code is accomplished through the use of a common file format and functions library, MDbinfmt. The library format provides a means that

standardized the access, creation, and manipulation of simulation data between the three applications.

To distribute the computational load of MD simulations, the programmer should explore parallel processing. Implementation of different algorithms and schemes are needed to identify the most efficient and optimized method for MD simulations. In order to attempt parallel formulation of MD simulation software, key features of the operation and methods for performing parallel MD, also known as N-body simulations, have been identified. These include an overview of the different approaches for organizing the programming of parallel versions of MD software. As discussed before, the user-friendly system must extend beyond the hardware and end-user needs to encompass the programming and implementation of software. This is where the MDbinfmt library can be utilized along with the different algorithms and formulations of parallel MD simulations to help the programmer develop, test, and implement complex parallel versions of MD simulations.

Integrating parallel simulations into the current programming scheme is extremely important to producing code that can be modified by future programmers. The user-friendly system provides the mechanism for running and creating any type of MD simulation software, however, it does not magically create the software. Current compilers are unable to automatically produce parallel code. Even if such a compiler were available, it would most likely be inefficient because of the compilers inability to understand the operational scheme of the designed software much less find an optimized solution. Creating optimized parallel formulation of MD becomes an extremely complicated process when simulation time load balancing and profiling is used to identify

areas of the code or hardware that can be optimized. This optimization must be done manually by the programmer and cannot be accomplished by any current compiling software.

Parallel versions of MD will be similar to the serial versions that currently function within the system. Calculation of the potential, forces, and energies will be unaffected by the fact that the simulation software is running in parallel. More advanced potentials can be employed and implemented into a framework that allows for closer approximations to the actual phenomenon. Extensions to other methods will be possible. Once the parallel framework is complete, modification and addition of new potentials will be trivial once the mathematics for the particular potential have been identified. Implementation and exploration into the different simulation approaches and techniques is needed to allow for atom numbers to be increased to the point where nano scale simulations approach the micro scale.

4.2. MDbinfmt Library

Previously written MD simulation software output multiple files containing coordinate data for different snapshots in time. This output methodology creates a large amount of extra bookkeeping when managing multiple simulations. In addition to this inefficient format, each researcher developed a proprietary version of the output format and animators had to be modified to display specific simulations. Another problem encountered from work completed in the past was that simulations were coded using arrays with static memory allocation. This method uses set amount of memory for every simulation whether large or small. Increasing the simulation size beyond the array limits

called for a change in the software as well as a recompile. Static memory allocation limits simulation size and causes the code to become relatively worthless when expanding the code to handle more advanced simulations. Static memory allocation along with other poor coding techniques has created a large amount of rather useless code.

To help alleviate these problems and provide a mechanism for researchers to create useful and portable code, a library named MDbinfmt was developed. The MDbinfmt library is a powerful tool that aims to hide some of the complexities of the simulation data while giving the programmer access to a defined data structure, described in detail in Appendix C, which can be used for easy manipulation of simulation runtime data. The library will be an essential piece if the MD software suite is ever sold commercially. Using a library format gives the programmer or customer access to important functions without revealing the actual source code.

The MDbinfmt library provides a set format for accessing and storing important information during and after the simulation process. This format consists of packing consecutive binary data that can be accessed later by reading the data in the same way it was written. Another advantage of the single formatted file is that the file can be shared between different applications and accessed by merely calling functions that are available via the library. This ensures that the data is stored and retrieved in the same manner regardless of the specific application. Using the MDbinfmt library, customizable modular software such as the preprocessing software, simulation software, and post processing software can be written without worrying about the complexities of the data storage mechanism for the file format. Simulation software can be designed to work with

the MDbinfmt library regardless of the processing hardware that is available. The library is portable to different operating systems as well as different architectures such as the 32-bit and 64-bit architectures. The library has been implemented in both the 32-bit and 64-bit systems. Details on use of the MDbinfmt library are given in Appendix C. Figure 4.1 is a schematic illustrating how different applications utilize the MDbinfmt library.

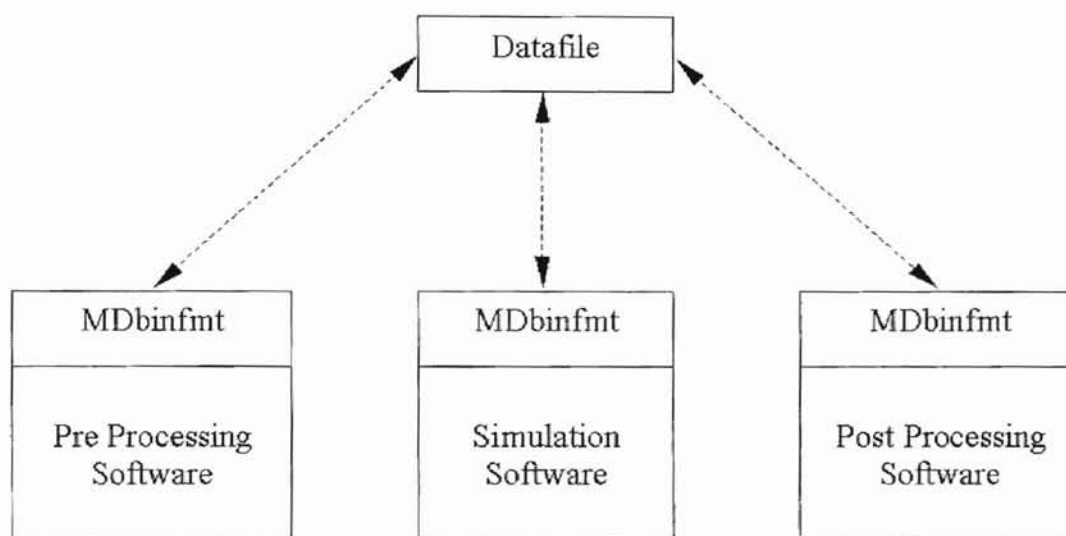


Figure 4.1 – MDbinfmt library overview

The MDbinfmt library allows researchers to interface with simulation data and implement software at a much faster rate than previously observed. Using the old technique for creating simulation software, a researcher developed a complete nanometric cutting code for MC in 4 months. The code was ported over in a fraction of the time. The ported version is compatible with versions of the pre and post processing software which eliminate the need for the particular researcher to maintain the pre and post processing applications for the given simulation software implementation. This idea

extends the programming capabilities by requiring less time dissecting code before implementing and testing different algorithms for solving MD simulations.

One problem exists with this scheme and relates to the different structures that are needed to handle different interatomic potentials within the library format. As new potentials are added, modifications to the MDbinfmt library materials structures will be needed. Modifications will also be required for the structures holding the potential parameters. With the current MDbinfmt, the post-processing applications and animation software is oblivious to these types of changes. However, the pre-processing and simulation software will need to be modified or updated with the newest library when modifications are made.

4.3. Parallel Simulations

There are three important features to be considered when developing parallel MD simulations [26]. These features are *communication*, *load balancing*, and *scaling*. *Communication* is the phenomenon that occurs when messages are passed from one processor to another within the cluster. The communication is delayed by latency in the initiation and acceptance of the communications as well as the throughput speeds. Options for improving communications include channel bonding of multiple network devices as well as faster network implementations, both of which are relatively simple to implement in the MDbeta cluster [40]. The *load balancing* refers to how the simulation is broken up and farmed out to different processors. If the computational work of one computer increases while another node decreases, a part of the simulation should be redistributed in order to optimize the rate at which simulations can be performed. The

ideal parallel simulation case would be to have all of the processors continually working while keeping communication to a minimum. Finally, the *scaling* refers to how well the written code expands when more atoms are used in a simulation and computational nodes are added to the cluster. Many N-body simulations scale as a function of N^2 .

It is important to identify the barriers, such as time step size and geometry, to overcome when programming parallel MD simulations. There are 5 basic strategies for solving MD simulations in parallel [41]. They are cloning, master-slave, replicated data, systolic loops, and domain decomposition. Other techniques have been proposed but can ultimately be derived from one or more of these 5 strategies.

The *cloning strategy* is useful when multiple runs of the same simulation with different conditions need to be executed. This strategy basically starts N independent simulations on N processors. This allows for multiple simulations to be spun off on different processors. This type of usage for the cluster is excellent for small simulations in which many runs with slight variations are desired. Communication between the nodes in this strategy is exclusively between the node and the fileserver.

The *master-slave strategy* uses a master node to allocate and control what the slave nodes compute. One major problem in this strategy is that the master node must communicate and distribute the appropriate information to each of the nodes. This creates a massive communication overhead and requires the master node to store all information for each atom. In some cases, the actual time to compute the specific task on a node processor compared to the time associated with latency in the data communication causes this method to be slower than running the simulation on a single processor. This type of strategy works excellent for systems where shared memory is found such as the

multiprocessor variety. When simulation size approaches the sub micron to micro size, the communication overhead swamps the master node's ability to perform the simulation.

The *replicated data strategy* operates by distributing all atom information to each processor prior to each time step calculation. However, a single processor only performs a certain portion of the calculation. After each time step, a global summation is performed and atom information is again distributed to each of the processors. A major drawback to this method is that memory usage is large because each node in the cluster contains all information for all atoms. For simulations approaching sub micron to micro size, this is unacceptable. However, this method allows the use of complicated functions, such as EAM and MEAM potentials, to be employed for the determining the forces of molecular interactions.

Another strategy called *systolic loops* distributes the atoms evenly and allows the nodes to pass information between the specific nodes. This method can be used with pair potentials but becomes complex when using more complicated potentials, such as the MEAM or EAM potentials.

The final and most promising strategy for use in parallel MD simulations is *domain decomposition*. In this strategy, different geometric spaces are assigned to unique processors. If an atom enters or leaves a geometric region, the atom is moved to the appropriate processor. This is similar to the linked cell method used for the calculation of the bond list, which is implemented into the current simulation software and decreases the computational time for the calculation of the bond list. Domain decomposition shows the most promise for conducting large-scale MD simulations because scaling and compound communication latency is minimized [42].

Developing parallel versions of MD code can be difficult and time consuming. However utilizing the domain decomposition strategy, different geometric areas of the system can be distributed prior to any set of potential calculations. This allows any type of potential to be used, such as the Tersoff, Morse, EAM and MEAM.

CHAPTER 5

IMPLEMENTATION OF USER-FRIENDLY SOFTWARE

5.1. Introduction

Simulation of nanometric cutting and tribology using MD, MC or other equivalent method is a complex operation. However, utilizing tools and software to perform the simulations, those unfamiliar with the mathematics and theory behind the simulation may benefit from performing simulation. Commercially available FEM and FDM software packages do not require that the end user understand the inner workings of the software. Nonetheless, results from these tests allow a large range of end users to study a multitude of areas with the help of a user-friendly interface.

The system implemented in this study aims to provide each user the tools that are needed to create, perform, and analyze nanometric cutting and tribology simulations at the nano level. The software discussed in this chapter was designed to be user-friendly, and easy to use. Each piece of software is classified in one of the three groups: pre-processing, simulation, and post-processing software.

The pre-processing application provides the means for the creation and setup of atomistic style simulations. A graphical application was created to ease the modification of simulation parameters. The simulation software is used to take the input file, perform the simulation, and create the output file. This application is designed to run using the

Linux operating system. This was selected because of the stability it provides as well as the computational clustering that can be built by the Beowulf cluster. The post-processing group provides the means for animating and analyzing results that the simulation software provides. Two important applications were developed for 3D animation and the calculation of the chip flow for the post-processing group. Figure 5.1 shows how the end-user accesses and utilizes the user-friendly system for the simulation of nanometric cutting and tribology.

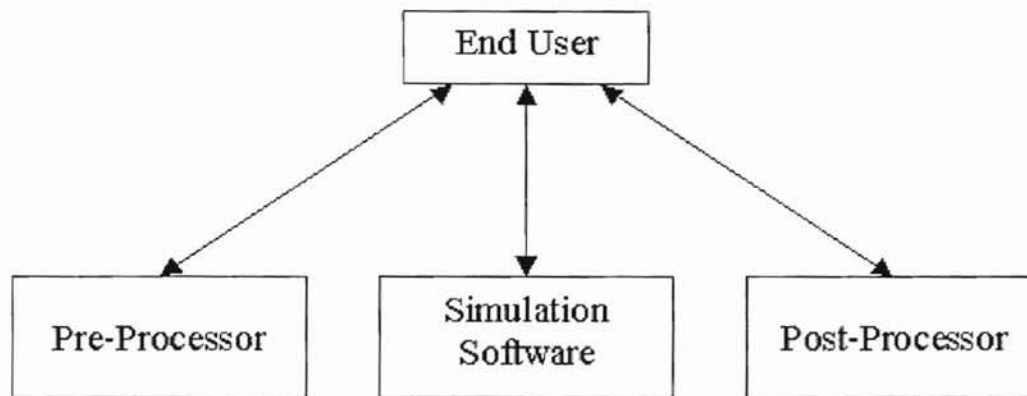


Figure 5.1 – End-user view of simulation software

5.2. Overview of Software Operation

There are three types of applications that are to be used for the simulation of nanometric cutting and tribology processes. They are the pre-processing, simulation, and post-processing software. Each of these three pieces of software is aimed to complete a specific task. They have been separated into three applications to help decrease the programming load and increase the efficiency of the user. The separation of the software into the three groups also helps ease the programming overhead when creating and

modifying aspects of the software applications. In order for the end user to properly operate the software aspects of the user-friendly system, the system operation is described.

The pre-processing application takes the simulation parameters and stores them in a special binary formatted file “.md” that only the simulation and animation software can read. Figure 5.2 shows a diagram of the operation of the MDii pre-processing software.



Figure 5.2 – Diagram of pre-processing operation

Figure 5.3 shows a diagram of the operation of the simulation software. The “.md” block to the left of the arrow is the input file. After starting the simulation, additional frame information containing runtime simulation is appended to the original “.md” file. This allows the simulation files to be animated by the post-processing animations software. The additional frames are denoted in the diagram as the small boxes attached to the original “.md” file.

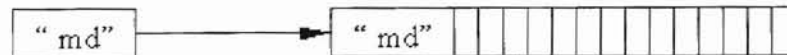


Figure 5.3 – Diagram of simulation operation

The final step is to take the assimilated “.md” file from the simulation software and perform post-processing operations by reading the simulation runtime information.

These operations range from animating to creating videos and snapshots of the simulation process.

5.3. Pre-Processing Software Implementation

The goal of the pre-processing software is to provide an easy-to-use interface that can be used to create input files for new simulations. The simulation then uses these newly created files to store and append important runtime information of the simulation. The actual operations performed when running the pre-processing software, MD input interface (MDii), is to take simulation parameters such as workpiece size, workpiece location, tool size, tool location, crystallographic orientation, various tool angles, material properties, and more.

Several approaches for the input creation software were explored. The first is a console program that is capable of reading an input file with a specific format. This application was named `mdsetupc` and can be found in the MD software suite. The second application that was created was a graphical application that has dialog boxes that can be used to modify the desired parameters for the simulation. After the desired input parameters are selected, click on the Create button and the simulation can be saved in any directory. To make the application backwards compatible with simulations created using the `mdseutpc` application, old-style `mdsetupc` input files can be opened. This allows simulations created using the console setup program to be opened and for simulation input files to be created. Simulations such as oblique and orthogonal cutting can be created using this input interface. Figure 5.4 shows a snapshot of the main dialog box.

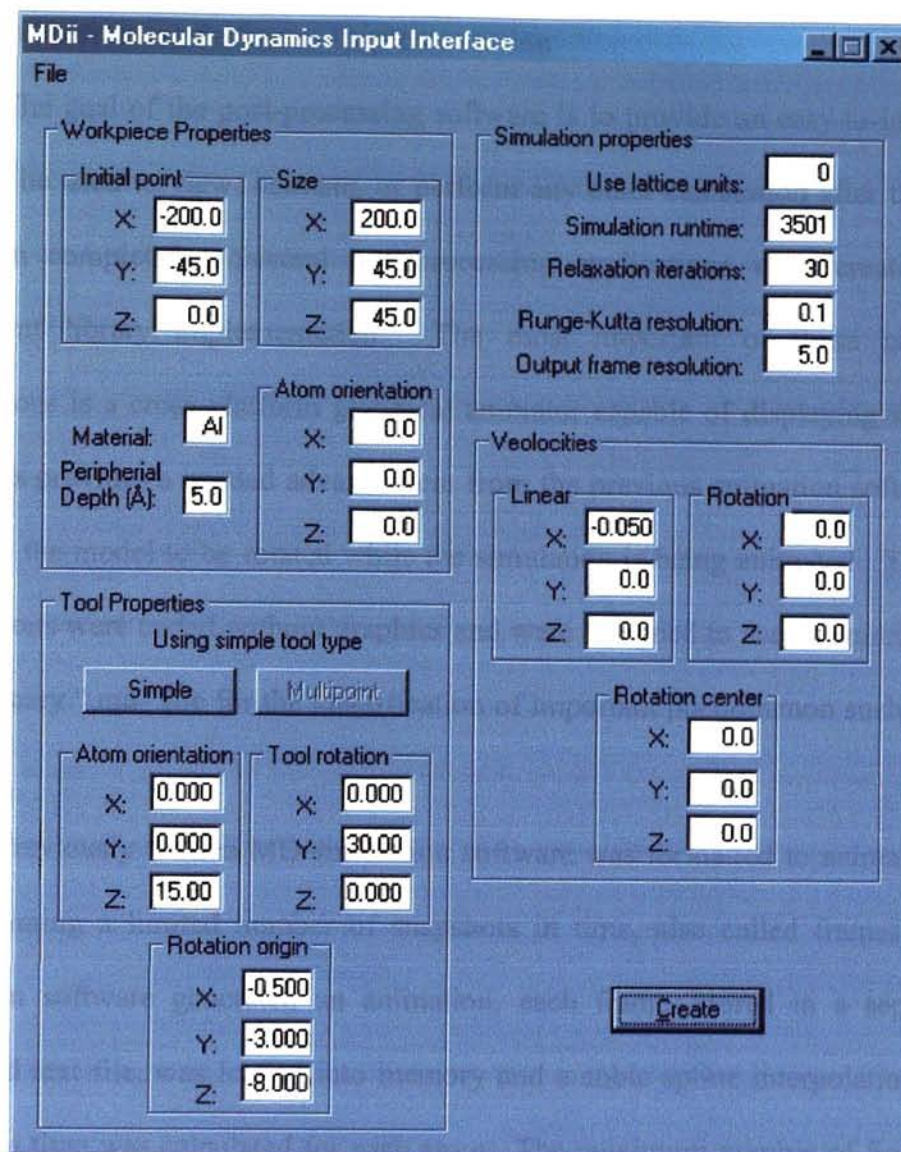


Figure 5.4 – Screenshot of MDii pre-processing software

5.4. Simulation Software Implementation

The current simulation software created runs the desired MD simulations using the Morse potential. Additional potentials and methods can be coded with the use of the MDbinfmt library. Nonetheless, performing the desired simulation is as simple as executing the application with the correct input file created using mdsetupe or MDii. Refer to Appendix B for more information on executing the simulation in Linux.

5.5. Post-Processing Software Implementation

The goal of the post-processing software is to provide an easy-to-use application that can be used to view, animate, or perform any other calculation after the simulation has been completed. Several post-processing applications were created using the MDbinfmt library implementation. The most important of these newly created applications is a cross-platform graphical animator capable of displaying simulations in 3D. This provides a needed advancement from the previous animation software because it allows the model to be rotated while the simulation is being animated. The remaining applications were coded without graphics and were designed to access stored information in the binary “.md” file for the identification of important phenomenon such as chip flow angle.

Previously written MD simulation software was formatted to animate coordinate data spanning a limited number of snapshots in time, also called frames. When the animation software generated an animation, each frame, stored in a separate ASCII formatted text file, was loaded into memory and a cubic spline interpolation curve with respect to time was calculated for each atom. The maximum number of frames that this old style animator was capable of displaying was approximately 20 frames. DirectX was then used to draw a planar representation of the spherical atoms on the screen for different frames to give the appearance that the simulation is animated. This interpolation technique unfortunately hides vibrational and other small movements of the atom trajectories.

The new animator, named MDui, is capable of displaying atom movements with a frame resolution matching that of the differential integration routine in the simulation

software. As the frequency of atom coordinates stored in the “.md” file increases, the smoother the post-processing animation. However, an infinite number of frames cannot be saved because of file size considerations. MDui utilizes OpenGL [43, 44], glui [45], and glut [46, 47] for the graphics and application programming interface (API). OpenGL and glut libraries provide the graphics animation, such as the drawing of atoms and rotations. The glui library provides the cross platform API to control the parameters in the graphics scene.

MDui is capable of displaying multiple crystal information for multiple frames. When the animation file is loaded crystal parameter panels are loaded into the menus of the animation software. Each crystal that is loaded for each frame can be adjusted several different ways. Colors can be applied to each crystal for easy identification of different parts of the simulation. The orientation of the scene within the view port of the animator can then be manipulated to see different phenomenon that occurred during the simulation process.

Controls over the animation are provided on the right hand side of the application and can be used to manipulate the scene. A viewing panel provides a mechanism for advancing the simulation that displays the atom movements. A feature for taking a screenshot of the current animation view port is also provided. Videos, in the form of “.avi” can also be created from a user selected start and end frame. The user can adjust the frame rate of the animation. These output options provides the researcher an easy interface for creating simulations that can be viewed easily in any graphical operating system. Figure 5.5 shows a screenshot of the animator with an nanometric oblique cutting simulation loaded.

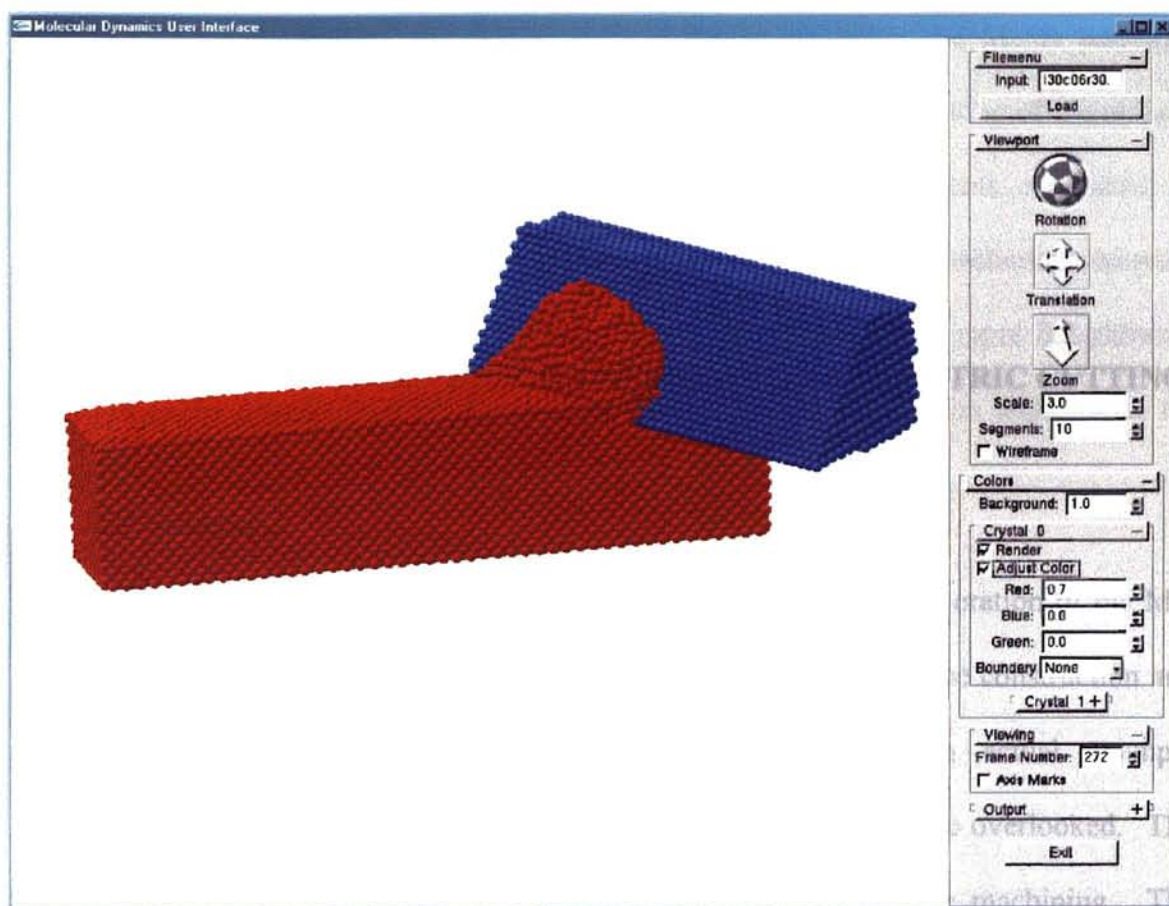


Figure 5.5 – Screenshot of the MDui animation software showing nanometric cutting

CHAPTER 6

APPLICATION OF USER-FRIENDLY SYSTEM TO NANOMETRIC CUTTING

6.1. Introduction

Documentation of key applications and instructions on the operation of the MD software suite has been provided in the preceding chapters to show the construction and operation of the user-friendly system. However, without an actual example implementation, the power of the overall user-friendly system may be overlooked. The example selected for this study was the MD simulation of oblique machining. The process is described by several angles, namely inclination, rake, and the clearance angles. The inclination angle is the angle the tool has been rotated from the velocity vector. Figure 6.1 shows the inclination angle that is created when the tool is rotated with respect to the motion of the workpiece.

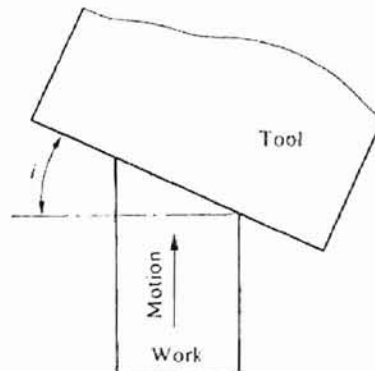


Figure 6.1 – Orientation of the tool creating the inclination angle

MD simulations of nanometric oblique machining using the Morse pair-wise potential were conducted and analyzed. This work, on nanometric cutting of the oblique cutting process, extends the work on orthogonal cutting experiments conducted by previous researchers. In order to analyze the simulations of oblique machining, the post-processing software is required to animate the simulation in 3D. Figure 6.2 shows a diagram of the oblique machining operation, where i is the inclination angle, α_n is the rake angle and α_r is the chip flow angle.

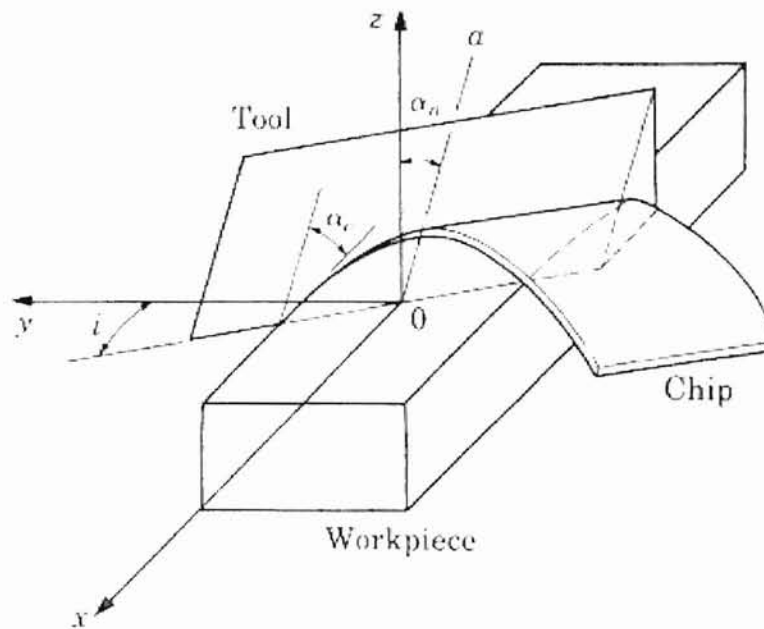


Figure 6.2 – Diagram of the oblique machining operation

Modeling of additional systems requires no effort from the user in terms of creating simulation software. Users only need to utilize the pre-processing applications created within the system by modifying the input files to model additional simulations.

6.2. Simulation Process

Nanometric simulations of oblique machining are performed with conditions given in Table 6.1. Internal computational parameters of the simulation are given in Table 6.2. Setup of the simulations was accomplished by using the MDii pre-processing application described in Chapter 5. Parameter selection of workpiece, tool, velocities of the tool, and frame output information were entered into MDii. Altogether, 35 simulations were setup varying the inclination angle from 0° to 45° while varying the rake angle from -45° to 45° . After organizing the simulation parameters into MDii, clicking on the Create button creates the simulation input file, which was then copied to the Linux workstations to run the simulation. Information on copying the files to the Linux workstation is provided in Appendix B.

Table 6.1 - Parameters used in the oblique simulation investigation

Material		
Workpiece		Al
Tool		(Infinitely hard) W
Cutting Speed		500 m/s
Depth of Cut		4 Å
Bulk Temperature		298 K
Workpiece Geometry		810 Å x 182 Å x 182 Å
Tool Geometry		
Rake	-45°, -30°, -15°, 0°, 15°, 30°, and 45°	
Clearance		6°
Inclination		0°, 15°, 30°, and 45°
Number of Atoms		
Workpiece		23,958 atoms
Tool (range)		4,729 - 22,165 atoms

Table 6.2 – Computational parameters used within the Morse simulation software

Workpiece-Workpiece Potential	Morse
Alpha	1.1646 /Å
D	0.2703 eV
r_{eq}	3.253 Å
Workpiece-Tool Potential	Morse
Alpha	5.14 /Å
D	0.087 eV
r_{eq}	2.05 Å
ω (velocity reset parameter)	0.1047

After creation of the simulation input file is complete, the simulation process begins. The simulation process consists of taking the simulation input parameters entered in the previous step and performing the simulation. Calculation of successive atom positions with respect to time is accomplished with the use of a Runge-Kutta differential integration routine. Values from the forces are used multiple times during this differential integration step to simultaneously calculate the new position of all atoms in the simulation. After a given number of steps, denoted by the output frame resolution in the MDii application, a snapshot of atoms positions are saved into the “.md” file.

Such a simulation, using the input parameters entered for the oblique machining simulations from Table 6.1, takes ~20 hours to complete. This corresponds to a total computational time requirement of ~700 hours or about 29 days for the entire ensemble of simulations. For more information regarding the actual commands executed for this section, refer to Appendix B.

Output files created by the simulation software consists of information containing system kinetic energy, potential energy, total energy, and forces for each differential integration time step and are stored in appropriately named comma separated value “.csv”

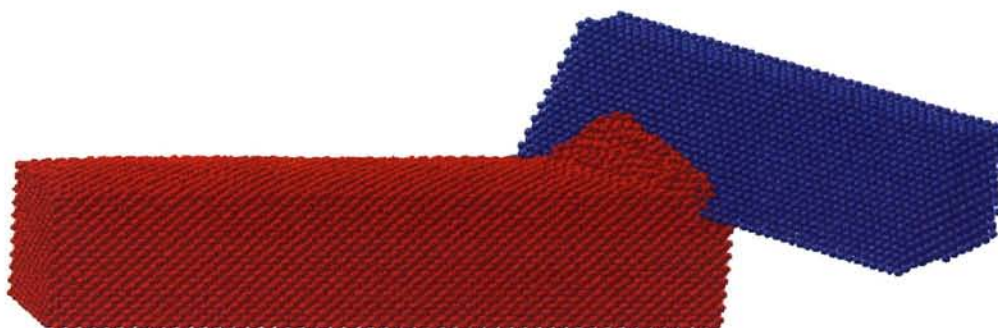
text files. Atom positions are stored after a given number of differential integrations and appended to the end of the input “.md” file after the specified number of differential integration time steps from the input file to help decrease the file size of the coordinate information per simulation. Completed simulations, in addition to those in the process of being completed, can be animated utilizing the post-processing animation software.

6.3. Results

The output files created by the simulation software yields interesting results about the behavior of oblique machining of aluminum at the nano level. Simulations of varying rake and inclination are given as described in Table 6.1. Figure 6.3 shows the various stages of nanometric oblique cutting of single crystal aluminum. Figure 6.4 shows three various important angles that the simulation can be oriented when using the post-processing animation software. Simulations are not limited to these angles but can be rotated in any orientation while in the animation software.



a). frame 20



b). frame 40

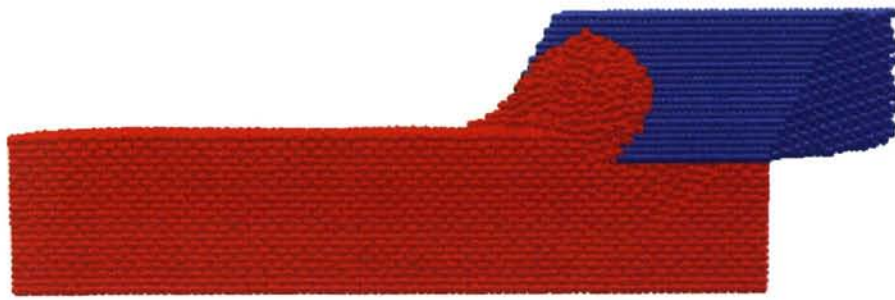


c). frame 60

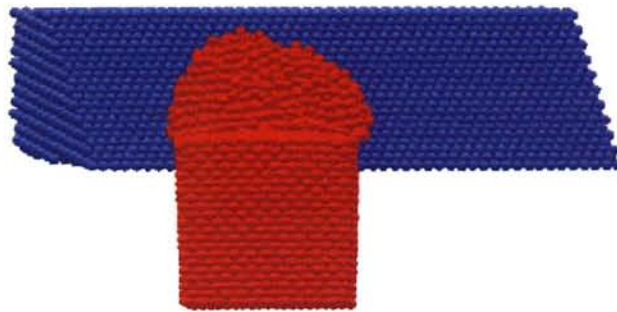


d). frame 123

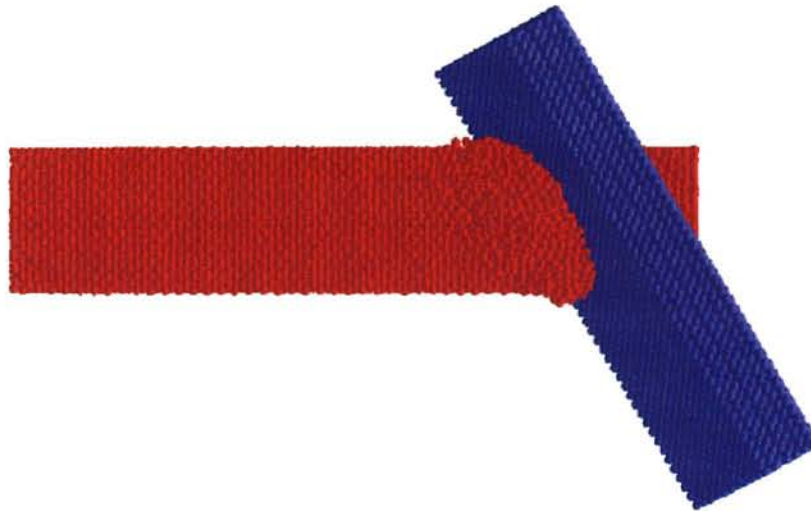
Figure 6.3 – Snapshots showing various stages of oblique nanometric cutting



a). Side view



b). Front view



c). Plan view

Figure 6.4 – Various orientations of the simulation that can be animated using MDui

Chip flow angles were calculated with the help of an additional post-processing application. The chip flow angle is defined as the angle normal to the cutting edge that the chip makes as it moves across the face of the tool [48]. An approximate relationship between the chip flow angle and inclination angle was proposed by Stabler, known as Stabler's rule [49], and is plotted in Figure 6.5 along with the values calculated from MD. Stabler's rule states that the chip flow angle is approximately equal to the angle of inclination. For the values computed by MD, the direction of all atoms in the chip is taken into account when calculating the chip flow angle in an MD simulation.

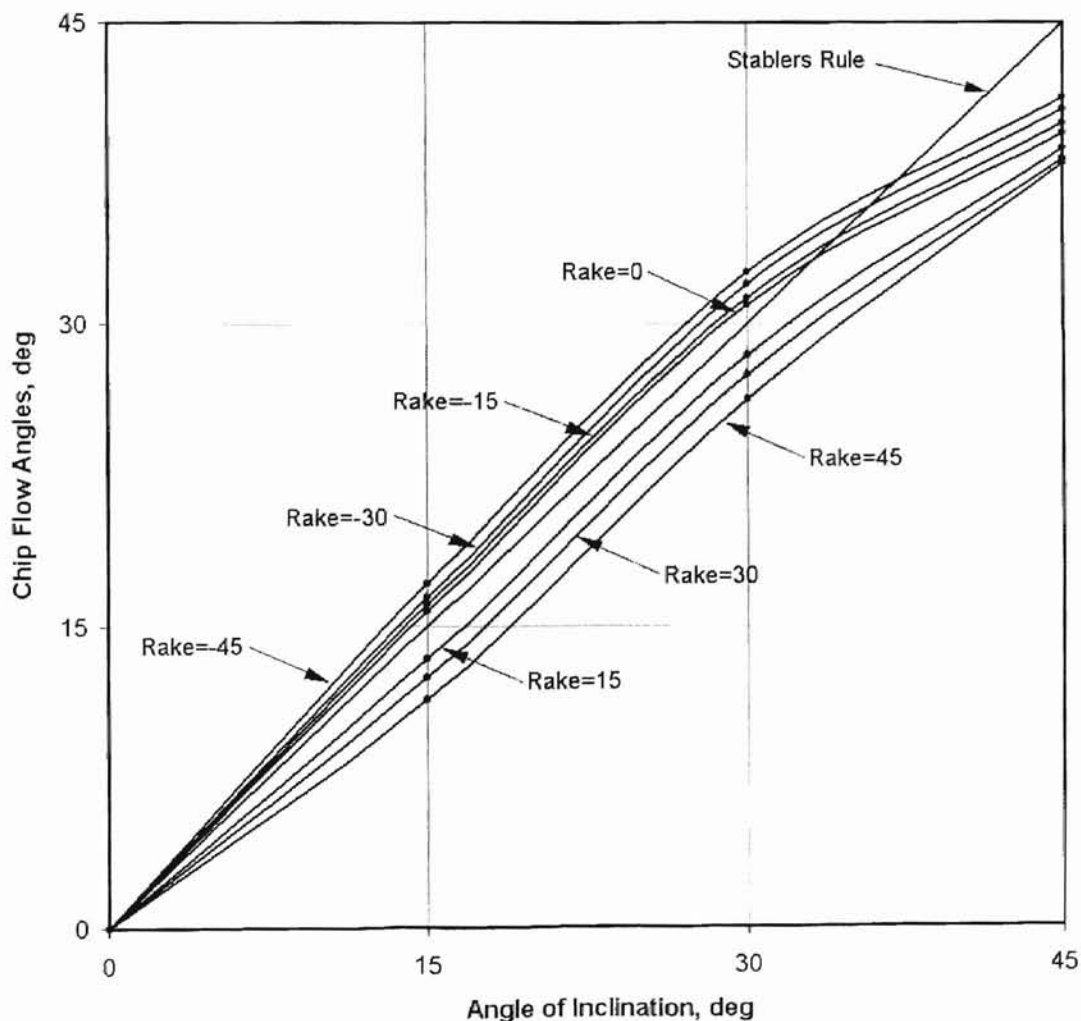


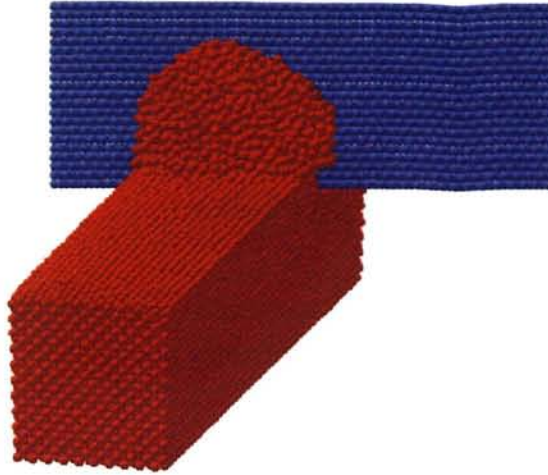
Figure 6.5 – Chip flow angle vs. angle of inclination

Chip flow angle values computed from the MD simulation of oblique machining coincide with those values reported by experimental work conducted by Kececioglu [50] and Brown and Armarego [51]. Furthermore, the values calculated by the simulations are approximated nicely by Stabler's rule up to about 30° . An interesting phenomenon observed is that as the rake angle is decreased from positive to negative at a given angle of inclination, a higher chip flow angle occurs.

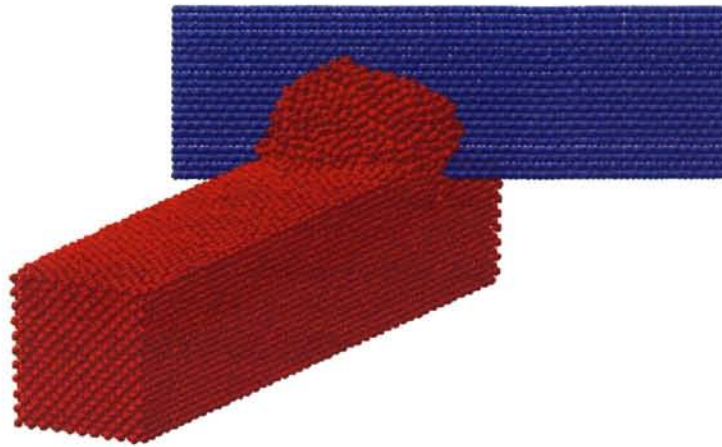
Snapshots of the chip flow angle are given by using the MDui post-processing animation application. The pictures were taken by orienting the camera with the normal to the tool face. This helps in the identification of the chip flow angle and is defined as the angle created by a perpendicular to the tool edge and the chip flow direction. Figure 6.6 shows a sample series of snapshots with a constant rake of 15° , varying the inclination angle from 0° to 45° .



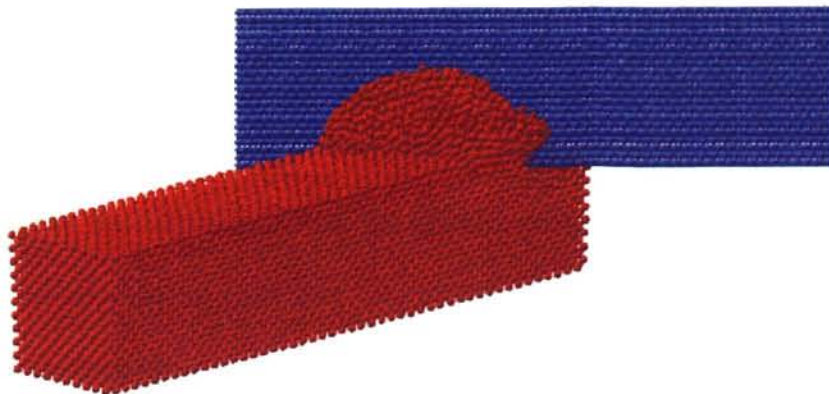
a) inclination=0, rake=15



b) inclination=15, rake=15



c) inclination=30, rake=15



d) inclination=45, rake=15

Figure 6.6 – Variation of the chip flow angle with the angle of inclination

CHAPTER 7

CONCLUSIONS

7.1. General Conclusions

MD simulation of nanometric cutting and tribology provides an important insight into the behavioral phenomenon found in engineering materials that is impossible to acquire by using experimental techniques. Information is provided in this study that ranges from an introduction of MD simulations to the implementation of the software coding of MD simulations. The purpose of this study is to develop a user-friendly MD simulation system for nanometric cutting and tribology. Such an approach helps propagate knowledge transfer from one generation of researchers to the next. This is accomplished by providing instruction and documentation from three perspectives, namely the end-user, programmer, and administrator of the system, on creating, programming, and executing MD simulations.

A user-friendly system was designed and implemented, taking into account the perspectives from the administrator of the system, programmer of the MD software, and end user of the simulation packages and cluster. These three perspectives allow the system to accommodate a larger range of users as well as provide a mechanism for increasing the rate at which simulations can be performed.

From the administrator's point of view, information on construction and maintenance of the Beowulf clustering system is given. Complete documentation is provided to help relieve some of the overhead in maintenance as well as provide an overview of the operation of the hardware, operating system, and simulation software. Identification of the important aspects of enhancing parallel processing is given, as it is the administrator's job to ensure that latency and communication overhead does not increase to the point where the system is inefficient.

For the programmer, information on the MDbinfmt library implementation as well as different approaches of parallel programming are identified. Information regarding the details of the functions and structures provided by the library are discussed. Software programming examples are also given to show the implementation and operation of the MDbinfmt library. Parallel processing algorithms and approaches are discussed but the focus of future parallel software implementations should utilize the domain decomposition method.

The end users of the user-friendly system benefit from the discussion of the applications that are provided in the MD software suite. A set of nanometric oblique machining simulations are given which provide the end user with an example of the usability of the system as well as an overview of the implementation provided. Results from the MD simulation of nanometric oblique machining yields useful insight into the phenomenon of the chip flow during cutting.

Beyond the different aspects to the user-friendly system, several chapters and appendices are devoted to details on utilizing each of the discussed topics. This provides

an available set of procedures and documentation for propagating the transfer of knowledge from one generation of researchers to the next.

7.2. Specific Conclusions

The following is a list of specific conclusions that were obtained in this study. As previously mentioned, the purpose of this study is to create a user-friendly system for MD simulation nanometric cutting and tribology. This was accomplished directly by the identification and implementation of the following list. One important outcome to this study is that it provides a means for developing, performing, creating, and running MD simulations in serial and parallel.

- 1). Two Beowulf class supercomputers were constructed: MDalpha and MDbeta. Both of these clusters were used in the nanometric simulation of oblique cutting. In addition to the power that these clusters harness, each provides a separate implementation and platform for the development of future simulation codes that utilize parallel processing.
- 3). A cross-platform animation software was created to fill the post-processing application need found in this study. This application helps the user to analyze and visualize 3D simulations.
- 4). Post-processing software for the measurement of the chip flow angle was created to measure the average direction of the atom flow within the chip for the oblique nanometric cutting simulations.
- 5). Pre-processing software was created for this study in order to provide an easy to use graphical interface for creating MD simulations.

- 6). A special C programming library was created, named MDbinfmt. This library provides access to data structures and functions that help alleviate some of the headaches when programming atomistic simulations. The library also handles all file operations, which encompasses writing simulation data to the output file.

7.3. Future Work

Application of MD simulations to nanometric cutting and tribology is a relatively new field. Future advances for MD simulations applied to engineering and materials research require diligent exploration by researchers. Several areas have been identified that should be explored in more detail as part of future work.

The creation of a parallel framework is needed that allows any programmed serial interatomic potential to be placed into the simulation software without adjusting the core algorithm. This method may not provide the most optimized parallel code but would be adequate if domain decomposition is employed. Nonetheless, a parallel framework will allow the users to harness the power of the massively parallel Beowulf cluster created in this study to perform MD simulations at a faster rate.

Once the creation of the parallel framework is complete, expansion of the current MDbeta cluster should be considered. Testing should be performed to determine network interface latency and saturation issues. When saturation occurs, upgrade of the communications interconnect should be performed. Channel bonding of the multiple onboard network cards should be explored. Once the cluster size outgrows the channel bonding method of reducing network latency and increasing throughput, new

interconnect technologies such as 10Gb Ethernet, Gigabit Ethernet, Myrinet, and other fiber interconnects should be explored.

Implementation of additional potentials using the MDbinfmt library structure should be performed. At the moment, the only potential that has been coded and tested with the new structure is the Morse potential. Coding and implementation of the MEAM, EAM, Tersoff, MC, and potential using neural networks should be implemented. Also, an extension of MD simulation to include a combined MD/MC method should also be explored to exploit the accuracy of MD with the speed of MC.

Application of the MD software suite should be utilized for studies on other processes such as milling, grinding, tension, and shear. Indentation and scratching should also be explored for frictional studies. MD simulation of each of these processes will provide insight into the behavior and phenomenon of materials at the nano level.

These are just a few of the future goals and research areas that should be focused upon for the advancement of molecular dynamics simulations of nanometric cutting and tribology. Implementation of different algorithms, potentials, and cluster technologies should always be explored to identify new and improved ways to utilize technology to advance the rate at which computational simulations can be performed.

REFERENCES

- [1] Hrenikoff, A., "Solution of the problems in elasticity by the framework method," Transactions of ASME: Journal of Applied Mechanics, 8 (1941) 169-175.
- [2] Courant, R., "Variational methods for the solution of problems of equilibrium and vibration," Bulletin of the American Mathematical Society, 49 (1943) 1-43.
- [3] Argyris, J. H. and S. Kelsey, "Energy theorems and structural analysis," London, Aircraft Engineering, (1955).
- [4] Turner, M. J., R.W Clough, H.C. Martin, and L. J. Topp, "Stiffness and deflection analysis of complex structures," Journal of Aeronautical Science, 23 (1956) 805-824.
- [5] Alder, B., and T. Wainwright, "Studies in molecular dynamics I: general method," Journal of Chemical Physics, 31 (1959) 459.
- [6] Alder, B., T. Wainwright, "Studies in molecular dynamics II: behavior of a small number of elastic spheres," Journal of Chemical Physics, 33 (1960) 1439.
- [7] Raff, L. M., "Molecular dynamics modeling," Lecture Notes, Oklahoma State University, Stillwater, OK, (2001).
- [8] Chapra, S. C. and R. P. Canale, Numerical Methods for Engineers, 3rd Ed, Mass. (1998) 695-715.
- [9] Frisch, M. J., G. W. Trucks, H. B. Schlegel, G. E. Scuseria, M. A. Robb, J. R. Cheeseman, V. G. Zakrzewski, J. A. Montgomery, Jr., R. E. Stratmann, J. C. Burant, S. Dapprich, J. M. Millam, A. D. Daniels, K. N. Kudin, M. C. Strain, O. Farkas, J. Tomasi, V. Barone, M. Cossi, R. Cammi, B. Mennucci, C. Pomelli, C. Adamo, S. Clifford, J. Ochterski, G. A. Petersson, P. Y. Ayala, Q. Cui, K. Morokuma, D. K. Malick, A. D. Rabuck, K. Raghavachari, J. B. Foresman, J. Cioslowski, J. V. Ortiz, B. B. Stefanov, G. Liu, A. Liashenko, P. Piskorz, I. Komaromi, R. Gomperts, R. L. Martin, D. J. Fox, T. Keith, M. A. Al-Laham, C. Y. Peng, A. Nanayakkara, C. Gonzalez, M. Challacombe, P. M. W. Gill, B. Johnson, W. Chen, M. W. Wong, J. L. Andres, C. Gonzalez, M. Head-Gordon, E. S. Replogle, and J. A. Pople, "Gaussian 98, Revision A.6," Gaussian, Inc., Pittsburgh PA, (1998).

- [10] Morse, P. M. "Diatomic molecules according to the wave mechanics II vibrational levels," *Physics Review*, 34 (1929) 54-57.
- [11] Lennard-Jones, J. E. "Forces between atoms and ions," *Proceedings of the Royal Society*, 109 (1925) 584.
- [12] Lennard-Jones, J. E. and B. M. Dent, "Forces between atoms and ions," *Proceedings of the Royal Society*. 112 (1926) 230-234.
- [13] Tersoff, J. "New empirical approach for the structure and energy of covalent systems," *Physical Review B*. (1988) 6991-7000.
- [14] Daw, M.S. and M. I. Baskes, "Embedded atom method: derivation and application to impurities, surfaces, and other defects in metals," *Physics Review B*, 29 (1983) 6443.
- [15] Baskes, M. I., J. S. Nelson, and A. F. Wright, "Semiempirical modified embedded atom potentials for silicon and germanium," *Physics Review B*, 40 (1989) 6085.
- [16] Baskes, M. I., "Modified embedded atom potentials for cubic materials and impurities," *Physics Review B*, 46 (1990) 2727.
- [17] Sobol, L. *A Primer for the Monte Carlo Method*, CRC Press, (1994).
- [18] Hagan, M. T. and H. B. Demuth, *Neural network design*, PWS Publishing Company, (1996).
- [19] Riley, M. E., M. E. Coltrin, D. J. Diestler, "A velocity reset method of simulating thermal motion and damping in gas- solid collisions," *Journal of Chemical Physics*, 88 (1988) 5934-5942.
- [20] Frenkel, D. and B. Smit. *Understanding Molecular Simulation*, Academic Press, Amsterdam, (1996).
- [21] Streett, W. B. and D. J. Tildesley, "Multiple time-step methods in molecular dynamics," *Molecular Physics*, 35 (1978) 639-648.
- [22] Nakano, A., P. Vashishta, and R. K. Kalia, "Parallel multiple-time-step molecular dynamics with three-body interaction," *Computer Physics Communication*, 77 (1993) 303-312.
- [23] Slaets, F. W. and G. Travieso, "Parallel computing: a case study," *Computer Physics Communication*, 56 (1989) 63.

- [24] Stokes, D. Parallel Processing of Molecular Dynamics Simulation in a Distributed Computing Environment and Application to the Modified Embedded Atom Method and Nanoindentation, Oklahoma State University, Masters Thesis, Dec. 2000.
- [25] Srinivasan, S. G., I. Ashok, H. Jonsson, G. Kalonji, and J. Zahorjan, "Parallel short-range molecular dynamics using the *adhara* runtime system," Computer Physics Communication, 102 (1997) 28.
- [26] Sterling T. L., J. Salmon, D. J. Becker, and D. F. Savarese, How to Build a Beowulf: A Guide to the Implementation and Application of PC Clusters, 2nd Ed. MIT Press, (1999).
- [27] Belak, J. and I. F. Stower, "A molecular dynamics model of the orthogonal cutting process," ASPE Annual Conference, (1990) 76.
- [28] Komanduri, R., N. Chandrasekaran, and L. M. Raff, "Molecular dynamics (MD) simulation of atomic scale friction," Physics Review B, 61 (2000) 14007-14019.
- [29] Komanduri, R., N. Chandrasekaran, and L. M. Raff, "Molecular dynamics (MD) simulation of uniaxial tension of some single crystal cubic metals at nanolevel," International Journal of Mechanical Science, 43 (2001) 2237-2260.
- [30] Komanduri, R., N. Chandrasekaran, and L. M. Raff, "MD simulation of indentation and scratching of single crystal aluminum," Wear, 240 (2000) 113-143.
- [31] Komanduri, R., Chandrasekaran, N. and Raff, L. M. MD Simulation of Nanometric Cutting of Silicon. Philosophical Magazine. 2001.
- [32] Chandrasekaran, N. Length Restricted Molecular Dynamics (LRMD) Simulation of Nanometric Cutting, Masters Thesis, Oklahoma State University, (1997).
- [33] Komanduri, R., N. Chandrasekaran, and L. M. Raff, "Effect of tool geometry in nanometric cutting: a molecular dynamics simulation approach," 219 (1998) 84-97.
- [34] Komanduri, R., N. Chandrasekaran, and L. M. Raff, "MD simulation of exit failure in nanometric cutting," Material Science Engineering A. 311 (2001) 1-12.
- [35] Komanduri, R., and R. Stewart, "MD simulation of the modeling of grain boundaries in nanometric cutting," unpublished work, (1998).
- [36] Komanduri, R., N. Chandrasekaran, and L. M. Raff, "MD simulation of nanometric cutting of single crystal aluminum: effect of crystal orientation and direction of cutting," Wear, 242 (2000) 60-88.

- [37] Gropp, W., E. Lusk, and A. Skjellum, Using MPI: Portable Parallel Programming with the Message Passing Interface, 2nd Ed, MIT Press, (1999).
- [38] Snir, M., S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, MPI – The Complete Reference: Volume 1, The MPI Core, 2nd Ed, MIT Press, (1999).
- [39] Gropp, W., S. Huss-Lederman, A. Lumsdaine, E. Lusk, B. Nitzberg, W. Saphir, and M. Snir, MPI – The Complete Reference: Volume 2, The MPI Extensions, 1st Ed. MIT Press (1999).
- [40] Becker, D. <http://www.beowulf.org/software/bonding.html>, Beowulf Ethernet Channel Bonding
- [41] Smith, B. “Molecular dynamics on parallel computers,” HPCI Seminar in MD Techniques on MPP Platforms, (1996).
- [42] Srinivasan, S. G., I. Ashok, H. Jonsson, G. Kalonji, and J. Zahorjan, “Dynamic domain decomposition parallel molecular dynamics,” Computer Physics Communications, 102 (1997) 44.
- [43] Woo, M., N. Jackie, T. Davis, and D. Schreiner, OpenGL Programming Guide: Redbook, 3rd Ed, Addison-Wesley, (1997).
- [44] Wright, R. S., and M. Sweet, OpenGL Super Bible 2nd Ed, Waite Group Press, (1999).
- [45] Rademacher, P. <http://www.cs.unc.edu/~rademach/glui/>, Glui User Interface Library v2.1.
- [46] Kilgard, M. Original implementation of the Glut Library.
- [47] Robbins, N. <http://www.xmission.com/~nate/glut.html>, Glut for Windows Library.
- [48] Shaw, M. C. Metal Cutting Principles, Oxford Science Publications. (1997) 429.
- [49] Stabler, G. V. “The fundamental geometry of cutting tools,” Proceedings of the Institutional Mechanical Engineers, 165 (1951) 14.
- [50] Kececioglu, D. “Force components, chip geometry, and specific cutting energy in orthogonal and oblique machining of SAE 1015 steel,” Transactions of the ASME, January, (1958) 149-157.
- [51] Brown, R. H. and E. J. A. Armarego, “Oblique machining with a single cutting edge,” (1964).

APPENDIX A

CLUSTER INSTALLATION AND CONFIGURATION MANUAL

A1. Introduction

Many engineering students are knowledgeable with graphics-based applications and operating systems. However, some are unfamiliar with text-based operating systems, such as Linux. In this investigation, the computational power that is harnessed in the MDbeta is accessed via the Linux operating system. Because of the selection of this operating system, this appendix is prepared to facilitate in the installation, configuration, and ultimately the maintenance of the MDbeta cluster. The goal is to provide the reader with an overview for the main services needed for each of the different types of nodes in the cluster.

To facilitate configuration of the MDbeta cluster, a custom installation and maintenance CD was created, named MDiso. MDiso can be found on both the md web server and mdf.mae file server. Furthermore, describing the operations for configuring the nodes can be difficult if the reader does not have a basic working knowledge of Linux. A series of screenshots with captions have been provided to guide the reader through the installation process. Keep in mind that each node in the cluster can be custom configured. This means that as the cluster scales in size, cluster

ervices may be moved to dedicated nodes in order to handle the added increase in usage from the additional nodes.

As described in Chapter 4 of this document, there are 2 types of nodes: computational nodes and server nodes. This section of the document is organized by basic services needed instead of explicitly discussing about computational nodes and server nodes. This allows the reader to further expand the cluster and distribute the core server services over multiple nodes.

All nodes in the cluster should be running the following mandatory cluster communication services via inetd:

- rlogin (port 513) for remote login capabilities
- rsh (port 514) for remote execution of commands

Each additional server node added to the cluster should be booted with hard disk drives. Information in Section A3 has been provided to help with the base Linux installation for nodes booted from local hard disks. Additional services run via the server node include the following:

- ssh (port 22) on the gateway node along with iptables
- dhcpd (port 67) to assign IP address to each node in the cluster
- etftpd (port 69) to transfer the kernel for the remote node to boot
- nfsd (port 2049) allows for drives to be mounted remotely

More information on the services as well as further details for the configuration of the Linux operating system can be found on the www.linuxdoc.org

website. There is a plethora of information located at [this large archive](#) and should be used whenever questions arise that are not answered in this investigation.

A2. Enabling Mandatory Clustering Services

Each node in the cluster must be running core services that allow for logins and commands to be executed remotely. If Section A4 is followed carefully, no external connections from outside the cluster can be made to these insecure services. These services, provided by `rlogin` and `rsh`, are run from the `inetd` super daemon server on all nodes. To enable these services, add the correct lines in the `inetd.conf` and rehash the daemon by executing **`kill -HUP `pidof inetd``**. By default, this server will be started through the startup scripts at boot on all nodes in the cluster. The `inetd.conf` file must be edited and two lines should be modified. The following lines are required in the `inetd.conf` on every node:

```
shell stream tcp  nowait root  /usr/sbin/tcpd  in.rshd -L -h -a
login stream tcp  nowait root  /usr/sbin/tcpd  in.rlogind -a -L
```

A3. Slackware Distribution Base Installation

When performing an installation, there are a series of screens that require input. Many of the screens are generally self-explanatory. However, an explanation has been provided for each screen that requires input during the installation. Bold typeset words are commands that can be executed at the prompt, if available, or by opening a new console and swapping to that console via **`alt-f1`** to **`alt-f5`**. Also, any standard commands can be executed while in the virtual terminals. During the

installation, the desired selection along with the desired action should be highlighted before pressing enter. Be careful when making selection during the installation. Any errors at this point require termination of the startup script by executing **ctrl-c**. Figure A1 is the first screen that appears when the installation CD is booted at startup. Press enter to begin the installation process.

```
Welcome to the Slackware Linux installation disk! (version 8.1.0)
##### IMPORTANT! READ THE INFORMATION BELOW CAREFULLY. #####

- You will need one or more partitions of type 'Linux native' prepared. It is
  also recommended that you create a swap partition (type 'Linux swap') prior
  to installation. For more information, run 'setup' and read the help file.

- If you're having problems that you think might be related to low memory (this
  is possible on machines with 16 or less megabytes of system memory), you can
  try activating a swap partition before you run setup. After making a swap
  partition (type 82) with cfdisk or fdisk, activate it like this:
    mkswap /dev/<partition> ; swapon /dev/<partition>

- Once you have prepared the disk partitions for Linux, type 'setup' to begin
  the installation process.

- If you do not have a color monitor, type: TERM=vt100
  before you start 'setup'.

You may now login as 'root'.

Custom serial slackware install login: root_
```

Figure A1 – Installation CD login screen

Login and execute **fdisk** to set up the partition table on the hard disk drives. Figure A2 shows a sample partition table. After the partition tables have been set up on the hard disk drives, the **setup** script should be executed. Select swap space=ram.

```

root@slackware:~# fdisk /dev/hda

The number of cylinders for this disk is set to 2495.
There is nothing wrong with that, but this is larger than 1024,
and could in certain setups cause problems with:
1) software that runs at boot time (e.g., old versions of LILO)
2) booting and partitioning software from other OSs
   (e.g., DOS FDISK, OS/2 FDISK)

Command (m for help): p

Disk /dev/hda: 255 heads, 63 sectors, 2495 cylinders
Units = cylinders of 16065 * 512 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/hda1            1          192     1542289+   83  Linux
/dev/hda2           193          323     1052257+   82  Linux swap
/dev/hda3           324         2495     17446590   83  Linux

Command (m for help): q
root@slackware:~# setup_

```

Figure A2 – Sample partition table and setup script execution

Highlight the addswap option and continue as shown in Figure A3. It is a good idea that this swap space be enabled when performing the installation. If a swap partition is found, a dialog box appears, such as Figure A4 shows.

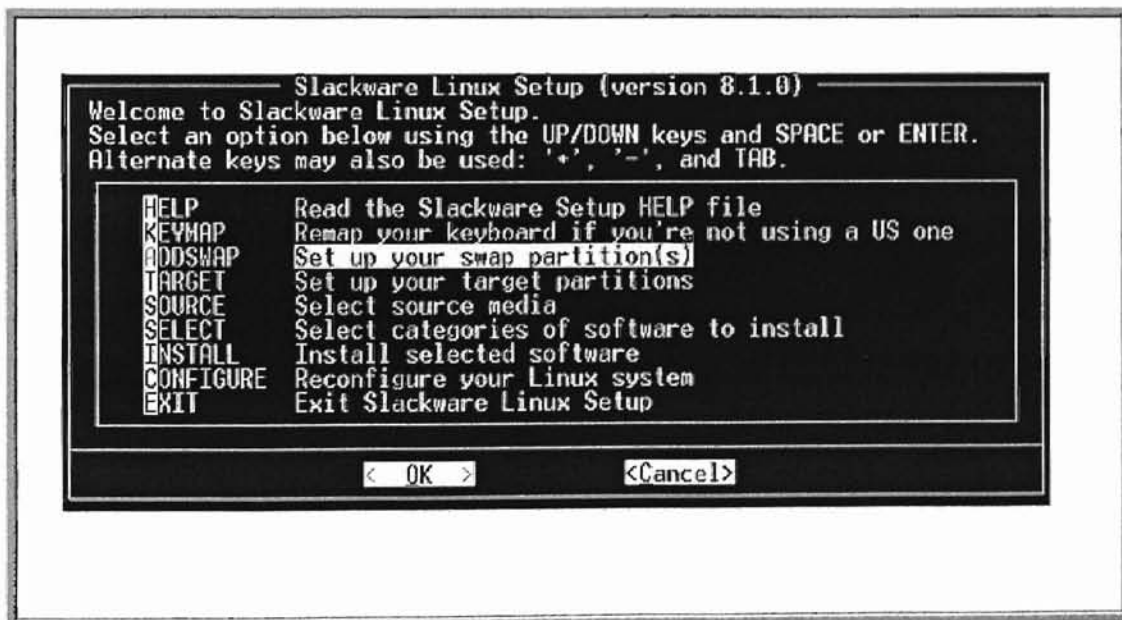


Figure A3 – Setup screen with swap option highlighted

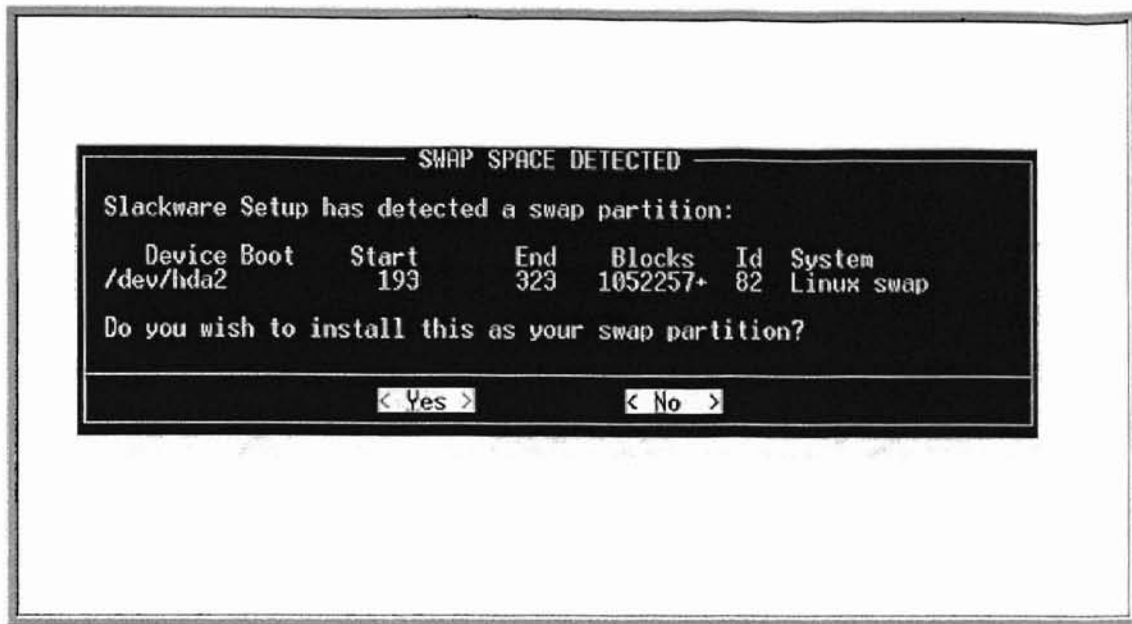


Figure A4 – Swap space detected dialog box

After enabling the selected swap space, as shown in Figure A5, the partitions defined in the partition table need to be formatted. The first partition is usually selected and mounted as the '/' or root partition, as shown in Figure A6.

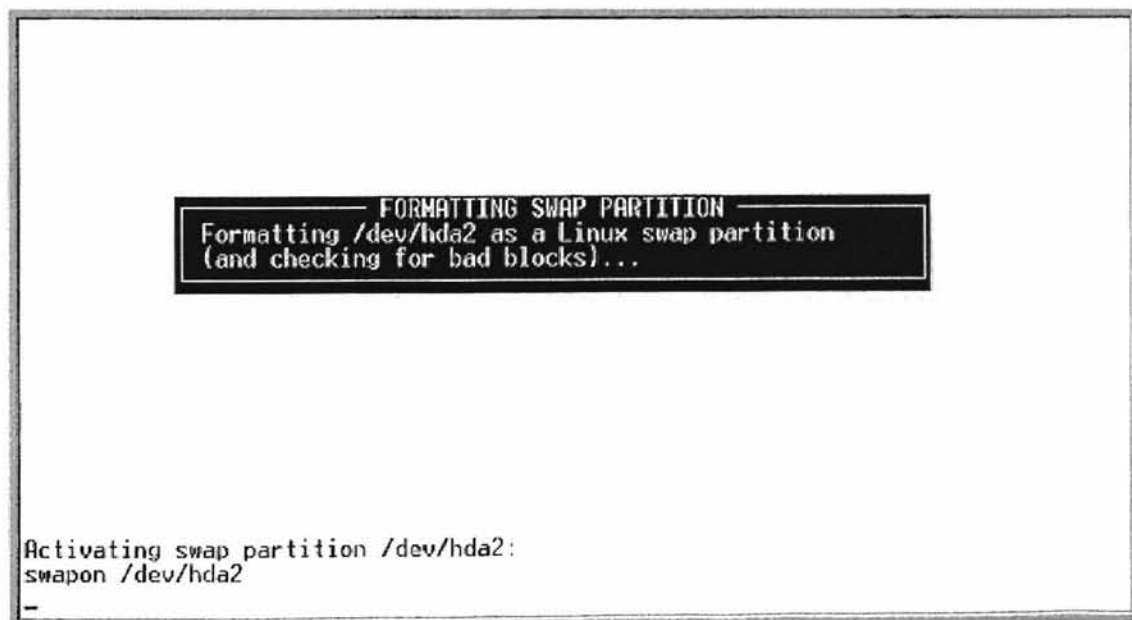


Figure A5 – Activation of the swap space dialog



Figure A6 – Root partition selection dialog

Before mounting the selected partition, it must be formatted. Figure A7 shows the dialog box for the partition formatting. If you are unsure of the integrity of the drive, select the check option; otherwise, the format option will suffice. Figure A8 shows the inode options. Select the default of 4096 bytes.

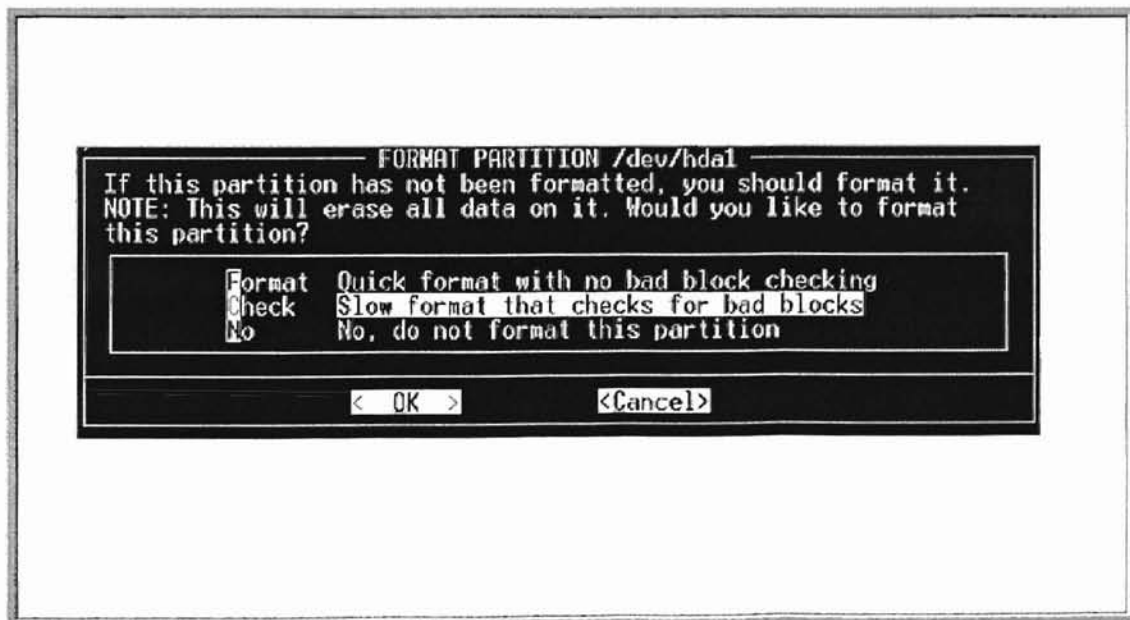


Figure A7 – Partition formatting selection dialog

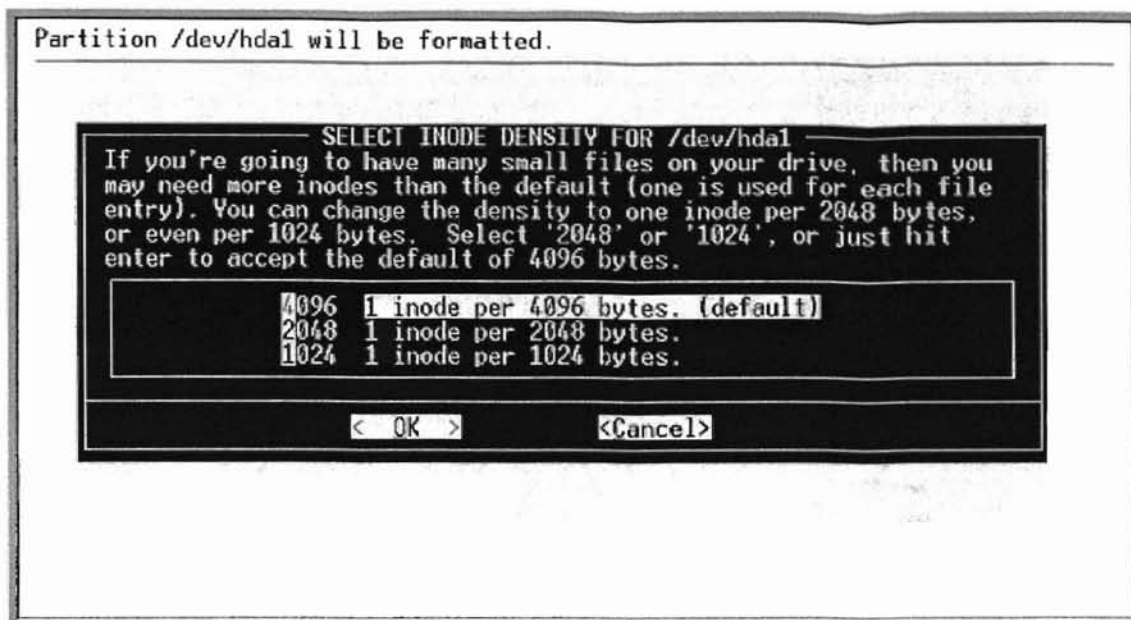


Figure A8 – Inode density selection dialog

After formatting the root partition, the remaining partitions need to be formatted and mounted in the appropriate locations. For the example given, the home drive space needs to be formatted. Figure A9 shows the remaining options in the Linux partition menu. All server nodes should still mount the home partitions via the nfs server. If the node that is being configured is the nfs server, then mounting a large partition for the user directories is needed.

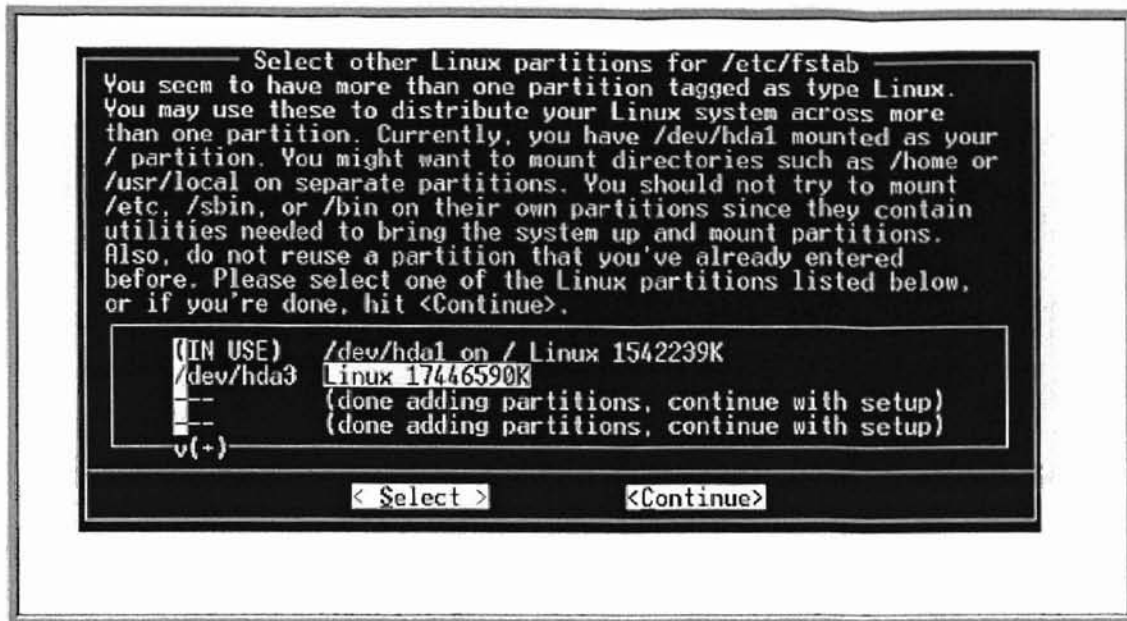


Figure A9 – Remaining options in the Linux partition menu dialog

After formatting and mounting the partitions, the file system tab file (fstab) will be displayed on the screen. Figure A10 shows a sample dialog. Review it to make sure that the partitions you selected were mounted at the correct location. After reviewing the fstab, continue the install from the Slackware CD-ROM, as shown in Figure A11.

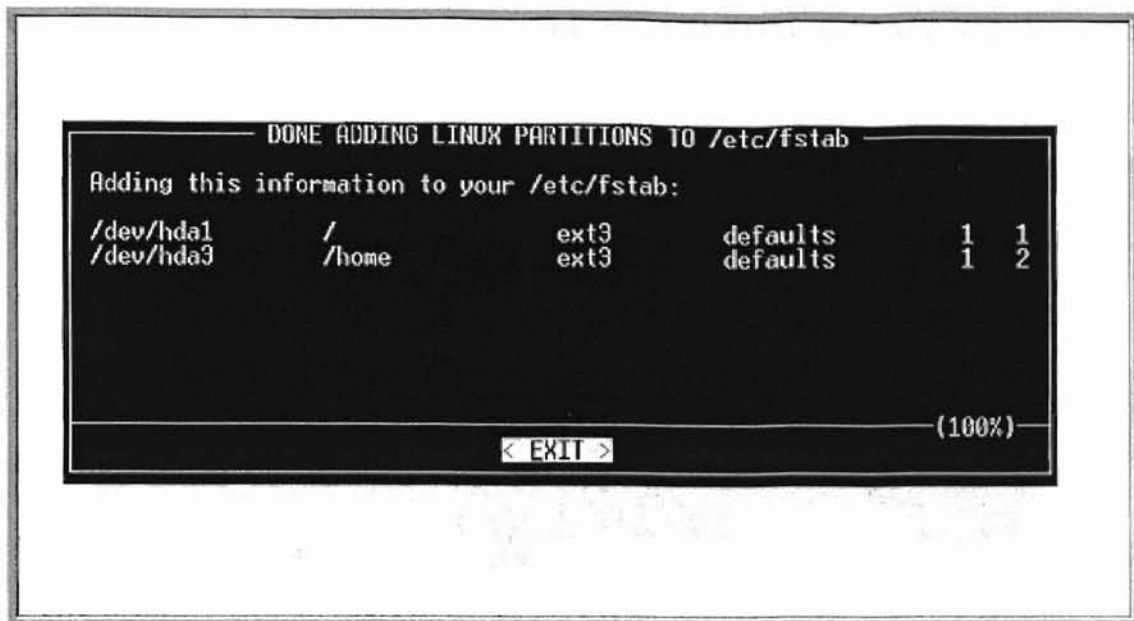


Figure A10 –File system tab file displayed for review dialog

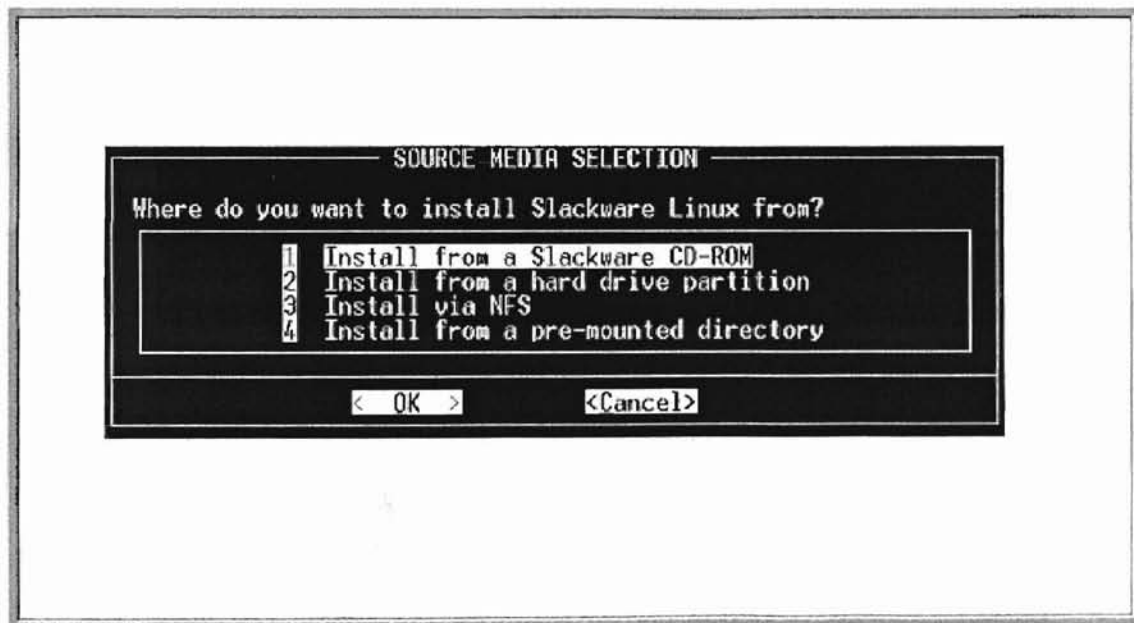


Figure A11 – Source media selection dialog

To install from the CD, you must know the device name. If you are unsure of the device name, you may determine this by the **dmesg** in a virtual console.

Otherwise select the auto option to search for the CD-ROM, as Figure A12 shows. If a drive is identified, it is mounted, as shown in Figure A13.

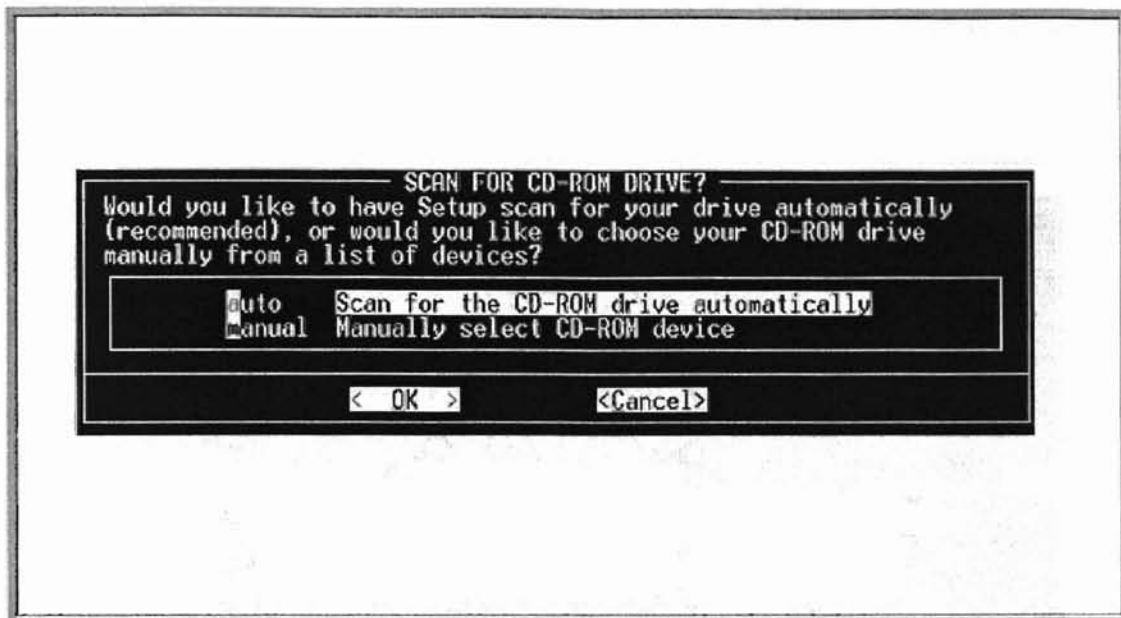


Figure A12 – Automatic CD-ROM detection dialog

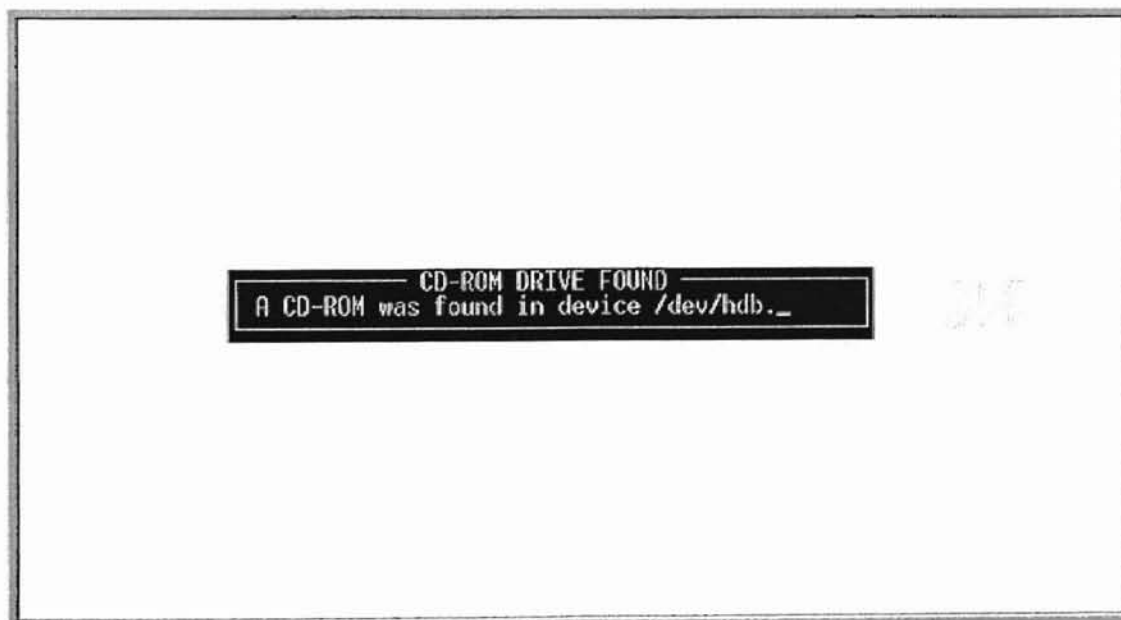


Figure A13 – Automatic mounting of CD-ROM dialog

Continuing with the installation process requires that the separate Linux packages be selected. To alleviate this repetitive task for multiple configurations, custom tagfiles were created. You may select all the package series or leave the defaults active as shown in Figure A14. However, do not exclude any default series.

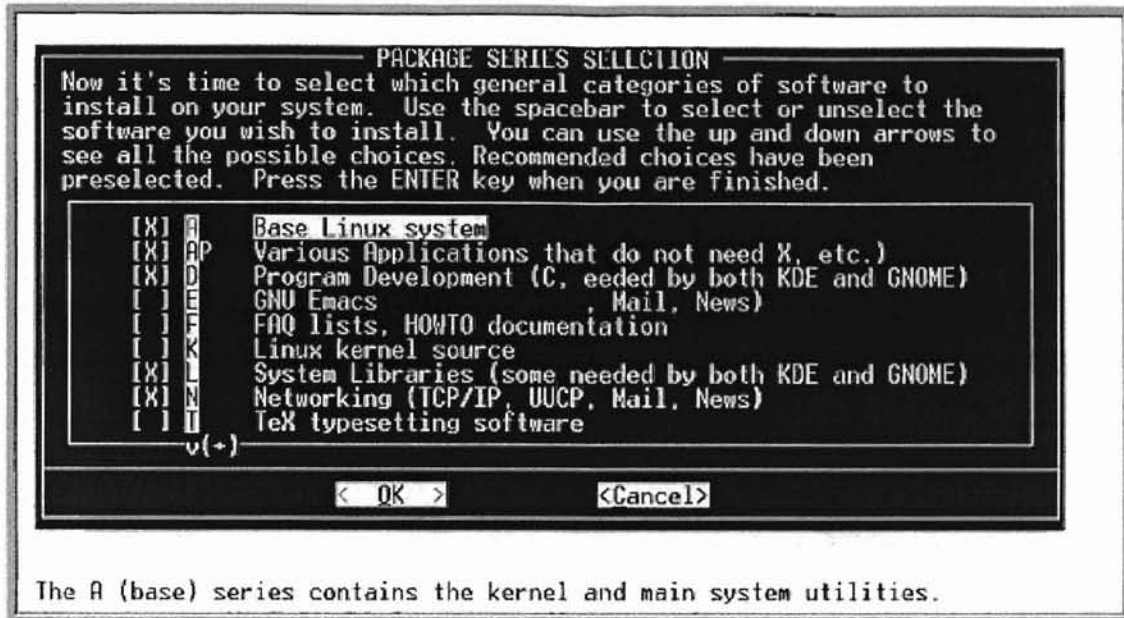
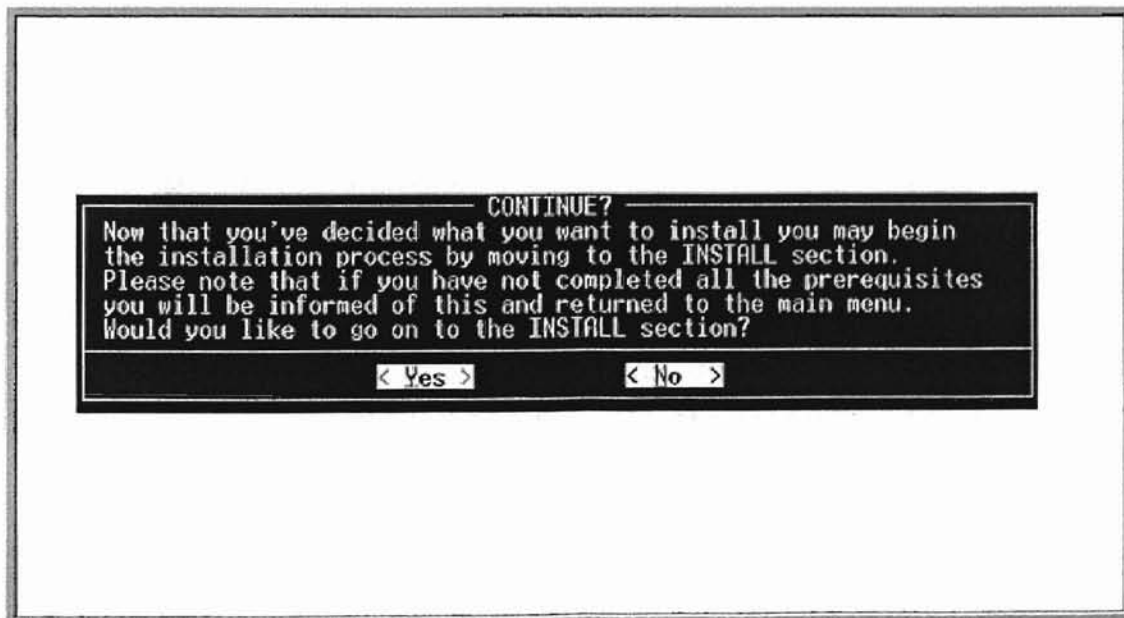


Figure A14 – Package series selection dialog



Custom tagfiles were created in order to automate the package selection step, which can be a daunting task for someone who is unfamiliar with the different software packages. Select custom as shown in Figure A16. If MDiso is used for the installation process, enter in '.sa' as shown in Figure A17.

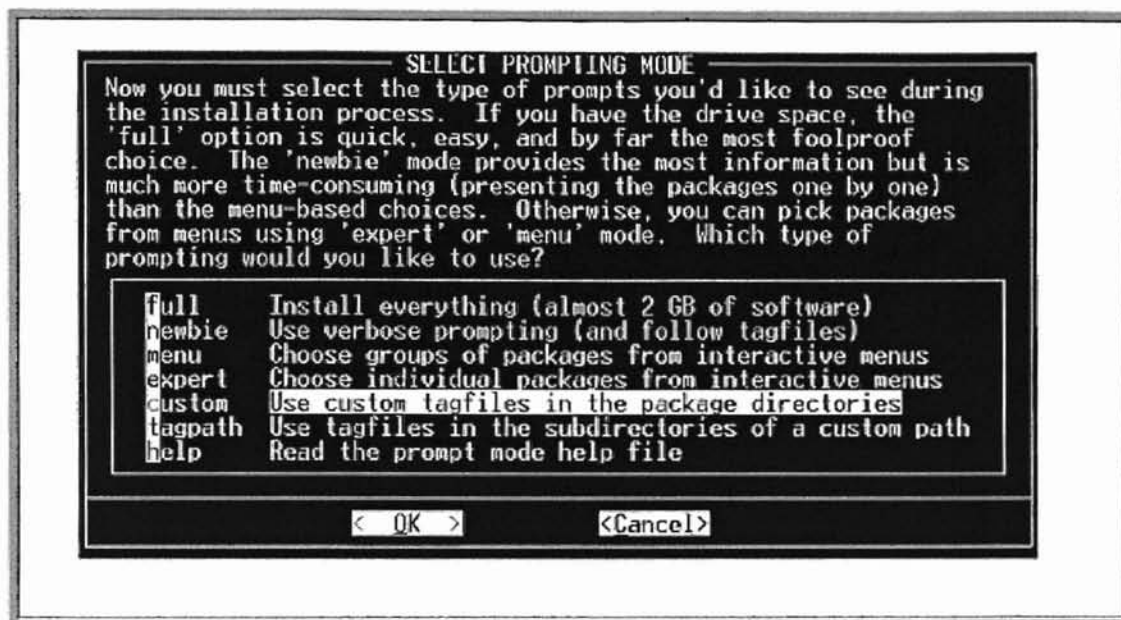


Figure A16 -- Custom tagfile install dialog

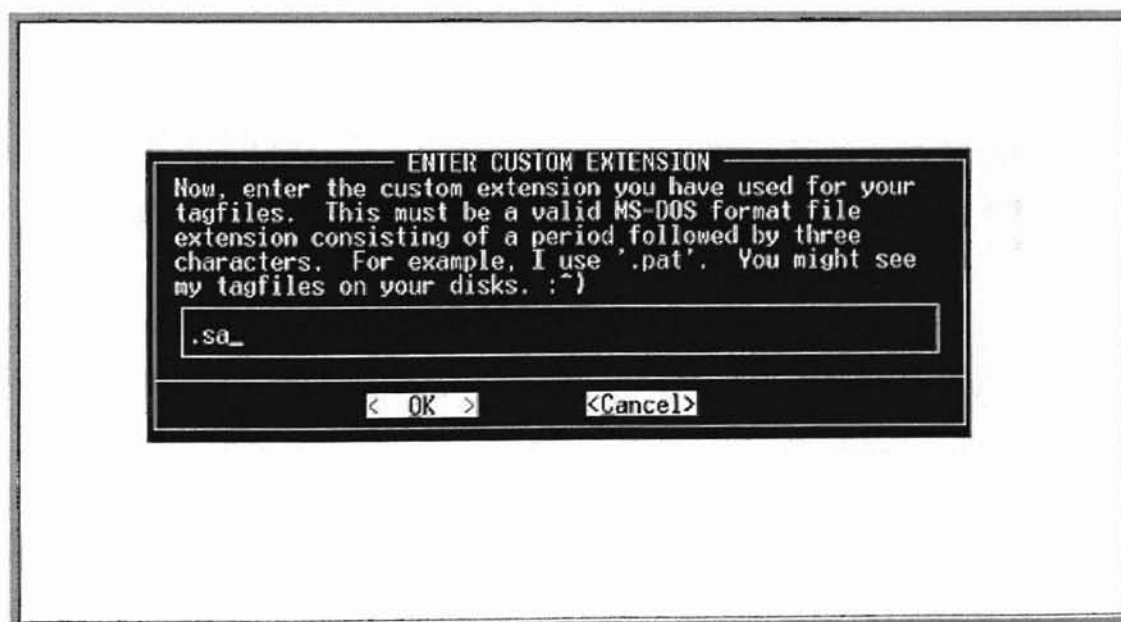


Figure A17 -- Custom tagfile extension dialog

The next part of the installation process is automated. A series of screens will flash as the different software packages install. Figure A18 shows the initial screen. After the software installation is complete, custom kernel installation can be completed. Select the kernel from the CD as shown in Figure A19.

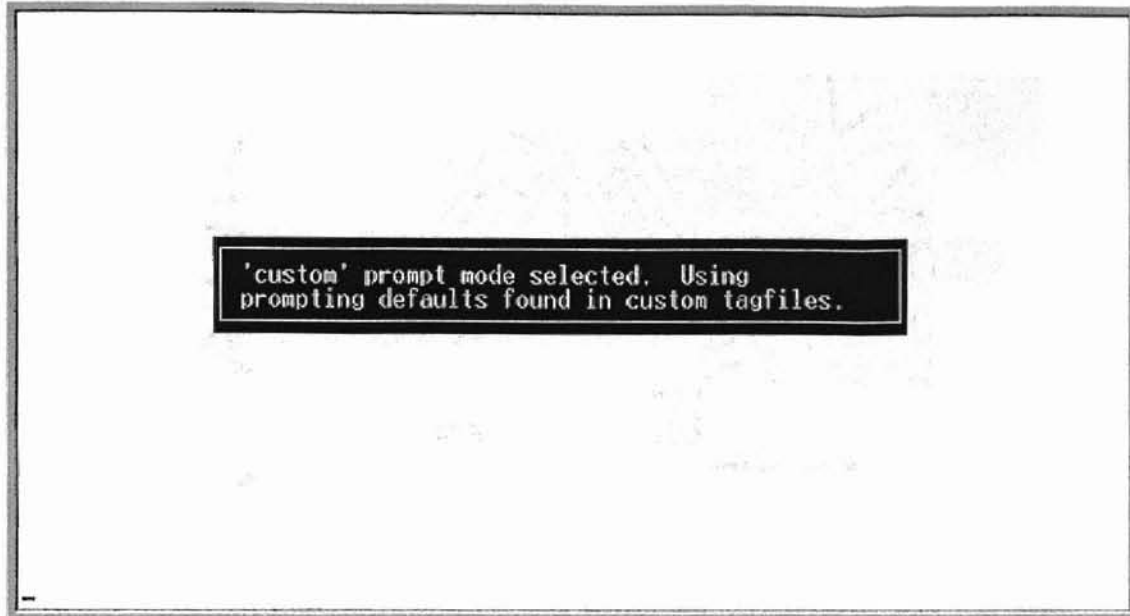


Figure A18 – Automatic installation dialog

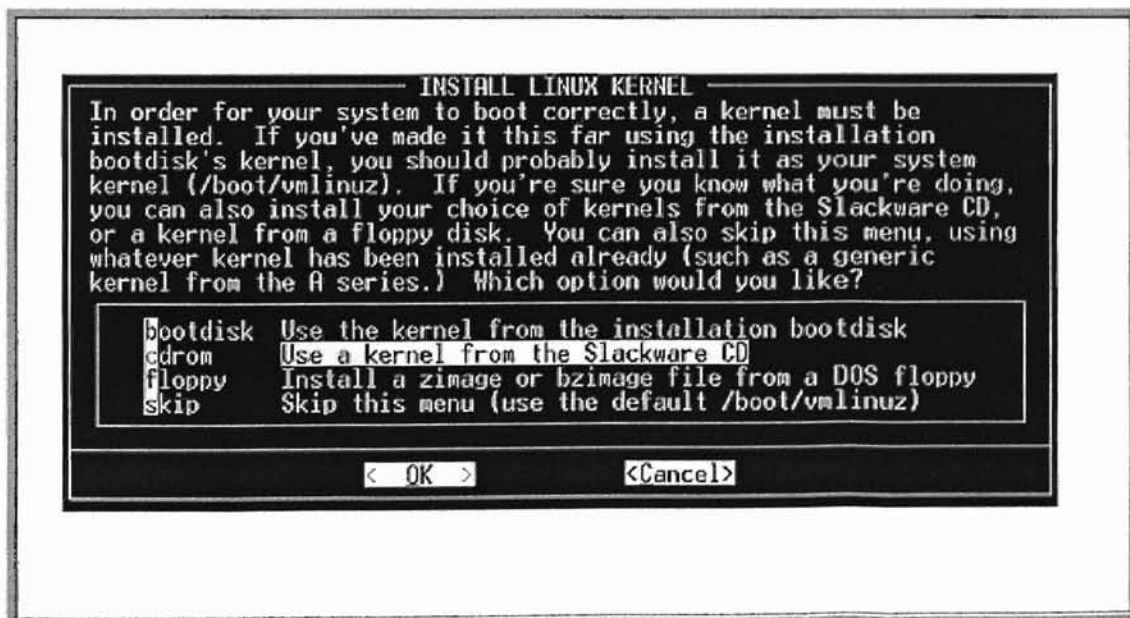


Figure A19 – Kernel installation dialog

The next menu, Figure A20, shows the different kernels that can be used to boot the system. Figure A21 shows the boot disk creation menu. Skip this menu if you booted from CD since MDiso can be used to jumpstart the system and act as the boot disk. The kernel used to boot the system will be highlighted.

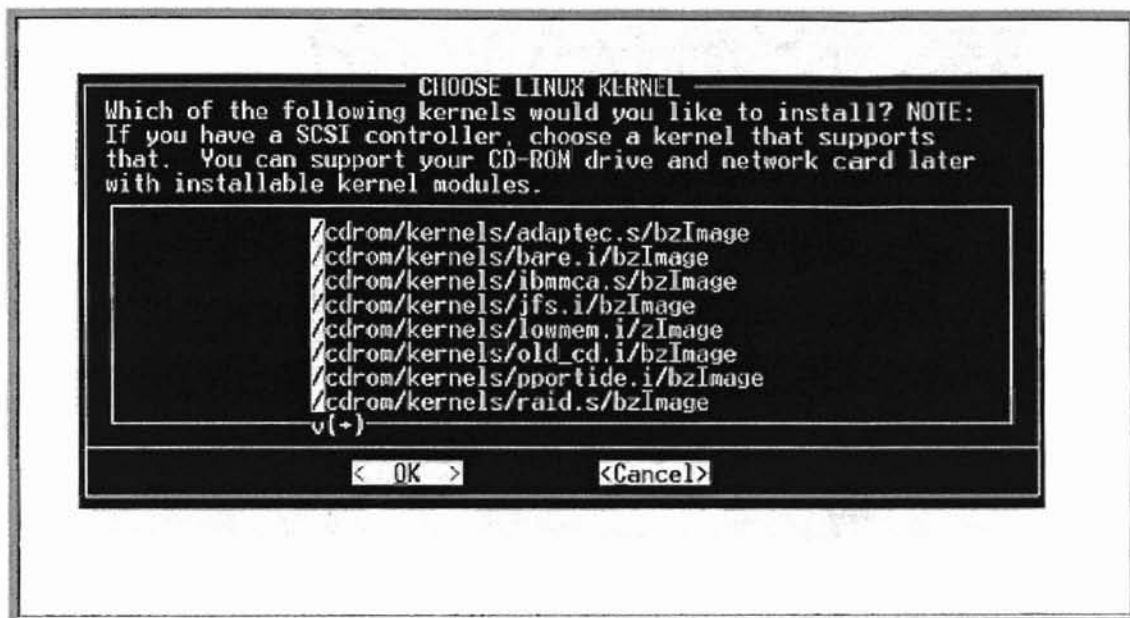


Figure A20 – Kernel selection dialog

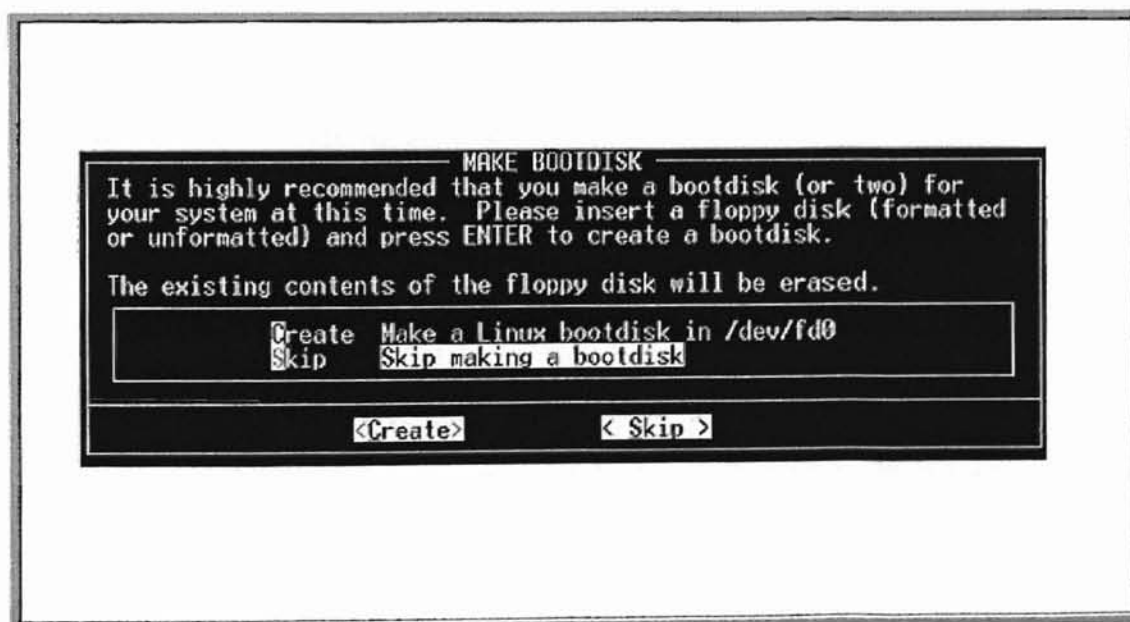


Figure A21 – Boot disk creation dialog

Figure A22 shows the modem configuration dialog. Since an Ethernet network adapter is used for remote access, there is no need to use or install a modem. Because of this, select no modem. Figure A23 shows the initial Linux Loader (LILO) installation menu. Select expert lilo.conf setup and proceed with the install.

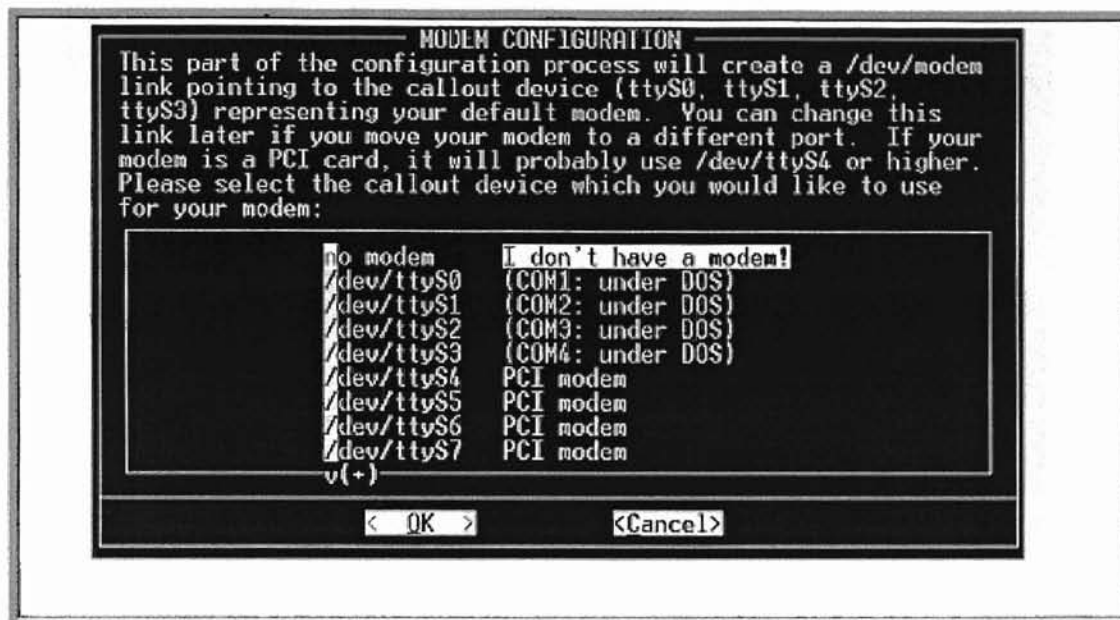


Figure A22 – Modem configuration dialog

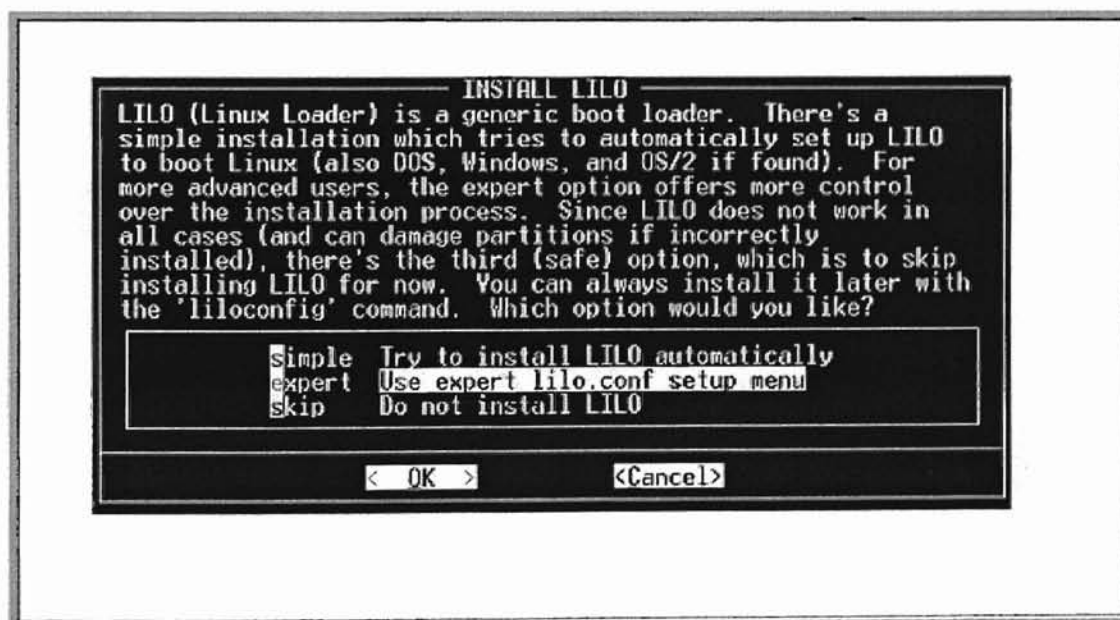


Figure A23 – LILO installation menu

Start the LILO configuration process by selecting 'begin' in the expert LILO installation menu, as shown in Figure A24. Additional parameters can be passed to the kernel at boot time by filling in the appropriate information as shown in Figure A25 or by editing the lilo.conf file and executing `lilo`.

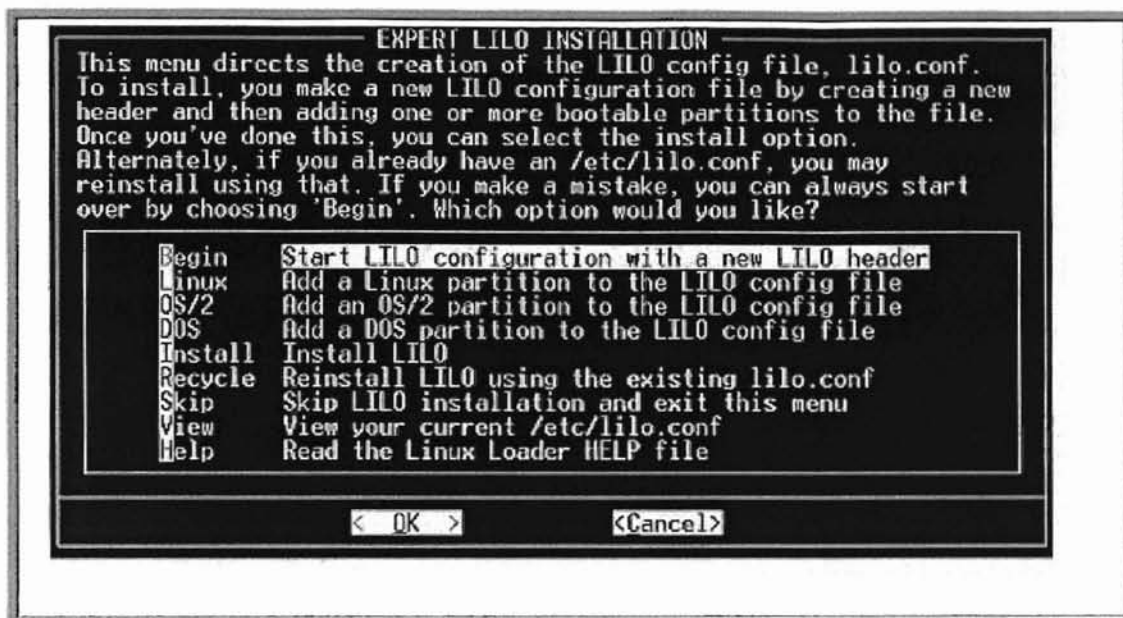


Figure A24 – Initial expert LILO installation dialog

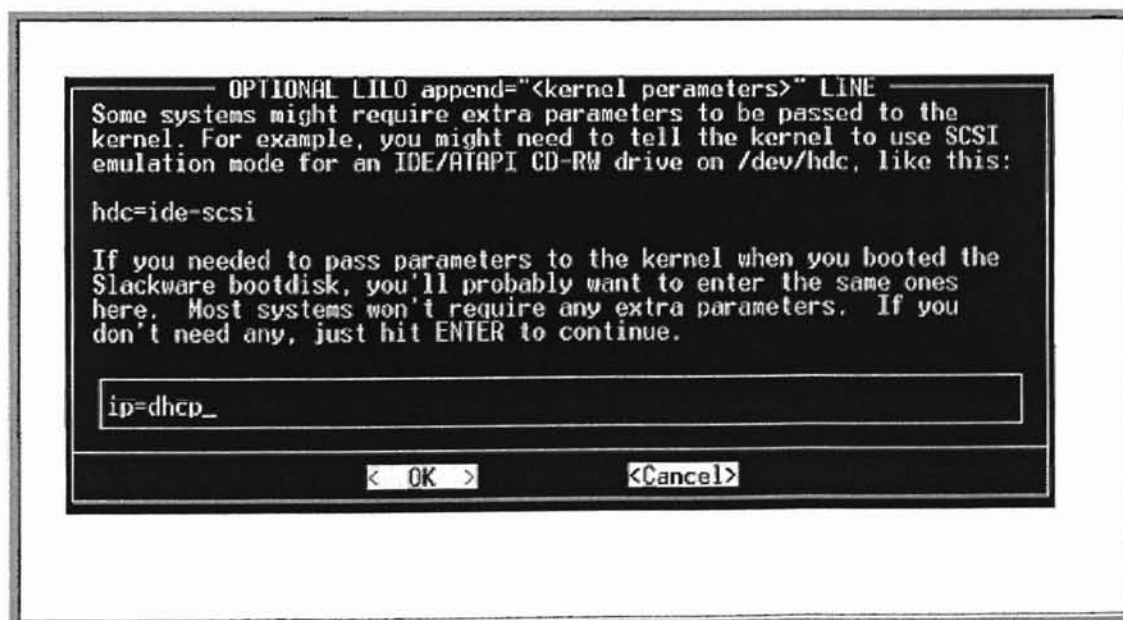


Figure A25 – LILO append line options dialog

After the boot time configuration information has been placed in the append line, select the standard non-frame buffer console, as shown in Figure A26. Figure A27 shows the target location for the LILO installation. Select MBR for the Master Boot Record and continue.

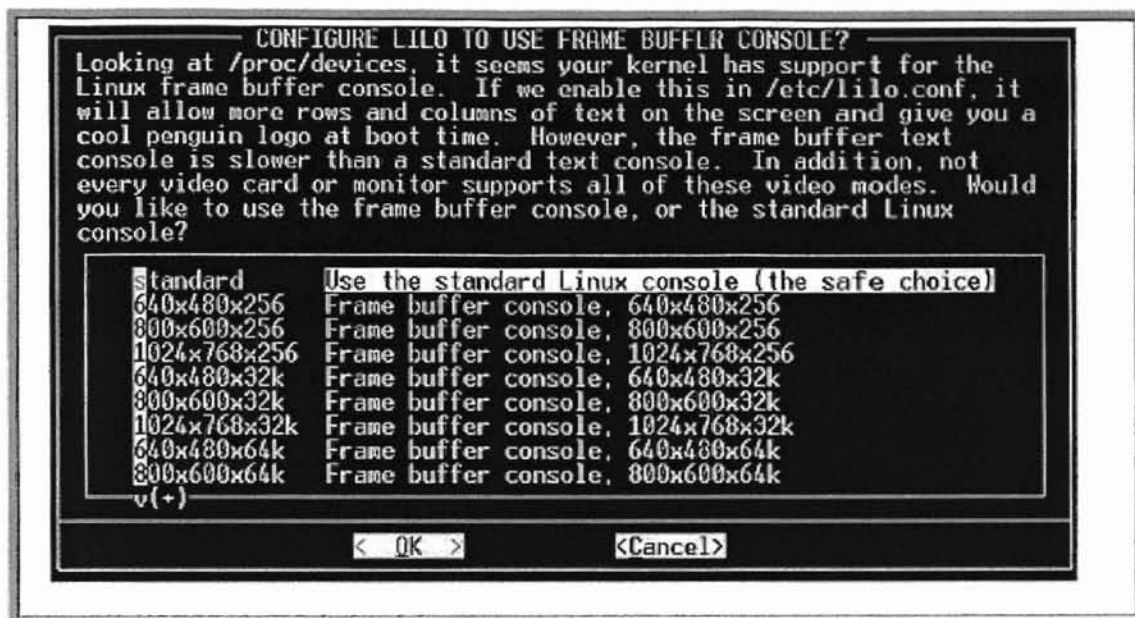


Figure A26 – Frame buffer console configuration

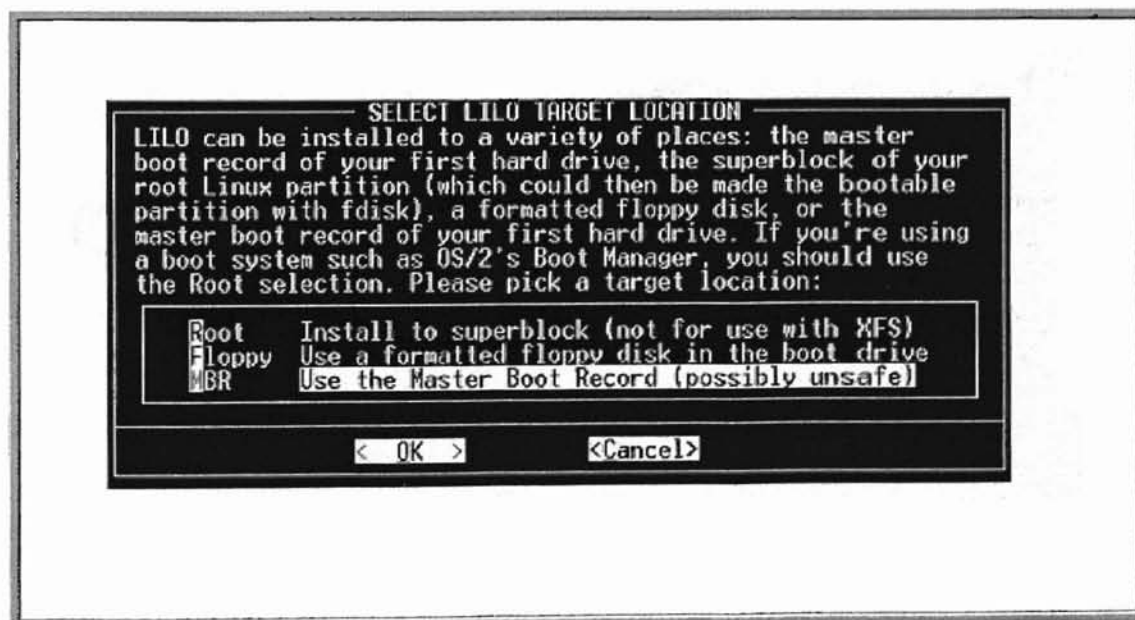


Figure A27 – LILO target installation location

The next step is to identify the root partition. Figure A28 shows the menu choices. Select Linux, and press enter. Figure A29 shows the mounted drives. Recall from Figure 6-11 which partition you selected as root and enter it in as shown in the following.

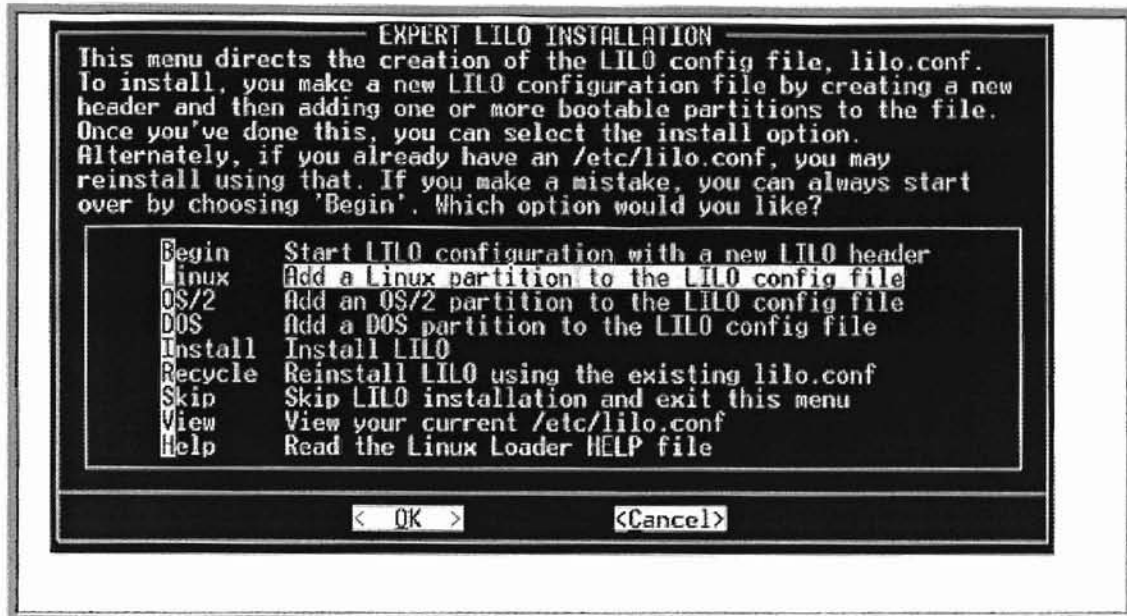


Figure A28 – Expert LILO installation dialog

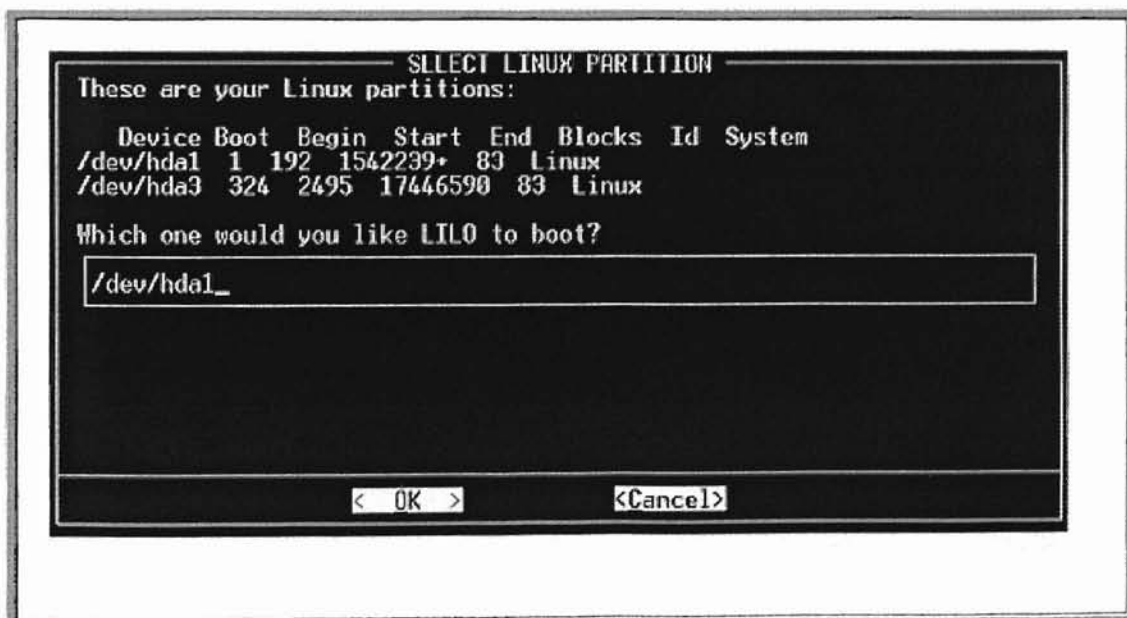


Figure A29 – LILO root partition selection dialog

Multiple kernels can be defined in the LILO configuration file (/etc/lilo.conf); therefore a label is important. Figure A30 shows the Linux kernel partition label. After completing all the above steps relating to installing the LILO loader, scroll to the Install menu and then proceed to the next section as shown in Figure A31.

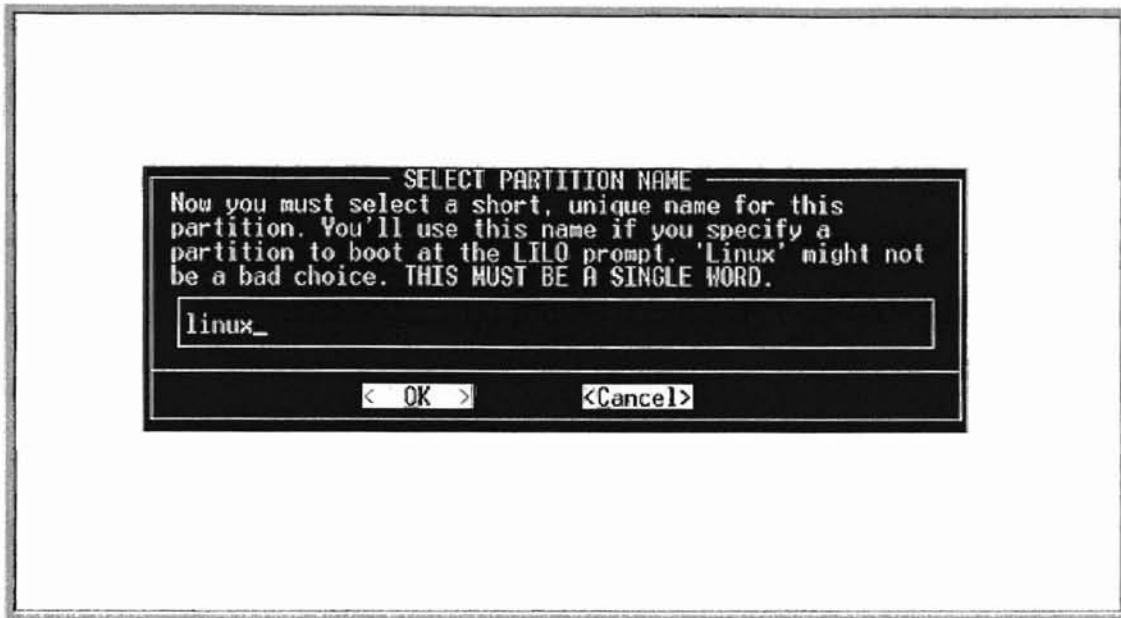


Figure A30 – LILO partition identifier dialog

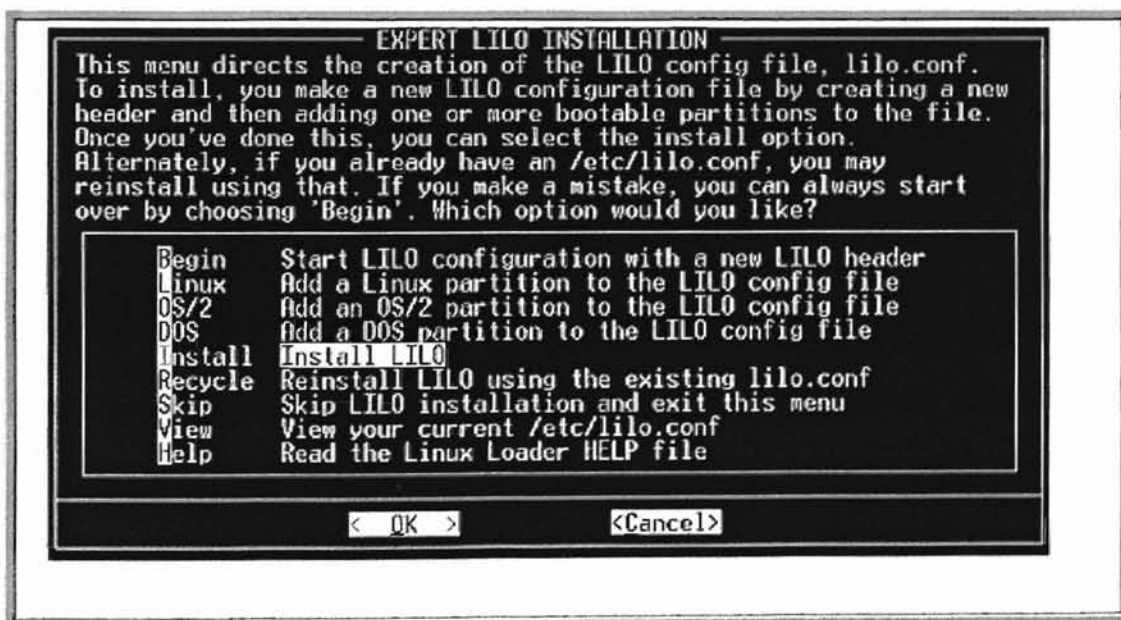


Figure A31 – Final LILO installation dialog

The machines constructed have custom configured kernels, so as to increase the stability of the systems. Configured kernels do not support modules because drivers have been compiled directly into the kernel. Due to this, the network configuration utility included in the installation scripts is rather useless. The network can be configured within the kernel configuration and the network device can be brought online during the startup process. Building a custom kernel also allows for all of the drivers for the hardware to have support while devices options are passed via kernel level parameters passed in during the boot process from the LILO append option. Another reason for not enabling the network connection at this point is that the machine is not secured. Figure A32 shows the network configuration dialog box. Select no and continue to the next part of the installation process.

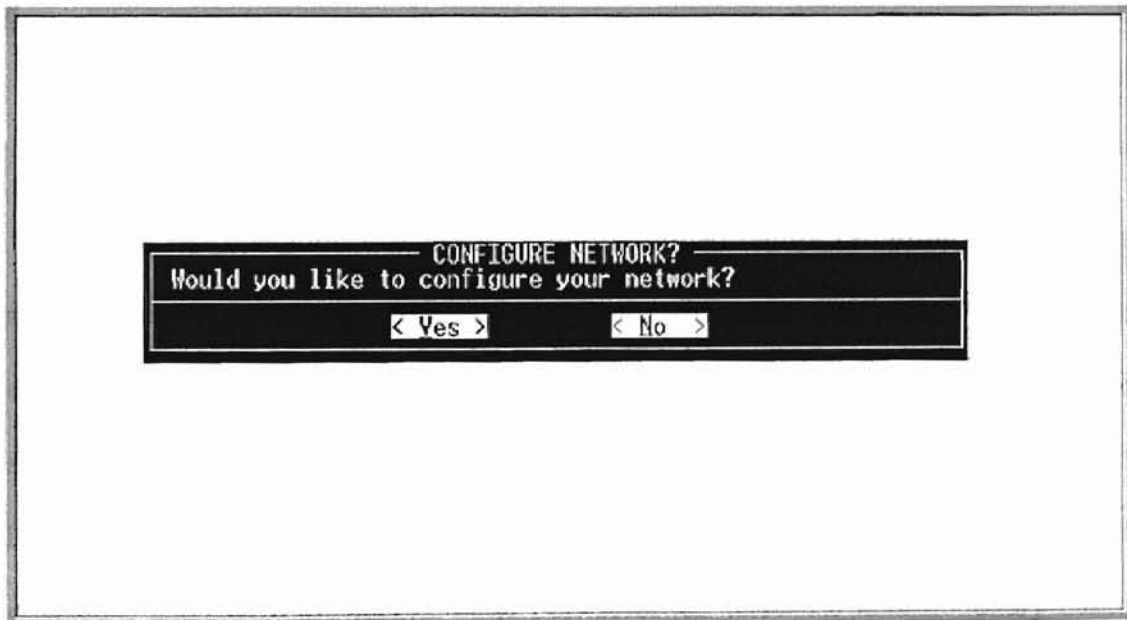


Figure A32 – Network configuration utility

The next step is to configure the hardware clock. If the BIOS contains the local time, then the hardware clock is set to local time and the no option should be selected. Figure A33 shows the hardware clock dialog box.

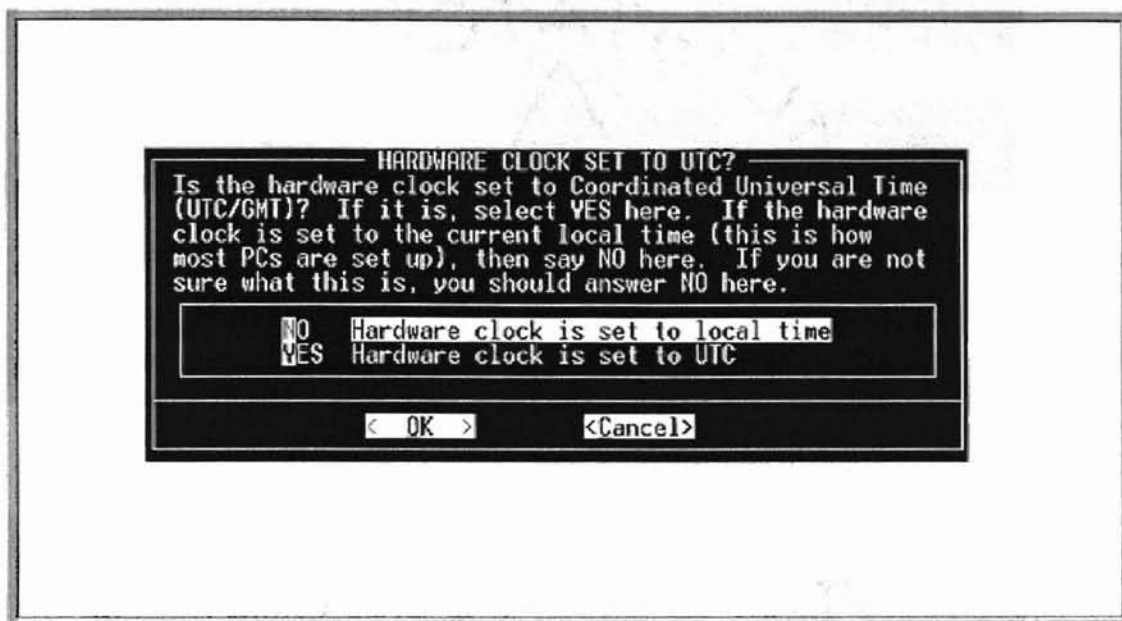


Figure A33 – Hardware clock dialog box

The last operation that needs to be performed before restarting the machine and working on the configuration is to set a root password. The root password is an extremely delicate piece of information. It should not be shared, and should never be abused. Care must also be taken when selecting a password. The password should have ASCII and numeric characters. The ASCII characters should be a mix of both uppercase and lowercase letters. The best selection is to use non-dictionary based words. Figure A34 shows the root password dialog warning. Make sure that you select, yes. The install will move to a console where you are prompted to enter in the password as shown in Figure A35.

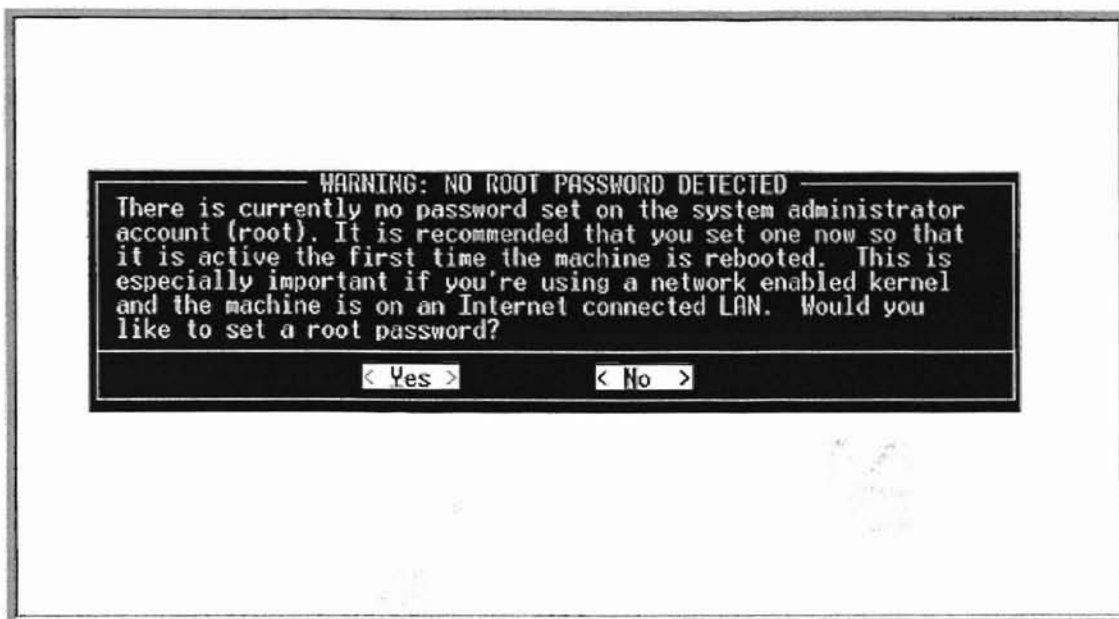
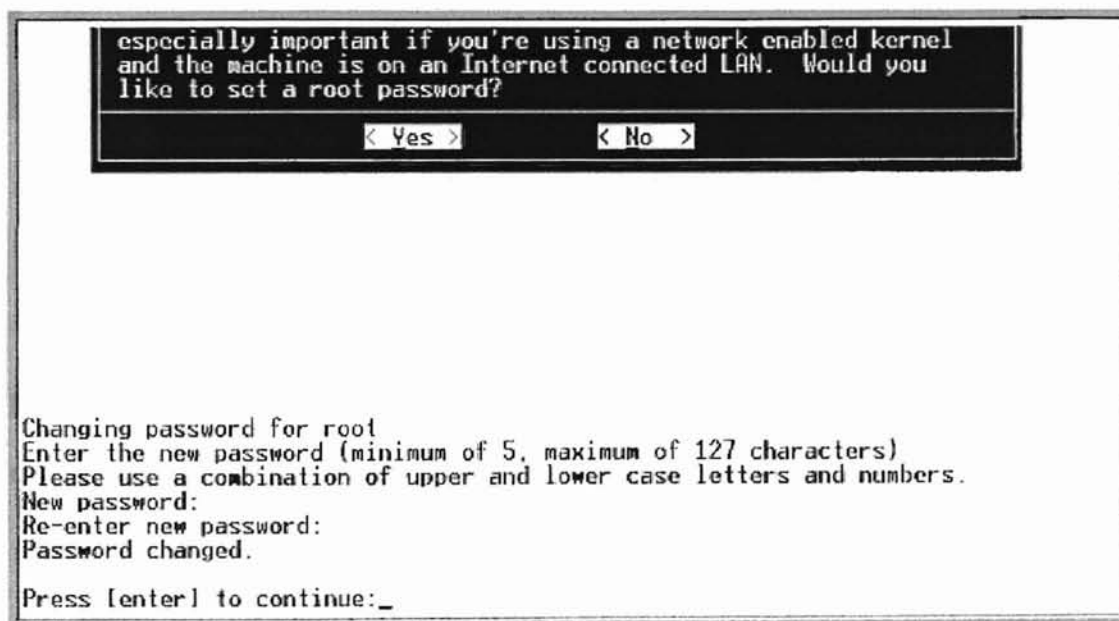


Figure A34 – Root password warning dialog



Changing password for root
Enter the new password (minimum of 5, maximum of 127 characters)
Please use a combination of upper and lower case letters and numbers.
New password:
Re-enter new password:
Password changed.
Press [enter] to continue: _

Figure A35 – Root password input from shell

After entering in the password that has been carefully selected and pressing enter as the prompt says, the final dialog box from the setup scripts will be displayed. Figure A36 shows the final dialog from the setup script. It instructs the user to

execute the **ctrl-alt-delete** reboot sequence. This will send **run-level 6** to the Linux **init**, causing the system to reboot. Be sure to remove the installation CD from drive or change the bios boot settings to boot from the hard disk first.

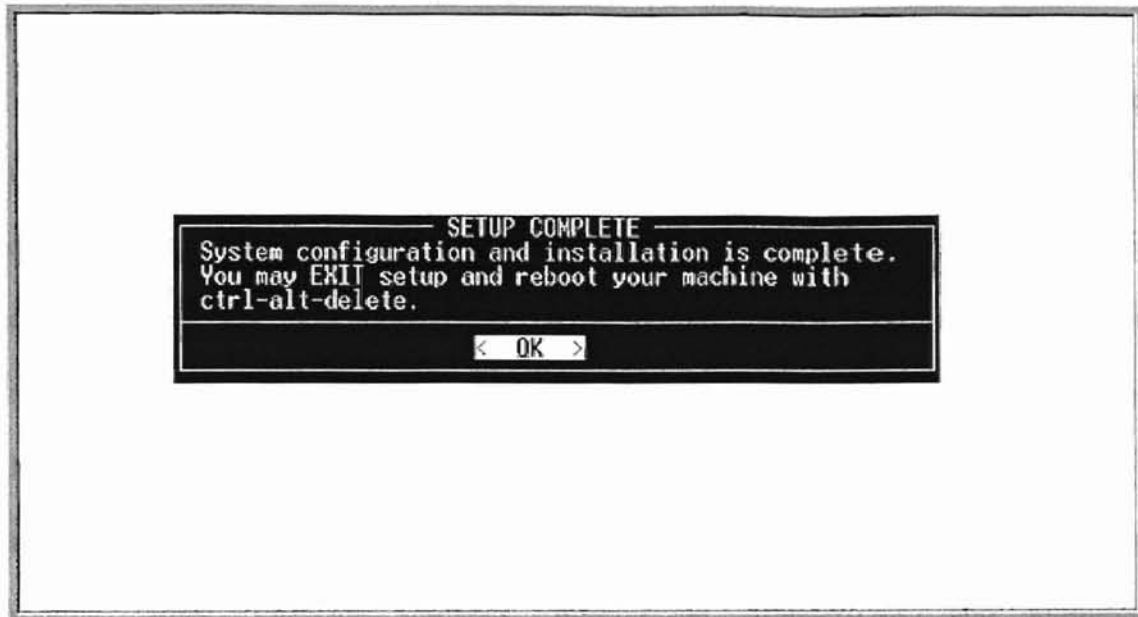


Figure A36 – Setup completion dialog

If LILO was installed correctly, the machine should boot up to a login shell prompt. At this point, the machine is ready to be configured. If all options were correctly entered, the system should boot to a login prompt.

A4. Serial Workstation Configuration

At this point in the configuration process, it is assumed that the steps up to this point have been completed. Mount the installation disk from the appropriate device. The command is **mount /dev/hdX /mnt/cdrom**, where **X** is replaced by the appropriate letter (a-z). After mounting MDiso, the custom startup scripts can be copied to the workstation. These scripts were custom written to ensure that only the

specialized services needed are running. Execute the following command to copy these custom startup scripts to the files that control the startup sequence: **cp -R /mnt/cdrom/special/saves/etc/* /etc/**. The mail message can be removed by executing **rm /var/spool/mail/root**.

The inet super daemon configuration file should be edited to not start the unwanted services. This can be done by editing the `/etc/inetd.conf` and placing a '#' character in front of each line that is unwanted. The files can be edited by executing either of the following commands: **vi /etc/inetd.conf** or **pico /etc/inetd.conf**. After modifying the configuration file, the server should be restarted by executing **kill -HUP `pidof inetd`**. To verify that all services have been shutdown, execute the **netstat -an** command. This program is capable of showing the network connections, routing tables, interface statistics, masquerade connections, and multicast memberships. The '-an' switch tells the application to display all listening and non-listening sockets in numeric form. This command is extremely useful for verifying what ports have services listening. Finally, the hostname of the machine should be changed to the appropriate machine name by executing **echo "hostname.okstate.edu" > /etc/HOSTNAME**. Figure A37 shows these commands executed on the workstation and the output of the specific commands.

```

Welcome to Linux 2.4.18 (ttyS0)
hostname login: root
Password:
Linux 2.4.18.
root@hostname:~# mount /dev/hdb /mnt/cdrom
mount: block device /dev/hdb is write-protected, mounting read-only
root@hostname:~# cp -R /mnt/cdrom/special/saves/etc/* /etc/
root@hostname:~# rm /etc/rc.d/rc.serial /etc/rc.d/rc.inet1 /etc/rc.d/rc.inet2
root@hostname:~# rm /var/spool/mail/root
root@hostname:~# kill -HUP `pidof inetd`
root@hostname:~# netstat -an
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags               Type                   State                  I-Node Path
unix  3      [ ]                   DGRAM                  38                      /dev/log
unix  2      [ ]                   DGRAM                  41
root@hostname:~# echo "hostname.okstate.edu" > /etc/HOSTNAME
root@hostname:~#

```

Figure A37 – Locking down a serial workstation or server node

All the remote services have been disabled at this point. A new kernel should be configured so that the network device is activated. It is possible to use a pre-configured kernel, however because hardware is evolving and improvements are constantly implemented into new versions of the kernel, a sample configuration file is provided and it is recommended that the newest kernel be configured with this file. If the reader is unable to compile a kernel correctly, a reconfigured kernel may be used and can be found in the **/special** directory of the MDiso. It can also be obtained from <ftp://ftp.kernel.org/pub/linux/kernel/v2.4/>. At the time MDiso was created, the current kernel was provided. Copy the kernel source file to the source directory: **cp /mnt/cdrom/special/linux-2.4.18.tar.gz /usr/src/**. CD to the **/usr/src** directory and extract the kernel source distribution in the current directory and create a link to access the directory: **cd /usr/src** and then **tar zxvf linux-2.4.18.tar.gz**. To use the configuration provided on MDiso: **ln -s linux-2.4.18 linux; cd /usr/src/linux; cp**

`/mnt/cdrom/special/saves/kernel.conf .config`; Figure A38 shows the setup of the kernel source tree.

```
root@hostname:~# mount /dev/hdb /mnt/cdrom
mount: block device /dev/hdb is write-protected. mounting read-only
root@hostname:~# cp /mnt/cdrom/special/linux-2.4.18.tar.gz /usr/src/
root@hostname:~# cd /usr/src
root@hostname:/usr/src# tar xzf linux-2.4.18.tar.gz
root@hostname:/usr/src# mv linux linux-2.4.18
root@hostname:/usr/src# ln -s linux-2.4.18 linux
root@hostname:/usr/src# cd linux
root@hostname:/usr/src/linux# make menuconfig_
```

Figure A38 – Kernel source tree setup

To compile and install the kernel, execute the following commands: **make dep**; **make clean**; **make bzlilo**. After the kernel has been compiled, reboot the machine by executing a **ctrl-alt-del** or the command **shutdown -r now**.

A5. SSH Configuration for Gateway Node

The configuration of the gateway node is extremely important. This node controls all connections to and from the cluster. The only service that should be allowed from the outside is the secure shell service running on port 22. Many services may be running on the gateway node. Using iptables as mentioned in the hardware section allows for connections to be controlled. To configure secure shell services, download the latest version of SSH from ftp.ssh.com/ssh/. This application

provides the encrypted communications for using the cluster remotely. Make sure that the latest version of the SSH distribution is downloaded because known bugs and vulnerabilities are usually fixed in current releases of software.

After obtaining the source distribution, extract the source by executing the following command: **tar zxvf ssh-x.x.x.tar.gz** where **x.x.x** is replaced by the most recent version. After extracting the source distribution, cd into the source directory: **cd ssh-x.x.x**. At this point, as with any source distribution, view the README and INSTALL files. These files give the reader proper information and warnings for configuring the software. Read carefully and follow the directions provided with the source installation.

After finishing the installation process for the SSH service, the firewall needs to be configured. Setting up access to the cluster through the firewall is a powerful tool to control access to the cluster.

The firewall script, Figure A39, populates the input and output chain that controls connections to the services. The important feature to understand about the script is not how it works or what it does, but how to configure it. By default all the services are blocked. This means that if external services or access is needed, the iptables firewall must be altered. After completing modifications, the firewall can be tested by executing the firewall script with the following command.

./etc/rc.d/rc.firewall

```
#!/bin/bash

# Set these variables
INET_IP="139.78.xx.xx"
INET_IFACE="eth1"
LAN_IP="192.168.0.254"
LAN_BCAST_ADRESS="192.168.0.255"
LAN_IFACE="eth0"
LO_IP="127.0.0.1"
IPTABLES="/usr/sbin/iptables"

# Bring up the lan interface
/sbin/ifconfig $LAN_IFACE $LAN_IP
/sbin/ifconfig $LAN_IFACE netmask 255.255.255.0 broadcast 192.168.0.255

# Flush the original chains
$IPTABLES -F FORWARD
$IPTABLES -F OUTPUT
$IPTABLES -F INPUT

# Set the default policies
$IPTABLES -P INPUT DROP
$IPTABLES -P OUTPUT DROP
$IPTABLES -P FORWARD DROP

# Set the rules for the INPUT chain
$IPTABLES -A INPUT -p ALL -m state --state INVALID -j DROP
$IPTABLES -A INPUT -p tcp ! --syn -m state --state NEW -j DROP
$IPTABLES -A INPUT -p TCP -m state --state ESTABLISHED,RELATED -j ACCEPT

# Allow cluster to return external ping requests
$IPTABLES -A INPUT -p ICMP -s 0/0 --icmp-type 0 -j ACCEPT
$IPTABLES -A INPUT -p ICMP -s 0/0 --icmp-type 3 -j ACCEPT
$IPTABLES -A INPUT -p ICMP -s 0/0 --icmp-type 5 -j ACCEPT
$IPTABLES -A INPUT -p ICMP -s 0/0 --icmp-type 8 -j ACCEPT
$IPTABLES -A INPUT -p ICMP -s 0/0 --icmp-type 11 -j ACCEPT

# External services (add access to ssh here)
$IPTABLES -A INPUT -p TCP -s 139.78.xx.xx --dport 22 -j ACCEPT
$IPTABLES -A INPUT -p TCP -s 0/0 --dport 22 -j ULOG          # log ssh attempts
$IPTABLES -A INPUT -p UDP -s 0/0 --sport 53 -j ACCEPT        # allow dns queries

# Set the rules for the cluster side, lo interface, and OUTPUT chain (no need to touch)
$IPTABLES -A INPUT -p ALL -i $LAN_IFACE -d $LAN_BCAST_ADRESS -j ACCEPT
$IPTABLES -A INPUT -p ALL -i $LAN_IFACE -j ACCEPT
$IPTABLES -A INPUT -p ALL -d $LAN_IP -j ACCEPT
$IPTABLES -A INPUT -p ALL -d $LO_IP -s $LO_IP -j ACCEPT
$IPTABLES -A OUTPUT -p ICMP -m state --state INVALID -j DROP
$IPTABLES -A OUTPUT -p tcp ! --syn -m state --state NEW -j DROP
$IPTABLES -A OUTPUT -p ALL -s $INET_IP -j ACCEPT
$IPTABLES -A OUTPUT -p ALL -s $LAN_IP -j ACCEPT
$IPTABLES -A OUTPUT -p ALL -s $LO_IP -d $LO_IP -j ACCEPT
```

Figure A39 – Firewall boot script

A6. Computational Slave Node Configuration

The last important area to focus on for administration and maintenance is the addition of a computational node. When more nodes are added, the following steps will be mandatory to give the node access to the services running on the server nodes.

The easiest way to explain how to add an additional node to the cluster is by way of example. At this point, 12 computational nodes are currently in the cluster. The following dialog is given for adding the 13th computational node.

The first step to adding an additional computational node into the MDbeta cluster is to connect to the file server node. After logging in as root into the file server node, **cd /tftpboot** to the nfsroot directory shares. This location stores all of the nfsroot mount points that are shared through the network file system. Next, make a copy of the base computational nodes directory IP by executing the following command: **cp -R 192.168.0.1 192.168.0.13** where '13' indicates the node number. Create a copy of the PXE batch file by executing the following command: **cp 192.168.0.1.bpb 192.168.0.13.bpb** where '13' indicates the node number. Edit the newly created file to reflect the new mount point that was just added: **LinuxBoot "linux.slave.krn" "root=/dev/nfs nfsroot=192.168.0.254:/tftpboot/192.168.0.13 ip=bootp"** where '13' again indicates the node number. Edit the /etc/exports file and add the appropriate line as follows: **/tftpboot/192.168.0.13 192.168.0.13(rw,no_root_squash)** where both '13's indicate the node number. Execute a **/etc/rc.d/rc.nfsd restart** to rehash network file server and allow for the node to attach to the newly created mount points.

Finally, the IP address needs to be assigned in the `/etc/dhcpd.conf` file. The DHCP server uses the hardware address of the newly added node in order to assign the correct address to the correct node. This is only done for bookkeeping measures in case nodal failure occurs. To determine the hardware address of the new node, power the machine and wait for the hardware address to be displayed. Once the hardware address is gathered, add the following series of lines in the DHCP configuration file in order to give the DHCP server permission to assign an IP address and host to the newly created node:

```
host md13 {  
    hardware ethernet XX:XX:XX:XX:XX:XX;  
    fixed-address 192.168.0.13;  
    option bpbatch-option "192.168.0.13";  
    option host-name "md13";  
}
```

The DHCP server must now be restarted. Unfortunately, the server does not support rehashing. Therefore, execute `killall `pidof dhcpd`` and then `dhcpd` to restart the server. At this point, the node is ready to be booted and will automatically be entered into the cluster. The final remaining step is to add the correct nodal name and number of processors into the mpich configuration file located `/tftpboot/usr-common/mpich/share/machines.LINUX`. Open this file and add the newly installed node using the same format as the existing entries. After completing all of the above steps, an additional node should be operational in the MDbeta cluster.

APPENDIX B

INTRODUCTION TO USING LINUX WITH SECURE COMMUNICATIONS

B1. Introduction

Many students are familiar with the term Linux, but some have little or no experience using this powerful operating system. For those who are not familiar, this document will be a valuable tool. Those who are familiar with Linux can use this document as a quick reference. Information provided here is aimed at the systems (and their configuration) that are used in our lab. Each user is given what is called a shell. This shell provides the environment for creating and running simulations. Common shells available are Bourne Again Shell (bash), Korn Shell (ksh), and C Shell (csh).

Manuals for all commands on the Linux workstations can be viewed by using the **man** command. This command is one of the single most important tools that can be found on the Linux machines. If in doubt as to what a specific program does, or how to execute the specific command in question, check **man <command>**. With the use of **man**, your question will be answered in most instances.

B2. Logging Into Workstations with Putty/SSH

All of the machines running in the MD laboratory at OSU are running a secure version of telnet called Secure Shell (ssh). Ssh provides a means for encrypting the

communications, including usernames and passwords, between two connected hosts. Connections can be made between Linux hosts or from Windows based machines to Linux hosts. To connect to the Linux workstation from a Windows machine, use putty. This application may be downloaded from <http://md.okstate.edu/tools/putty.exe>. To connect to the machines in the lab, open putty, and type in the host name of the machine that you wish to connect. Please note that it is a requirement to select the ssh protocol below the host name. After selecting the machine to which you wish to connect, click on the open button at the bottom. Options for saving sessions and other features can be found by browsing the application menu on the left. Figure B1 shows the putty configuration dialog.

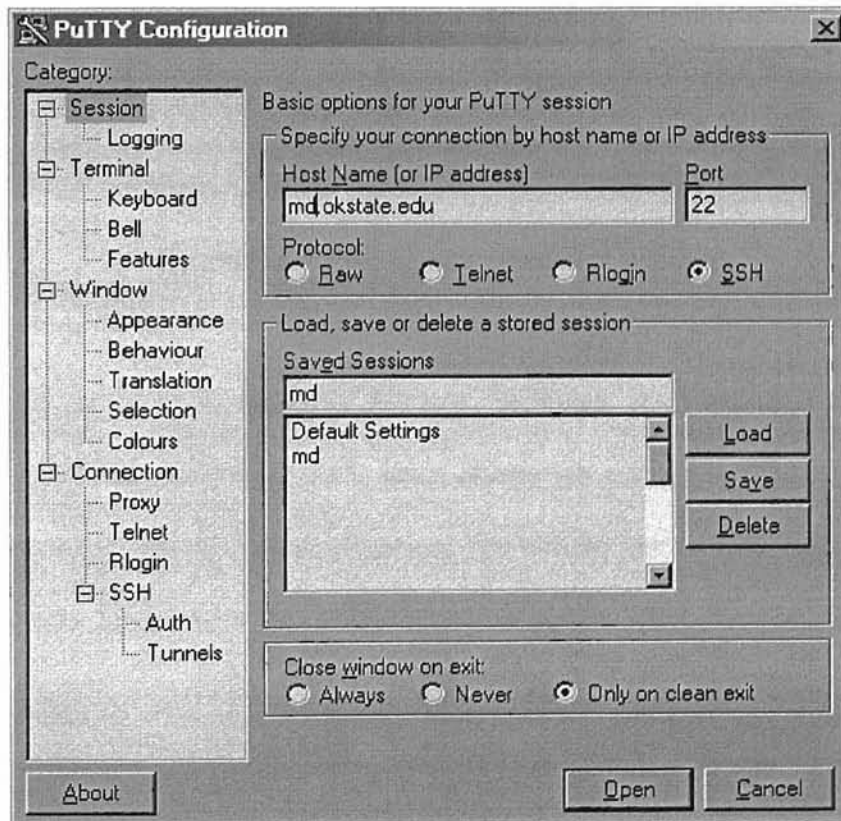


Figure B1 – Putty configuration dialog

Enter your assigned username and password. Keep in mind that all dialog and communications entered into the putty window are encrypted. Figure B2 shows the putty communications window.

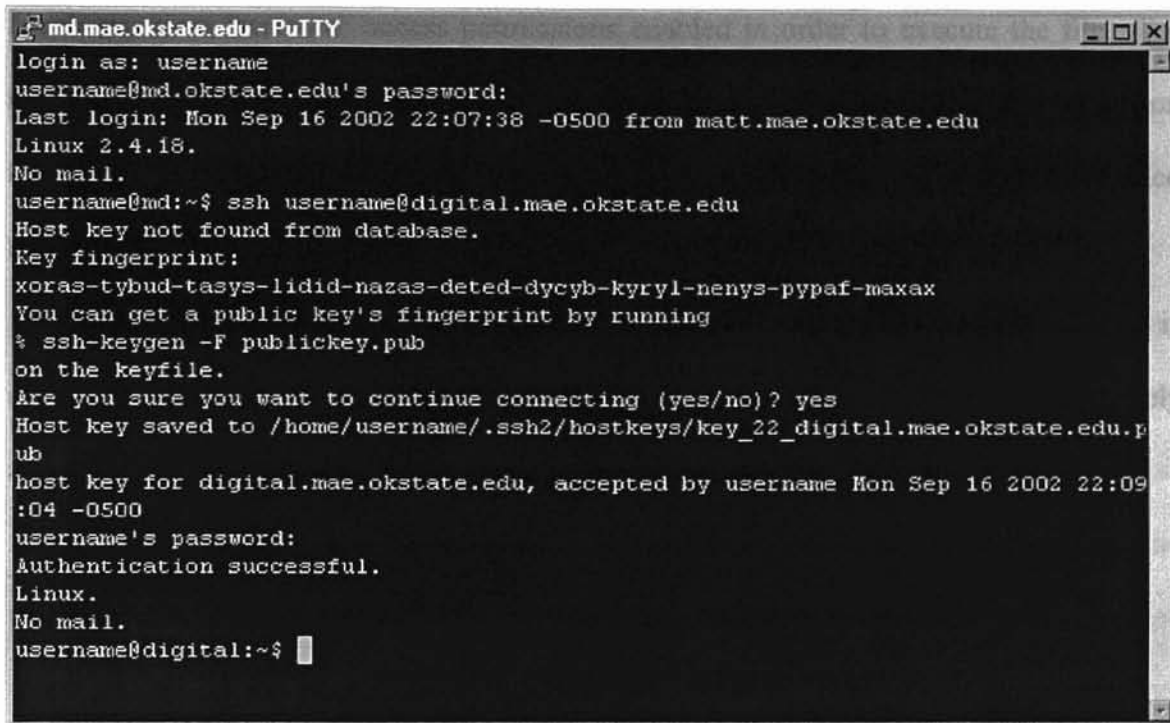


Figure B2 – Putty communications window

The above communications window provides the mechanism for running simulations and doing other important tasks on the lab machines running Linux. Once you have logged into the desired workstation, you will be placed into your user directory. Each unique user logged into the workstation has his/her own directory that other users cannot modify.

If a user on a Linux machine desires to connect to another Linux machine, execute the **ssh <username>@<host>**. This can be useful for quickly checking the

status of another application or any other minor task. Note that all communications to the second host is encrypted, sent to the first host, decrypted, re-encrypted and sent to the second. Figure B3 shows the execution of ssh within an existing putty terminal.



```
md.mae.okstate.edu - PuTTY
login as: username
username@md.okstate.edu's password:
Last login: Mon Sep 16 2002 22:07:38 -0500 from matt.mae.okstate.edu
Linux 2.4.18.
No mail.
username@md:~$ ssh username@digital.mae.okstate.edu
Host key not found from database.
Key fingerprint:
xoras-tybud-tasys-lidid-nazas-deted-dycyb-kyryl-nenys-pypaf-maxax
You can get a public key's fingerprint by running
% ssh-keygen -F publickey.pub
on the keyfile.
Are you sure you want to continue connecting (yes/no)? yes
Host key saved to /home/username/.ssh2/hostkeys/key_22_digital.mae.okstate.edu.p
ub
host key for digital.mae.okstate.edu, accepted by username Mon Sep 16 2002 22:09
:04 -0500
username's password:
Authentication successful.
Linux.
No mail.
username@digital:~$
```

Figure B3 – Connecting between shells

B3. Utilizing the Shell

The shell is an important application on the Linux machine. It is the interface to the kernel and allows for applications to be run. The shell is much like the command prompt in the DOS or Windows environment. The default shell on the Linux machines is bash.

Directory structures are similar to those found in the DOS/Windows environment. Binary files in the current directory, but not in the user path, (viewable by executing **echo \$PATH**) require a **./** in front of the binary name in order to be executed. Linux does not

use file extension associations for execution. The period “.” is an ASCII character just as the letter “p” is. This means that a file called file.1.2.23 is just as valid as a file named data.dat. This implies that Linux regards the filename as unimportant. However, binary files must have “execute” access permissions enabled in order to execute the file. For more information, type **man chmod**. Use file extensions that describe the information that is contained in the file for identification purposes only. For example, a file named test.c would most likely be C source code.

A list of some of the basic commands that the reader should be familiar with is shown in Table B1. Keep in mind that the following list is not exhaustive, but merely aims to touch on the surface of the commands available on the Linux machines.

Table B1 – List of commonly used Linux commands

cd	change working directory
clear	clear the terminal screen
cp	copy files or directories
df	report filesystem disk space usage
du	estimate file space usage
exit	terminate the current process
grep	print lines matching pattern
ls	list directory contents
man	format and display the on-line manual pages
mkdir	make directories
mv	move (rename) files
ping	send ICMP ECHO_REQUEST packets to network hosts
ps	report process status
pstree	display a tree of processes
rm	remove files or directories
screen	screen manager with VT100/ANSI terminal emulation
tar	tar archiving utility
uname	print system information
vi	vi text editor (man elvis)
zip	package and compress (archive) files

Remember that wildcards may also be used as parameters when executing many of these commands. The `*` matches text of any length whereas the `?` matches a single ASCII character. Keep in mind that any of the commands can be reviewed in detail by executing **man <command>**.

B4. Transferring Files Under Windows Using psftp

Utilizing the workstation from the shell is an important aspect to master. However, one function that putty/ssh does not handle is transferring files to and from the workstations. The file transfer protocol (ftp), much like telnet, is not a viable method of transferring files because data and passwords are sent in clear text. An alternative to this method is to use the secure ftp protocol, provided by the ssh software suite. A graphical implementation of sftp, called Cute FTP Pro, has been developed and can be downloaded from the web. However, this piece of software requires a license and is therefore not recommended. The recommended software for transferring files is psftp. Psftp is freeware or free software and can be downloaded from <http://md.okstate.edu/tools/psftp.exe>. After downloading and executing this application, type **open <username>@<host>** to initiate the connection. Once connected, regular ftp commands like **get** and **put** can be used. If you are unsure what commands to execute, enter **help** once login has been granted. Figure B4 shows an example of transferring a file from a remote workstation to the local machine.

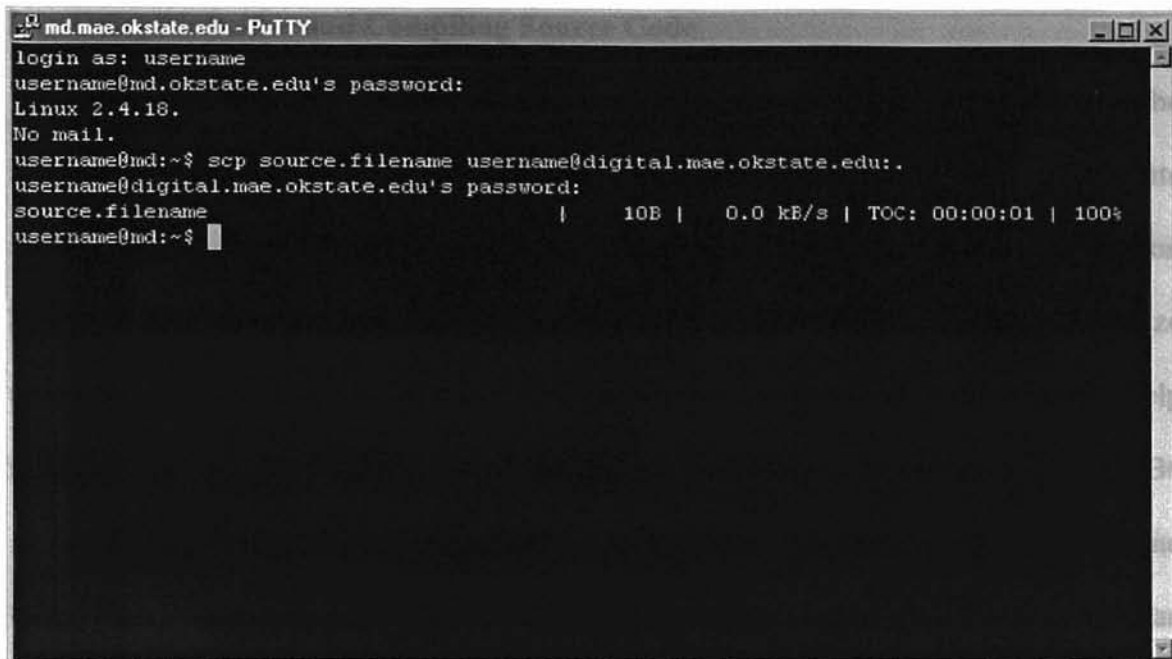
```
D:\WINNT\System32\cmd.exe - psftp -v username@md.okstate.edu
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-1999 Microsoft Corp.

D:\>psftp -v username@md.okstate.edu
Looking up host "md.okstate.edu"
Connecting to 139.78.74.164 port 22
Server version: SSH-2.0-3.2.0 SSH Secure Shell (non-commercial)
We claim version: SSH-2.0-PuTTY-Snapshot-2002-02-21
Using SSH protocol version 2
Doing Diffie-Hellman key exchange
Host key fingerprint is:
ssh-dss 1024 19:62:23:3f:f1:53:71:38:81:3c:e0:76:bb:b8:7d:e2
Using username "username".
username@md.okstate.edu's password:
Sent password
Access granted
Opened channel for session
Started a shell/command
Connected to md.mae.okstate.edu
Remote working directory is /home/username
psftp> ls
Listing directory /home/username
drwxr-xr-x  3 username users      4096 Sep 16 23:55 ./
drwxr-xr-x  6 root      root      4096 Sep 16 23:50 ../
-rw-r--r--  1 username users      126 Sep 16 23:55 .bash_history
-rw-r--r--  1 username users    3394 Sep 16 23:50 .screenrc
drwxr-xr-x  3 username users     4096 Sep 16 23:52 .ssh2/
-rwxr-xr-x  1 username users       68 Sep 16 23:50 mailsetup*
-rw-r--r--  1 root      root       10 Sep 16 23:51 source.filename
psftp> get source.filename
remote:/home/username/source.filename => local:source.filename
psftp>
```

Figure B4 – Transferring files using sftp

B5. Transferring Files Under Linux Using sftp and scp

To transfer files between different Linux workstations, sftp or scp can be used. Sftp on the Linux workstations operates in the same manner as psftp. Another alternative is to use scp. Scp requires less input from the user and just copies the file to the desired workstation via encrypted communications. Execute **scp <source.filename> <username>@<host>:<destination.filename>**. Figure B5 shows an example of a file transfer between two Linux machines.

A screenshot of a PuTTY terminal window titled "md.mae.okstate.edu - PuTTY". The terminal shows a login sequence: "login as: username", "username@md.okstate.edu's password:", "Linux 2.4.18.", and "No mail.". The user then enters the command "scp source.filename username@digital.mae.okstate.edu:.". This is followed by the destination host's password prompt: "username@digital.mae.okstate.edu's password:". The transfer progress is shown as "source.filename | 10B | 0.0 kB/s | TOC: 00:00:01 | 100%". The prompt "username@md:~\$" appears at the end of the line.

```
md.mae.okstate.edu - PuTTY
login as: username
username@md.okstate.edu's password:
Linux 2.4.18.
No mail.
username@md:~$ scp source.filename username@digital.mae.okstate.edu:.
username@digital.mae.okstate.edu's password:
source.filename | 10B | 0.0 kB/s | TOC: 00:00:01 | 100%
username@md:~$
```

Figure B5 – Transferring files with scp between Linux workstations

B6. Using the Text Editor

There are several different text editors available on the MD workstations. The two most commonly used text editors on the Linux workstations are **vi** and **pico**. Pico, found in the pine mail distribution, allows for editing of files in a user-friendly format, similar to the notepad text editor. To open a file in pico, execute **pico filename**. Instructions are given once the editor has been executed. 'vi' is the recommended text editor for users that plan on spending large amounts of time working in Linux. It was originally designed to operate smoothly without using large amounts of bandwidth by employing shortcut keys to help speed up editing text files.

Because pico is simple to use, the usage of this program will not be given. For those interested in becoming proficient with 'vi', download a detailed manual from any one of the many 'vi' text editor web pages.

B7. Using Makefiles and Compiling Source Code

Maintaining large and even small source distributions on a Linux machine can be a daunting task. S.I. Feldman at AT&T Bell Labs designed **make** in 1975 to automate and optimize the construction of programs. When building projects, the make application keeps track of what files have been modified since the last build. This helps to minimize time spent in compiling source code. Default macros are provided within make to help manage projects. To display a list of the default macros execute **make -p**. Figure B6 shows a sample makefile. In the sample makefile, **make clean** removes all object and binary files. Executing **make** builds the project and creates the binary. These scripts can be modified to suit custom compilation needs.

```
OBJECTS = main.o

EXEC = EXEFILE

COMPILER = gcc
FLAGS = -O3
LIBS =

$(EXEC): $(OBJECTS)
    $(COMPILER) $(FLAGS) -o $(EXEC) $(OBJECTS) $(LIBS)

.c.o:
    $(COMPILER) -c $(FLAGS) $<

clean:
    rm *.o
    rm $(EXEC)
```

Figure B6 – Sample makefile

B8. Running Applications

Running multiple applications or simulations on various machines can be a cumbersome process. To help alleviate the difficulties encountered during this process, a series of commonly asked questions and answers have been provided to help the reader with spawning processes.

How do I run an application?

MD simulations are computationally intensive. Some simulations can take several hours, if not days, to complete. For this reason, it's a good idea to run binary files using the following syntax:

```
nohup ./<binary> <inputfile> > <outputfile> &
```

Can I log out after starting an application?

Unless you plan to sit in front of the simulation until it has completed, you need to log out. The command **nohup** starts the process and tells it to ignore hang-up signals (or logout signals). This allows the user to log out of the system without the application terminating. It is important to logout after you have finished using the workstation so that the systems are left in a secure mode.

Why do I need the ./?

<binary> is, of course, the name of the simulation application's binary. The **./** simply refers to the current directory (just like DOS/Win). When you enter the name of an executable, the operating system looks for the program only in the path described by

your \$PATH environment variable, which may not include the current directory. Therefore, when executing a program, it may be necessary to preface the executable name with ./

What is the name of the input file and what should I name it?

Most simulations require an input file. These input files should be labeled with some type of sequence or with an identifier. In the sample case, the input file was named **<inputfile>**. Any name may be selected but care should be taken to help organize the different simulations that have been constructed and completed.

How can I capture standard output to a file?

You can capture the standard output (stdout) of any program by appending **> <outputfile>** to the end of the command you type in. This redirects the output from the screen to a text file called **<outputfile>**. All the data written to standard output will go to this file except for errors, which are still displayed to stdout.

How can I return to the shell after executing an application?

Appending **&** to the end of an executed command run the command as a background process. This means that the command prompt is returned so that the user can issue additional commands, such as viewing the output of the file **tail -f <outputfile>** or checking on the status of the process.

How do I know if my simulation is still running?

You can check the status of your program (and other programs) using **ps aux**. This will display all processes currently running on the system along with other information. Other useful applications for listing the current processes are **pstree** and **top**.

What if I want to stop or terminate my simulation?

To stop or terminate a running application, use the **kill** command. To successfully use the kill command, the process id is needed. To identify the process id, execute **ps aux**, **top**, or **pstree**. Another alternative to identifying the process id is to execute **pidof <binary>**. Keep in mind that you are only allowed to kill processes that you own.

APPENDIX C

USING MDBINFMT LIBRARY VERSION 1.0.0

C1. Introduction

The purpose of this library is to give the programmer an easy to use, user-friendly, interface to MD data structures and functions for rapid development of MD simulations. In order to harness the power of the library, descriptions on the data structures and functions implemented are given. Source code for the library is available to researchers interested in expanding the library to include other data structures for different potentials, but has not been included here in case the code is moved to a commercial state.

The library itself provides a mechanism to easily access, store, and read MD simulation information during and after the simulation. The data structures can be used in two ways: as a dynamically allocated array or as a linked list. Files created and accessed using this library should be named with the extension **.md** for identification purposes. There are two important parts to the MDbinfmt library, the data structures and functions. The next two sections explain the data structures and functions, respectively. The last section in this document gives examples on how to perform common tasks that are needed during an MD simulation. To implement the library into code, download the distribution from the MD web or fileserver and make sure the library is linked during

compilation and the appropriate header files are included. All code written in the rest of this document is in italics while examples are bolded to stand out.

C2. MDbinfmt Structures

The structures defined for the MDbinfmt library are made available to help decrease the overhead while programming molecular dynamics simulations. They can be used in conjunction with the functions discussed in the next section or without. The following is a list of the defined structures available in the MDbinfmt library.

Table C1 – Table of available structures in the MDbinfmt library

Structure Name	Use of Structure
material_t	Contains material information
velocity_t	Contains velocity information
atom_t	Contains atom information
atom_ll_t	Contains atom information in linked list
grain_t	Contains grain information on atoms in array form
grain_ll_t	Contains grain information on atoms in linked list form
crystal_t	Contains crystal information on grains in array form
crystal_ll_t	Contains crystal information on grains in linked list form
bond_ll_t	Contains atom bonds information in linked list form
rk_intern_t	Temporary information used for Runge-Kutta integration

The following pages outline the structures that are defined in the MDbinfmt library. Information regarding the variable types as well as a description of what they are used for in the Morse version of the code has been provided.

Name:*material_t*

Synopsis:

```

struct material_t {
    int id;
    int pot_type;
    int crystal_type;
    float lattice_constant;
    float atomic_weight;
    float r_o;
    float d;
    float alpha;
    float cutoff_radius;
    float temp_reset_interval;
    float next_temp_reset;
};

```

Description:

This structure contains all relevant data about a material in the Morse potential as well as runtime data needed for the reset velocities functionality.

Details:

int id is a unique id given to each material.

int pot_type specifies the type of potential used.

0=Morse

int crystal_type specifies the type of crystal structure.

0=Body Centered Cubic. 1=Face Centered Cubic. 2=Hexagonal Close Packed.

float lattice_constant is the constant that specifies the crystal structure spacing (Å).

float atomic_weight is the atomic weight of the element specified (AMU).

float d is the well depth in the Morse curve or minimum potential (eV).

float r_o is the location of the well depth (Å).

float alpha is the parameter that helps to simulate the atom vibration-related to the debye freq.

float cutoff_radius specifies the distance for which atom interactions are considered (Å).

float temp_reset_interval specifies the rate the velocities should be reset (TU).

float next_temp_reset is a runtime variable that specifies the next time that the atoms of this type should have their velocities reset (TU).

Name:*velocity_t*

Synopsis:

```

struct velocity_t {
    int crystal_id;
    int type;
    float start;
    float stop;
    float x, y, z;
    float xa, ya, za;
    float xc, yc, zc;
};

```

Description:

This structure contains all the information necessary to move a crystal during a simulation.

Details:

int crystal_id is the id of crystal the velocity will be applied.

int type specifies the type of movement that will be applied to the crystal.

1=Linear.

2=Rotational.

3=Linear and Rotational.

float start is the simulation time that the movement will start (TU).

float stop is the simulation time that the movement will stop (TU).

float x,y,z are vector speeds to move the crystal in a linear type (Å/TU).

float xa, ya, za are the angular speed to move the crystal in a rotational type (rad/TU).

float xc, yc, zc specify the point about which to rotate in a rotational type (Å).

Name:

atom_t

Synopsis:

```
struct atom_t {
    int crystal_id;
    int grain_id;
    int material_id;
    struct material_t *material;
    int type;
    double x, y, z;
    double dx, dy, dz;
    double pe;
    double px, py, pz;
    struct rk_interm_t *rk_interm;
};
```

Description:

This structure contains all the information about a single atom and a pointer to a structure that stores data needed during integration.

Details:

int crystal_id is the crystal id for the atom.

int grain_id is the grain id for the atom.

int material_id is the material id for the atom.

*struct material_t *material* is a pointer to the materials structure for this

atom.

int type specifies the type of atom.

0=Boundary Atom.

1=Peripheral Atom.

2=Moving Atom.

double x, y, z is the current atom position (Å).

double dx, dy, dz is the current force on the atom (eV/angstrom).

double pe is the magnitude of the potential energy of this atom (eV).

double px, py, pz is the potential energy of this atom (eV).

*struct rk_interm_t *rk_interm* is a pointer to a structure that stores temporary values needed for integration (defined below)

Name:

atom_ll_t

Synopsis:

```
struct atom_ll_t {
    struct atom_ll_t *next;
    struct atom_ll_t *prev;
    struct atom_ll_t *next_in_cell;
    int crystal_id;
    int grain_id;
    int material_id;
    struct material_t *material;
    int type;
    double x, y, z;
    double dx, dy, dz;
    double pe;
    double px, py, pz;
    struct rk_interm_t *rk_interm;
};
```

Description:

This structure contains all the information about a single atom, as in the *atom_t* but includes additional pointers to handle the linked cell formulation.

Details:

*struct atom_ll_t *next* is a pointer to the next atom in the list.

*struct atom_ll_t *prev* is a pointer to the previous atom in the list.

*struct atom_ll_t *next_in_cell* is a pointer to the next atom in the cell.

Used in cell cutoff

int crystal_id is the crystal id for the atom.

int grain_id is the grain id for the atom.

int material_id is the material id for the atom.

*struct material_t *material* is a pointer to the materials structure for this

atom.

int type specifies the type of atom.

0=Boundary Atom.

1=Peripheral Atom.

2=Moving Atom.

double x, y, z is the current atom position (Å).

double dx, dy, dz is the current force on the atom (eV/angstrom).

double pe is the magnitude of the potential energy of this atom (eV).

double px, py, pz is the potential energy of this atom (eV).

*struct rk_interm_t *rk_interm* is a pointer to a structure that stores temporary values needed for integration.

Name:

grain_t

Synopsis:

```
struct grain_t {  
    int id;  
    int atom_count;  
    struct atom_t *atoms;  
};
```

Description:

This structure is used to organize the atoms within a crystal into separate grains.

Details:

int id is the unique id for the grain.

int atom_count is the number of atoms in the grain.

*struct atom_t *atoms* is a pointer to the array of atoms.

Name:

grain_ll_t

Synopsis:

```
struct grain_ll_t {  
    struct grain_ll_t *next;  
    struct grain_ll_t *prev;  
    int id;  
    struct atom_ll_t *atoms;  
};
```

Description:

This structure is used to organize the atoms within a crystal into separate grains. It is the same as *grain_t* except that it contains the necessary list variables.

Details:

*struct grain_ll_t *next* is a pointer to the next grain in the list.

*struct grain_ll_t *prev* is a pointer to the previous grain in the list.

int id is the unique id for the grain.

*struct atom_ll_t *atoms* is a pointer to the list of atoms.

Name:

crystal_t

Synopsis:

```
struct crystal_t {
    int size;
    double total_pe;
    double total_ke;
    int id;
    int frame;
    int grain_count;
    struct grain_t *grains;
};
```

Description:

Structure used to organize and categorize atoms in the simulation.

Details:

int size is the size in bytes that this entire crystal will use in the binary file format.

double total_pe is the total potential energy of the atoms in the crystal.

NOTE: This is not populated by MDbinfmt

double total_ke is the total kinetic energy of the atoms in the crystal

NOTE: This is not populated by MDbinfmt

int id is the unique id for the crystal.

int frame is current frame number the crystal is in.

int grain_count is the number of grains in the array.

*struct grain_t *grains* is a pointer to the array of grains.

Name:

Crystal_ll_t

Synopsis:

```
struct crystal_ll_t {
    struct crystal_ll_t *next;
    struct crystal_ll_t *prev;
    double total_pe;
    double total_ke;
    int id;
    int frame;
    struct grain_ll_t *grains;
};
```

Description:

Structure used to organize and categorize atoms in the simulation. It is the same as *crystal_t* except that it contains the necessary list variables.

Details:

*struct crystal_ll_t *next* is a pointer to the next crystal in the list.

*struct crystal_ll_t *prev* is a pointer to the previous crystal in the list.

double total_pe is the total potential energy of the atoms in the crystal.

NOTE: This is not guaranteed to be populated.

double total_ke is the total kinetic energy of the atoms in the crystal

NOTE: This is not guaranteed to be populated.

int id is the unique id for the crystal.

int frame is the frame number that this crystal is in.

*struct grain_ll_t *grains* is a pointer the the list of grains.

Name:

bond_ll_t

Synopsis:

```
struct bond_ll_t {  
    struct bond_ll_t *next;  
    struct atom_ll_t *atom1;  
    struct atom_ll_t *atom2;  
};
```

Description:

This structure defines a list of pair wise bonds between atoms.

Details:

*struct bond_ll_t *next* is a pointer to the next bond in the list.
*struct atom_ll_t *atom1* is a pointer to one atom in the bond.
*struct atom_ll_t *atom2* is a pointer to the other atom in the bond.

Name:

rk_interm_t

Synopsis:

```
struct rk_interm_t {  
    double x0,y0,z0;  
    double px0,py0,pz0;  
    double rkx,rky,rkz;  
    double drkx,drky,drkz;  
};
```

Description:

This structure simply stores temporary data needed during integration. It is used inside each atom.

Details:

double x0,y0,z0 is the initial position of the atom before integration begins.
double px0,py0,pz0 is the initial potential of the atom before integration

begins.

double rkx,rky,rkz are intermediate Runge-Kutta variables.
double drkx,drky,drkz are intermediate Runge-Kutta variables.

C3. MDbinfmt Functions

In order to populate the structures with atom information stored in the data file, several functions were created. The following table gives a list of the available functions in the MDbinfmt library. Following the table, detailed explanations have been given so that the programmer will be able to implement the functions to help decrease some of the programming overhead.

Table C2 – Table of available functions in MDbinfmt library

Function Name	Use of Function
get_size_of_crystal	Used internally for MDbinfmt
get_size_of_crystal_ll	Used internally for MDbinfmt
binfmt_read_header	Read header information from file
binfmt_read_id	Read crystal id
binfmt_next_id	Read next crystal id
binfmt_read_frame	Read current frame into structure
binfmt_get_frame_max	Get max number of frames
binfmt_read_materials	Reads and populates structure
binfmt_read_velocities	Reads and populates structure
binfmt_read_crystal	Reads and populates structure
binfmt_read_crystal_ll	Reads and populates into linked list
binfmt_read_cells	Reads cell cutoff information into cell structure
binfmt_write_header	Writes header information to data file
binfmt_write_materials	Writes materials structure to data file
binfmt_write_velocities	Writes velocities structure to data file
binfmt_write_crystal	Writes crystal structure to data file
binfmt_write_crystal_ll	Writes crystal linked list structure to data file
binfmt_write_cells	Writes cell cutoff information into cell structure
binfmt_append_crystal	Writes crystal to end of data file
binfmt_append_crystal_ll	Writes crystal to end of data file
binfmt_close	Closes the binary data file

Name:

get_size_of_crystal

Synopsis:

int get_size_of_crystal (struct crystal_t crystal);

Description:

The purpose of this function is to calculate the size of the crystal to be used in aligning the data for writing the file. This function is used internally to MDbinfmt.

Details:

struct crystal_t crystals is a pointer to the crystal.

Return Value:

Crystal size is returned.

Name:

get_size_of_crystal_ll

Synopsis:

*int get_size_of_crystal_ll (struct crystal_ll_t *crystal);*

Description:

The purpose of this function is to calculate the size of the crystal to be used in aligning the data for writing the file. This function is used internally to MDbinfmt.

Details:

*struct crystal_ll_t *crystal* is a pointer to the crystal.

Return Value:

Crystal size is returned

Name:*binfmt_read_header*Synopsis:

```
int binfmt_read_header (char *fname, float *total_runtime, float *rk_res,
                        float *output_res, int *relax_iterations);
```

Description:

The purpose of this function is to open the simulation file and read the header information from the file. This is also the function that verifies that the simulation file was created with the same version of the library.

Details:

*char *fname* is null terminated string of the file to read.

*float *total_runtime* is the total time for the simulation to run (TU).

*float *rk_res* is the integration resolution to use during the simulation (TU).

*float *output_res* is the frequency at which to output frames during the simulation (TU).

*int *relax_iterations* is the number of iteration to run the relaxation before beginning the simulation.

Return Value:

0 if successful.

BINFMT_ERROR_FILE_NOT_FOUND if the file was not found.

BINFMT_ERROR_INVALID_FILE if the file was not created using the correct version.

BINFMT_ERROR_FILE_OPEN if the file was already open.

Name:*binfmt_read_id*Synopsis:

```
int binfmt_read_id (int *id);
```

Description:

This function will read the ID of the next block of data in the file format and return it in *id*. The file must be aligned to an ID for this function to work properly.

Details:

*int *id* is the address of the variable that will contain the ID of the next block of data in the file.

Return Value:

0 if successful.

BINFMT_ERROR_NOT_ALIGNED if the file is not aligned to an ID

BINFMT_ERROR_FILE_NOT_OPEN if the file is not open.

Name:

binfmt_next_id

Synopsis:

*int binfmt_next_id (int *id).*

Description:

This function will skip over the current block of data and read the ID of the next block of data and return it in *id*. The file must be aligned to a block for this function to work properly.

Details:

*int *id* is the address of the variable that will contain the ID of the next block of data in the file.

Return Value:

0 if successful.
BINFMT_ERROR_NOT_ALIGNED if the file is not aligned to a block
BINFMT_ERROR_FILE_NOT_OPEN if the file is not open.

Name:

binfmt_read_frame

Synopsis:

*int binfmt_read_frame (int *count, struct crystal_t **crystals, int frame_id);*

Description:

This function will read all the crystals associated with the specified frame into a dynamically allocated array of crystals and return the number of them in *count*. It is suggested that this function be called immediately after opening the file to be sure that all the crystals associated with the specified frame are read. This function will read to the end of the file and therefore the file should be closed after this call.

Details:

*int *count* is a pointer to the number of crystals read from the file.
*struct crystal_t **crystals* is a pointer to the array of crystals read from the file.
int frame_id is the id of the frame for which to read all the crystals.

Return Value:

0 if successful.
BINFMT_ERROR_NOT_ALIGNED if the file is not aligned to an ID.
BINFMT_ERROR_FILE_NOT_OPEN if the file is not open.

Name:

binfmt_get_fram_max

Synopsis:

*int binfmt_get_frame_max (int *max);*

Description:

This function is useful for the post processing applications where the final frame returned is the max frame number. Keep in mind that the relaxed crystal is found at 0 and the non-relaxed crystal resides at frame -1.

Details:

*int *max* is a pointer to an integer where the maximum frame will be returned

Return Value:

0 if successful.
BINFMT_ERROR_NOT_ALIGNED if the file is not aligned to an ID
BINFMT_ERROR_FILE_NOT_OPEN if the file is not open

Name:

binfmt_read_materials

Synopsis:

*int binfmt_read_materials (int *count, struct material_t **mats);*

Description:

This function will read an array of material definitions into a dynamically allocated array of materials and return the number of them in *count*.

Details:

*int *count* is a pointer to the number of material definitions read from the file.

*struct material_t **mats* is a pointer to the array of material definitions read from the file.

Return Value:

0 if successful.
BINFMT_ERROR_WRONG_BLOCK_TYPE if this function was called when the last block ID read was not BLOCK_MATERIALS.
BINFMT_ERROR_NOT_ALIGNED if the file is not aligned to an ID.
BINFMT_ERROR_FILE_NOT_OPEN if the file is not open.

Name:

binfmt_read_velocities

Synopsis:

*int binfmt_read_velocities (int *count, struct velocity_t **vels);*

Description:

This function will read an array of velocity definitions into a dynamically allocated array of velocities and return the number of them in *count*.

Details:

*int *count* is a pointer to the number of velocity definitions read from the file.

*struct velocities_t **vels* is a pointer to the array of velocity definitions read from the file.

Return Value:

0 if successful.

BINFMT_ERROR_WRONG_BLOCK_TYPE if this function was called when the last block ID read was not BLOCK_VELOCITIES.

BINFMT_ERROR_NOT_ALIGNED if the file is not aligned to an ID.

BINFMT_ERROR_FILE_NOT_OPEN if the file is not open.

Name:

binfmt_read_crystal

Synopsis:

*int binfmt_read_crystal (struct crystal_t *crystal);*

Description:

This function reads the next crystal from the file and returns the information in the crystal structure.

Details:

*struct crystal_t *crystal* is a pointer to the crystal array definitions read from the file.

Return Value:

0 if successful.

BINFMT_ERROR_WRONG_BLOCK_TYPE if this function was called when the last block ID read was not BLOCK_CRYSTAL.

BINFMT_ERROR_NOT_ALIGNED if the file is not aligned to an ID.

BINFMT_ERROR_FILE_NOT_OPEN if the file is not open.

Name:

binfmt_read_crystal_ll

Synopsis:

*int binfmt_read_crystal_ll (struct crystal_ll_t *crystal);*

Description:

This function reads the next crystal from the file into a linked list structure and returns the information in the crystal structure.

Details:

*struct crystal_t *crystal* is a pointer to the crystal array definitions read from the file.

Return Value:

0 if successful.
BINFMT_ERROR_WRONG_BLOCK_TYPE if this function was called when the last block ID read was not BLOCK_CRYSTAL.
BINFMT_ERROR_NOT_ALIGNED if the file is not aligned to an ID.
BINFMT_ERROR_FILE_NOT_OPEN if the file is not open.

Name:

binfmt_read_cells

Synopsis:

*int binfmt_read_cells (int *count, struct cell_t **cells);*

Description:

This function will read an array of cell definitions into a dynamically allocated array of cells and return the number of them in *count*

Details:

*struct cell_t **cell* is a pointer to the cell array definitions read from the file.

Return Value:

0 if successful.
BINFMT_ERROR_WRONG_BLOCK_TYPE if this function was called when the last block ID read was not BLOCK_CELL.
BINFMT_ERROR_NOT_ALIGNED if the file is not aligned to an ID.
BINFMT_ERROR_FILE_NOT_OPEN if the file is not open.

Name:
binfmt_write_header

Synopsis:
*int binfmt_write_header (char *fname, float total_runtime, float rk_res,
float output_res, int relax_iterations);*

Description:
The purpose of this function is to open the simulation file, or create it if it is not already created, and write the header information to the file. This function also writes the correct header so that the version of the library will be identified for this file.

Details:
*char *fname* is null terminated string of the file to read.
float total_runtime is the total time for the simulation to run (TU).
float rk_res is the integration resolution to use during the simulation (TU).
float output_res is the frequency at which to output frames during the simulation (TU).
int relax_iterations is the number of iteration to run the relaxation before beginning the simulation.

Return Value:
0 if successful.
BINFMT_ERROR_FILE_OPEN if the file was already open.
BINFMT_ERROR_FILE_NOT_FOUND if the file could not be created.

Name:
binfmt_write_materials

Synopsis:
*int binfmt_write_materials (int count, struct material_t *mats);*

Description:
This function will write the number of arrays passed in by count of material definitions into the data file.

Details:
int count is the number of material definitions read from the file.
*struct material_t *mats* is a pointer to the array of material definitions to be written to the file.

Return Value:
0 if successful.
BINFMT_ERROR_FILE_NOT_OPEN if the file is not open.

Name:

binfmt_write_velocities

Synopsis:

*int binfmt_write_velocities (int count, struct velocity_t *vels);*

Description:

This function will write the number of arrays passed in by count of velocity definitions into the data file.

Details:

int count is the number of material definitions read from the file.
*struct velocities_t *vels* is a pointer to the array of velocity definitions to be written to the file.

Return Value:

0 if successful.
BINFMT_ERROR_FILE_NOT_OPEN if the file is not open.

Name:

binfmt_write_crystal

Synopsis:

int binfmt_write_crystal (struct crystal_t crystal);

Description:

This function writes the crystal to the file.

Details:

struct crystal_t crystal is a pointer to the crystal array definitions to be written to the file.

Return Value:

0 if successful.
BINFMT_ERROR_FILE_NOT_OPEN if the file is not open.

Name:

binfmt_write_crystal_ll

Synopsis:

*int binfmt_write_crystal_ll (struct crystal_ll_t *crystal);*

Description:

This function writes the crystal to the file from a linked list structure.

Details:

*struct crystal_t *crystal* is a pointer to the crystal array definitions to be written to the file.

Return Value:

0 if successful.

BINFMT_ERROR_FILE_NOT_OPEN if the file is not open.

Name:

binfmt_write_cells

Synopsis:

*int binfmt_write_cells (int count, struct cell_t *cells);*

Description:

This function writes the number of cells passed in by count from the dynamically allocated array of cells to the data file

Details:

*struct cell_t *cell* is a pointer to the cell array definitions to be written to the file.

Return Value:

0 if successful.

BINFMT_ERROR_FILE_NOT_OPEN if the file is not open.

Name:

binfmt_append_crystal

Synopsis:

*int binfmt_append_crystal (char *fname, struct crystal_t crystal);*

Description:

This function should be used to append a crystal to the end of the data file.

Details:

*char *fname* is a null terminated string containing the file name to append data.

struct crystal_t crystal contains the crystal to append to the file.

Return Value:

0 if successful.

BINFMT_ERROR_FILE_NOT_OPEN if the file is not open.

Name:

binfmt_append_crystal_ll

Synopsis:

*int binfmt_append_crystal_ll (char *fname, struct crystal_ll_t *crystal);*

Description:

This function should be used to append a crystal in linked list form to the end of the data file.

Details:

*char *fname* is a null terminated string containing the file name to append data.

*struct crystal_ll_t *crystal* contains a pointer in linked list form of the crystal to append to the file.

Return Value:

0 if successful.

BINFMT_ERROR_FILE_NOT_OPEN if the file is not open.

Name:

binfmt_close

Synopsis:

int binfmt_close ().

Description:

This function should be used to close the file after every write operation.
This will ensure that the data file is not corrupted.

Return Value:

0 if successful.
BINFMT_ERROR_FILE_NOT_OPEN if the file is not open.

C4. Using the MDbinfmt Library

Details on the operation and usage of the MDbinfmt library may not be completely clear after outlining the structures and functions. Examples on how to accomplish some important tasks have been provided.

How can the crystals be read into a linked list?

There are 4 important steps to performing this operation

1. Open the file and read in the header data. If you don't want the header data, simply pass a temporary variable to accept the value then discard it.

```
binfmt_read_header(filename, &runtime, &rk_res, &out_res,  
&relax_iterations);  
binfmt_close();
```

2. Find a crystal in the simulation file. A simple test using the block constants can be used to determine this. You may want to continue the search if your simulation contains more than one crystal.

```
binfmt_read_header(filename, &runtime, &rk_res, &out_res,  
&relax_iterations);  
binfmt_read_id(&id);  
while (id!=BLOCK_EOF) {  
    if (id==BLOCK_CRYSTAL) {  
        binfmt_read_id(&id);  
    } else binfmt_next_id(&id);  
}  
binfmt_close();
```

3. Allocate memory for the crystal. This needs to be done for each crystal that you read.

```
struct crystal_ll_t *cur_crystal=NULL;  
*crystals=(struct crystal_ll_t*) malloc(sizeof(struct crystal_ll_t));  
((struct crystal_ll_t*)*crystals)->next=NULL;  
((struct crystal_ll_t*)*crystals)->prev=NULL;  
binfmt_read_header(filename, &runtime, &rk_res, &out_res,  
&relax_iterations);  
binfmt_read_id(&id);  
while (id!=BLOCK_EOF) {  
    if (id==BLOCK_CRYSTAL) {  
        if (cur_crystal) {  
            cur_crystal->next=(struct crystal_ll_t*)
```

```

        malloc(sizeof(struct crystal_ll_t));
        cur_crystal->next->prev=cur_crystal;
        cur_crystal=cur_crystal->next;
        cur_crystal->next=NULL;
    } else {
        cur_crystal=*crystals;
    }
    binfmt_read_id(&id);
} else binfmt_next_id(&id);
}
binfmt_close();

```

4. Call the library function to actually read the data.

```

struct crystal_ll_t *cur_crystal=NULL;
*crystals=(struct crystal_ll_t*) malloc(sizeof(struct crystal_ll_t));
((struct crystal_ll_t*)*crystals)->next=NULL;
((struct crystal_ll_t*)*crystals)->prev=NULL;
binfmt_read_header(filename, &runtime, &rk_res, &out_res,
&relax_iterations);
binfmt_read_id(&id);
while (id!=BLOCK_EOF) {
    if (id==BLOCK_CRYSTAL) {
        if (cur_crystal) {
            cur_crystal->next=(struct crystal_ll_t*)
                malloc(sizeof(struct crystal_ll_t));
            cur_crystal->next->prev=cur_crystal;
            cur_crystal=cur_crystal->next;
            cur_crystal->next=NULL;
        } else {
            cur_crystal=*crystals;
        }
        binfmt_read_crystal_ll(cur_crystal);
        binfmt_read_id(&id);
    } else binfmt_next_id(&id);
}
binfmt_close();

```

How can the atoms be traversed once they have been loaded into a linked list?

1. Define traversal variables. You need some pointers to keep track of where you are in the list.

```

struct crystal_ll_t *cur_crystal;
struct grain_ll_t *cur_grain;
struct atom_ll_t *cur_atom;

```

2. Traverse through every crystal.

```
cur_crystal to the crystal of interest.  
struct crystal_ll_t *cur_crystal;  
struct grain_ll_t *cur_grain;  
struct atom_ll_t *cur_atom;  
cur_crystal=*crystals;  
while (cur_crystal) {  
    cur_crystal=cur_crystal->next;  
}
```

3. Traverse through every grain in each crystal.

```
struct crystal_ll_t *cur_crystal;  
struct grain_ll_t *cur_grain;  
struct atom_ll_t *cur_atom;  
cur_crystal=*crystals;  
while (cur_crystal) {  
    cur_grain=cur_crystal->grains;  
    while (cur_grain) {  
        cur_grain=cur_grain->next;  
    }  
    cur_crystal=cur_crystal->next;  
}
```

4. Traverse through every atom in each grain of each crystal.

```
struct crystal_ll_t *cur_crystal;  
struct grain_ll_t *cur_grain;  
struct atom_ll_t *cur_atom;  
cur_crystal=*crystals;  
while (cur_crystal) {  
    cur_grain=cur_crystal->grains;  
    while (cur_grain) {  
        cur_atom=cur_grain->atoms;  
        while (cur_atom) {  
            cur_atom=cur_atom->next;  
        }  
        cur_grain=cur_grain->next;  
    }  
    cur_crystal=cur_crystal->next;  
}
```

5. Perform whatever operation you need to on each atom. In this example the potential energy variable of each atom is set to 0.

```
struct crystal_ll_t *cur_crystal;  
struct grain_ll_t *cur_grain;  
struct atom_ll_t *cur_atom;  
cur_crystal=*crystals;
```

VITA 2

Mathew S. Lee

Candidate for the Degree of

Master of Science

Thesis: DEVELOPMENT OF A USER-FRIENDLY MOLECULAR DYNAMICS (MD)
SIMULATION SYSTEM FOR NANOMETRIC CUTTING AND TRIBOLOGY

Major Field: Mechanical Engineering

Biographical:

Education: Graduated from Putnam City North High School in 1996;
received Bachelor of Science degree in Mechanical Engineering from
Oklahoma State University in December 2000; completed the requirements
for the Master of Science degree in Mechanical Engineering at Oklahoma
State University in December 2002.

Experience: Research Assistant for Dr. Ranga Komanduri at Oklahoma State
University from January 1999 -- December 2002.

Teaching Assistant for Dr. Gary Young at Oklahoma State University from
January 2002-December 2002.