

Determining the Effects of Disorder on a System of Electrons with a Beowulf Cluster

Andrew Thomas Jones
University of Northern Iowa

October 1, 1999

Abstract

We describe the application of a Beowulf cluster to modeling the effects of disorder on a system of electrons. The details of constructing a Beowulf cluster are presented. The freely available computational libraries used in our calculation will also be described. We present the model studied and results obtained for the optical conductivity in a system with disorder.

1 Introduction

The goal of this research was to determine the effects of disorder on a system of electrons. In order to create a program that can model a large system there were several preliminary steps. First we needed to build an environment to develop and run programs in.

In our case a single computer would not be enough because we need to compute large amounts of data. So not only did we need to set up one computer for programming, but also an entire network of computers that could work together. This network of computers is known as a Beowulf cluster, or just as a Beowulf.

Once the Beowulf is complete we then need to write programs to perform a preliminary analysis of a particular model for a system of electrons in a disordered environment.

2 What is a Beowulf?

When asking what a Beowulf is one should first ask what a supercomputer is. A supercomputer is a machine with lots of processing power. Wouldn't it be safe to assume that a room full of N computers where each computer contains a processor with a given amount of processing power could actual be considered a single machine with N times the processing power of a single machine if there was a way to combine them? In 1994 Thomas Sterling and Don Becker [1] implemented this idea with PC hardware, creating what they called a Beowulf.

A Beowulf is a group of computers that contain some method of communicating with each other. A typical Beowulf is a cluster of PC's networked together with 100 Mbps Ethernet. These machines also typically run a Unix-like operating system; most commonly Linux is used.

When programming a Beowulf the fact that your code needs to run on multiple remote CPUs must be considered. However, it is not as difficult as one might expect due to a large amount of libraries and software written that make it easy to distribute processes. In our research we became familiar with MPI, Message Passing Interface, and HPF, High-Performance Fortran.

3 The Hardware

3.1 A Single Node

This project was certainly started correctly when thirteen 500 MHz Pentium III computers were delivered. Each machine is also armed with 128 MB of RAM and

most importantly a 10/100 Ethernet adapter. Since these machines are commodity boxes that produce over 300 Mflops¹ each we are conforming to the specification that states to be a Beowulf it must be cheap and powerful.

3.2 Hubs and Switches

One of the first things we realized when we were ready to start this project was the fact that we had very nice computers that contained 10/100 Ethernet adapters, but they were all hooked into a 10 mbit hub. We knew that we would not be getting nearly the performance we could get if we were going to run on a slow hub rather than a fast switch. So we requested a 100 mbit switch.

We did some testing before and after the new switch came in with a utility called netperf[2]. Netperf can be used to perform many network tests. We chose to see the throughput of a stream of both UDP and TCP packets. There are two runs shown below for the switch. One with the network busy and one without. You can see that there is only a small difference between throughput of a busy network and a non busy network. I wish I had results from a busy hub because since unlike a switch a hub does not direct traffic and I assume the difference between busy and non busy would be much larger. The throughput is given in 10^6 bits/sec.

	UDP_STREAM	TCP_STREAM
10 mbit hub	9.53	6.89
10 mbit hub (busy)	—	—
100 mbit switch	94.18	90.84
100 mbit switch (busy)	87.24	73.55

4 Linux

Linux is an open source project started by Unix hackers as a hobby. It was originally started by Linus Torvalds who was student in Finland looking to improve Minix which was another Unix-like operating system. The reason Linux is so robust is because it is open source which allows ambitious gurus to find and fix bugs. Open source software is simply software where the programmers distribute the actual source code they wrote. It is commonly freely distributed.

¹The whetstone benchmark software proved the PIII 500 to produce 362 Mflops

4.1 Why use Linux?

Why use Linux for a Beowulf? By using a Unix-like operating system we gain a large collection of well-established networking tools. To list a few, which will be described more later, it supports such things as rsh and NFS which make remote use easy.

It is also a good choice for running CPU intensive jobs because the kernel is quite stable and manages memory well. Of course, another reason to use Linux is that it is free. Unlike some operating systems where you must purchase an expensive license for every machine running it, with Linux you can have a cluster of any size all running Linux for free.

Why use Linux for a development platform? Linux, being a Unix-like operating system, has the advantage of using the many programs of the GNU project. This means that free editors and compilers are available for it. In fact if you use a Linux distribution like Red Hat, those compilers and utilities are quite easy to install and you can have a development platform ready for use in a matter of minutes.

5 Useful Networking Software

As mentioned earlier, Linux, like all Unix-like operating systems, supports networking tools to make remote use very easy, even transparent to the user. There are several utilities for managing a network, there are even quite a few available for specifically managing a Beowulf. We only used the most common tools that are available with Linux.

5.1 rsh and Friends

The two main remote use applications are telnet and rsh. Telnet allows you to login to the remote machine and returns a shell that allows you to remotely control it by use of commands on the command line. You may even use the GUI (graphical user interface) programs by simply sending the display of the remote machine to your local machine.

Rsh is quite similar to telnet, it too will return a shell. Or if you send a command along with your rsh call then it will execute the command on the remote host and return to the local machine. Rsh is very handy for starting a process up on several remote machines simultaneously. Rsh has a sister called rcp. Rcp allows you to copy a file from one machine to another as easily as coping it from from directory to another on a single machine. Here are some example calls of these two important commands.

```
[drewj@cec11 drewj]$ rsh cec00 uname -n
cec00
[drewj@cec11 drewj]$ rcp cec00:/etc/HOSTNAME .
[drewj@cec11 drewj]$ cat HOSTNAME
cec00
```

If you set up the systems to allow root, the superuser, to rsh to all the nodes then it is quite easy to install software on all of them simultaneously with a simple script.

I created such a script which I named 'doall'. This script takes two sets of arguments. First it took a list of machine numbers and second it took the list of commands. Here is the synopsis;

```
[drewj@cec11 drewj]$ doall -h

doall 'cmd(s)'          -command(s) run on all nodes
doall 2-4 'cmd(s)'      -command(s) run on nodes 2,
3, 4
doall 2,6,9 'cmd(s)'    -command(s) run on nodes 2,
6, 9
```

You must use quotes ' ' around the command(s).

Here is an example of how one might use it to install the free Fortran compiler from the GNU project on all the nodes.

```
[drewj@cec11 drewj]$ doall 'rpm -i egcs-g77-1.1.2-
12.rpm'
```

5.2 NFS

From the earlier example in the last section it would appear that the file egcs-g77-1.1.2-12.rpm is in my home directory on every machine. It could be distributed around with rcp, but to save time it is easiest to use NFS, Network File System, to distribute certain directories around to all nodes all the time from a central location such as a master node or a file server. We have a master node that holds our home directories and it allows NFS mounts from any of the nodes. With Unix and Linux,

NFS allows local and remote file systems to blend transparently. If I login to cec00 I get the same home directory as if I login to cec10 or any other node and it looks like a local home directory, however those files in my home are only local to the master.

While NFS is very handy it does generate network overhead. In a Beowulf you want your network to be as clear as possible so your code passes messages as quick as possible when you run it. To avoid unneeded network overhead we use autofs which is a program that watches for needed NFS mounts and automatically mounts them on demand. If a mount has not been used for a certain amount of time then it unmounts that directory thus lessening the network load. This is very necessary if there are several users with homes. For example if only one person needed the cluster but fifty home directories were mounted on all the nodes then you would have a lot of wasted bandwidth.

The nodes also share some directories from themselves to the others, they are `/mnt/data` and `/mnt/cdrom`. `/mnt/data` is a large partition that is shared because some programs may like to store partial solutions on remote nodes during execution. `/mnt/cdrom` is shared because that allows us to sit down at any node, insert a cdrom, and access it from any other node remotely.

6 Useful Programming Software

With any Unix-like operating system there is a wealth of compilers, editors, and libraries available, because Unix has been the primary development platform for several years. There are even free versions of these tools. In our case we used the free Fortran 77 compiler, that is distributed under the GNU Public License (GPL), called `g77`. We also used the standard text editor distributed with most every Unix called `vi`. As for the libraries in my specific problem I needed to calculate eigenvectors and eigenvalues to of a matrix and there happens to be a parallel linear algebra package available called Scalapack. Scalapck also requires a few other libraries to be included.

6.1 Pros and Cons of Fortran and of `g77`

This was my first experience with Fortran 77 and I was pleased to see that it was quite easy to convert mathematical equations into functions. Containing a built-in complex data-type was also a great time saver. I was disappointed with the lack of the ability for dynamic allocation of arrays in `g77` and the lack of the ability to use recursion.

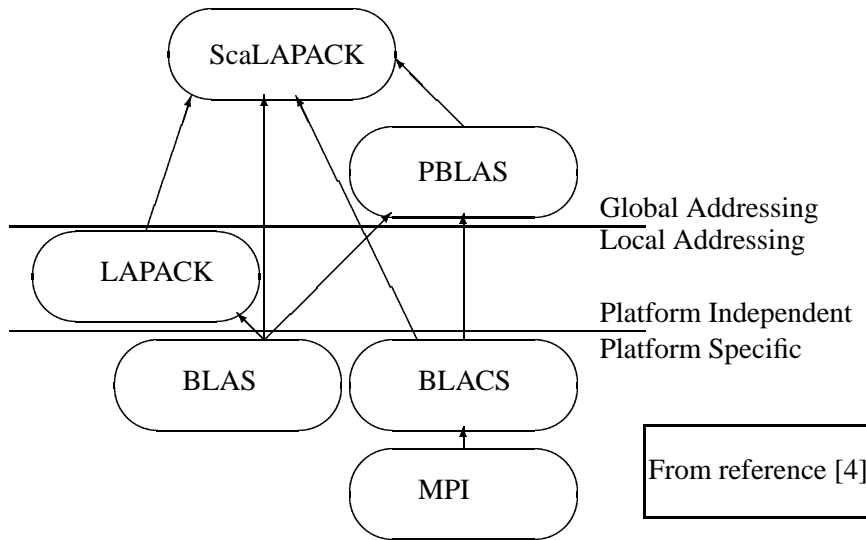
This was also my first experience with `g77`. I believe `g77` is an ‘ok’ compiler.

I am thrilled that there is a free Fortran compiler, but I believe it still needs work. Compiler warnings and errors were either not very helpful or non-existent, which caused hours of adding print statements in code to track down bugs. I also heard from a talk[3] at Linux Expo '99 that through Toon Moene's benchmarks they discovered that `g77` is quite inefficient and it was more efficient to write the Fortran code, use `f2c` (utility for converting Fortran to C), then compile with `gcc`, the GNU C compiler.

6.2 The Joy of Scalapack

Scalapack is a parallel linear algebra library. If you link to this library you are granted numerous linear algebra routines. When these routines are executed they are broken into several processes that can be distributed on all the CPUs in the cluster at the same time. So if you have a very large matrix the problems are solved faster because each machine only has to work with a percentage of the whole matrix. For example if we have a 64×64 matrix and we divide the problem over four processors then each processor only has to work with a 32×32 matrix.

Scalapack was designed to sit on top of other libraries to be more abstract and to allow users to only link to the routines they need. There are actually four or five other libraries one would need if they wish to use Scalapack. They are Pblas, Blas, Blacs, and some library of message passing primitives. Pblas is a collection of parallel basic linear algebra subprograms. Some functions in Pblas call on Blas. Blacs is the basic linear algebra communication subprogram. We chose to use MPI, Message Passing Interface, for our message passing primitives. Blas, Blacs, and MPI are all platform specific while Pblas and Scalapack are easily portable. The possible fifth one is Lapack. Lapack is the single process version of Scalapack that Scalapack is based upon. Some routines in Scalapack may depend on Lapack, but the ones we used did not. See the diagram below.



Getting these libraries to work together in harmony proved to not be an easy task. I feel I should note though that every problem I had was really directly related to the poor documentation on Scalapack which is available from <http://www.netlib.org/scalapack>. Some of the things that I wished were better documented are the fact that a patch must be applied to Blacs before it can be used with mpich (the MPI package we used) and how to better edit the Bmake.inc file before building Blacs.

As mentioned briefly above the single processor version of Scalapack called Lapack also contains a large collection of linear algebra routines. We had no problem installing Lapack or Blas because they can be downloaded in rpm format from <ftp://metalab.unc.edu>. Rpm format is the installation format Red Hat Linux uses so it is very easy to install on to Red Hat.

7 Cloning

In order to get all the software we needed on all the machines in an efficient way I installed and configured everything we wished to use on a single machine. Then to save time I cloned that machine twelve times to produce thirteen identical machines.

At first I was going to use a piece of software called ghost which I had heard was good for cloning. However since ghost is proprietary I could not get a copy. So in order to clone with tools at hand I used a utility named dd[5] that is designed to convert and copy a file and I converted the partition containing Linux and all the software into a single file. Then to save time copying it across the network I used another utility called gzip and I compressed that file down to almost half the size.

Now I created a Linux boot disk that had networking configuration tools, NFS support, and a that handy utility called dd. I booted each of the other twelve boxes off this floppy, set up networking and used NFS to mount a shared directory on the already complete box that held the file I created. Then I uncompressed the file and used dd to turn the file into an image on a pre-made partition.

The machine was instantly bootable to Linux and only needed some networking parameters to be modified before it was ready to participate in a Beowulf cluster.

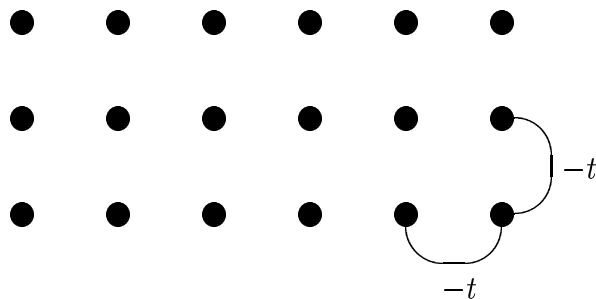
8 A Simple Model for Electrons in Disordered

8.1 The H-matrix

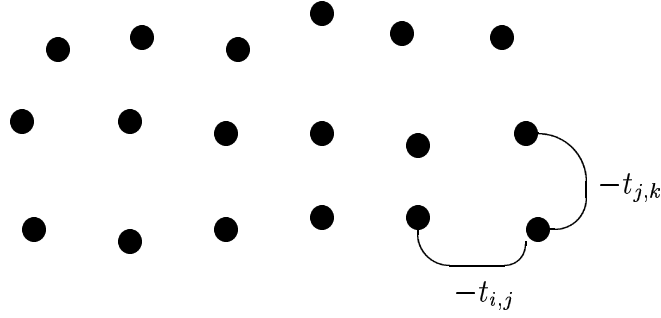
The H-matrix is a $n \times n$ matrix, where $n = n_x \cdot n_y \cdot n_z$ is the number of sites in an atomic lattice. This matrix determines how the state vector for an electron changes with time and the state describes the probability for finding an electron at a particular atomic site. In our model this matrix is defined as

$$H(i, j) = \begin{cases} \epsilon_d(i) & : i = j \\ t_{i,j} & : \text{if } i, j \text{ are nearest neighbors} \\ 0 & : \text{if otherwise} \end{cases}$$

where $\epsilon_d(i)$ is the local electrostatic potential and $\frac{1}{\hbar}t_{i,j}$ is the hopping frequency, the rate at which an electron attempts to move from site i to site j . \hbar is Planck's constant. In an ordered atomic lattice $\epsilon_d(i)$ is the same for all atomic sites and $t_{i,j}$ is the same for all nearest neighbor pairs, i, j . A look at how electrons would hop due to an H-matrix that describes a system that is ordered looks like the following:



If the atomic lattice is disordered it may appear more like this:



The values $t_{i,j}$ and $\epsilon_d(i)$ for a disordered lattice are read in from a program that determines their values based in a particular random configuration of the atomic sites.

What we are really interested in is finding the probability that an electron will be at a given site. It is true that the probabilities of finding an electron at a given site in an ordered system are all equal because all sites are evenly spaced. However in a disordered system we need to use the H-matrix to determine what the probability is.

The H-matrix can be diagonalized to gain the state vectors and the state is used for finding the probability of finding an electron at a given site. Diagonalization gives you the eigenvectors, Ψ , and the energies for each vector, the eigenvalues ϵ . This is where Lapack and Scalapack were needed to complete the computations.

8.2 Localization as a Function of Disorder

The chemical potential, μ , is determined from the density of electrons in the lattice. The electrons most responsible for electrical conduction have energies close to μ . So we will plot the probability density for an electron whose energy is closest to μ . If the probability is large only for a small number of sites, we say the electron is localized.

8.3 Impact on Optical Conductivity

Now with the information obtained from diagonalizing H , we can obtain the optical conductivity. In this model a system with low disorder has high conductivity at low frequency (metal-like) where as a system with higher disorder is expected to have conductivity at high frequency (insulator-like). Changing a conductor to an insulator by increasing the disorder is called the “Anderson transition”. The optical conductivity, $\sigma_1(\omega)$ can be computed as follows where ω is the frequency.

$$\sigma_1(\omega) = -2 \sum_{l, l'} \frac{f(\epsilon_{l'}) - f(\epsilon_l)}{\omega[(\omega + i\delta) - (\epsilon_{l'} - \epsilon_l)]} F^x(l, l') F^x(l', l)$$

where

$$f = \frac{1}{e^{(\epsilon_l - \mu)/T} + 1}$$

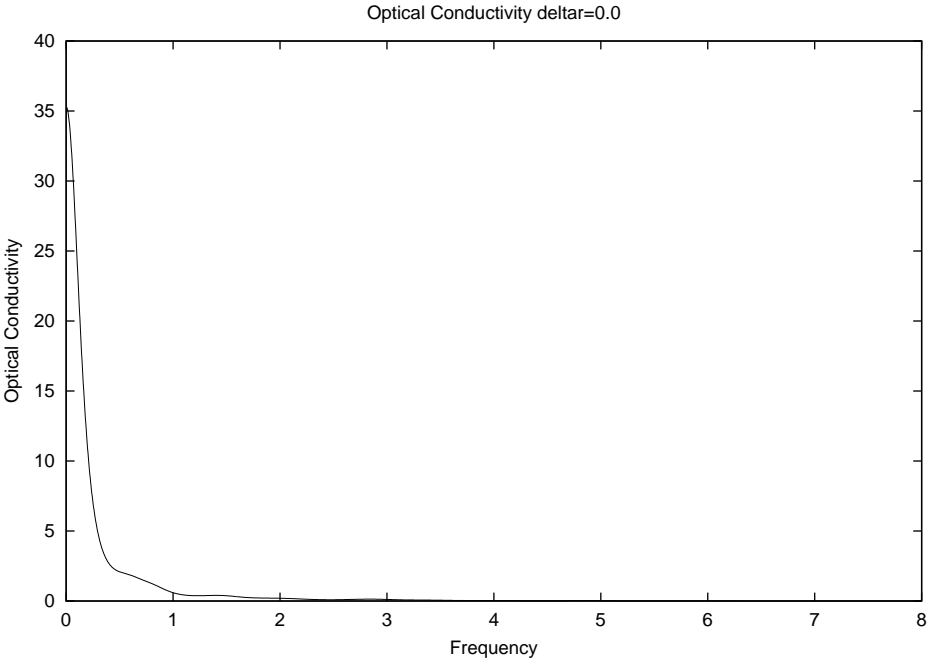
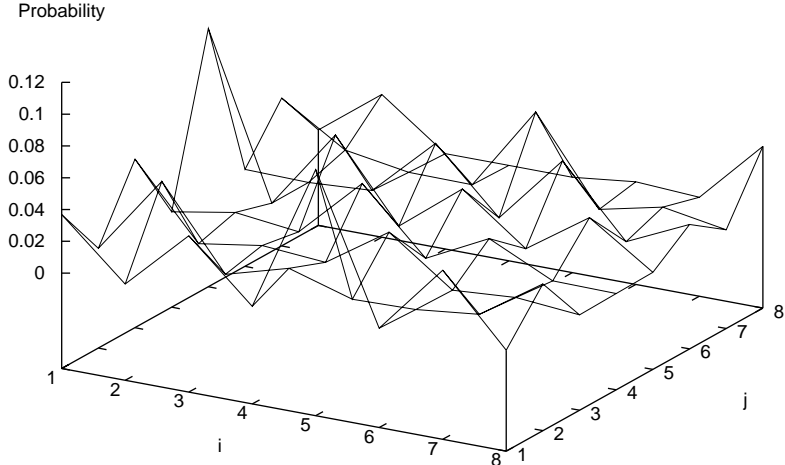
and

$$F^x(l, l') = \sum_j H(j, j + \hat{x}) \Psi_{l'}(j) \Psi_l(j + \hat{x}) - H(j, j - \hat{x}) \Psi_{l'}(j) \Psi_l(j - \hat{x})$$

The results are as follows:

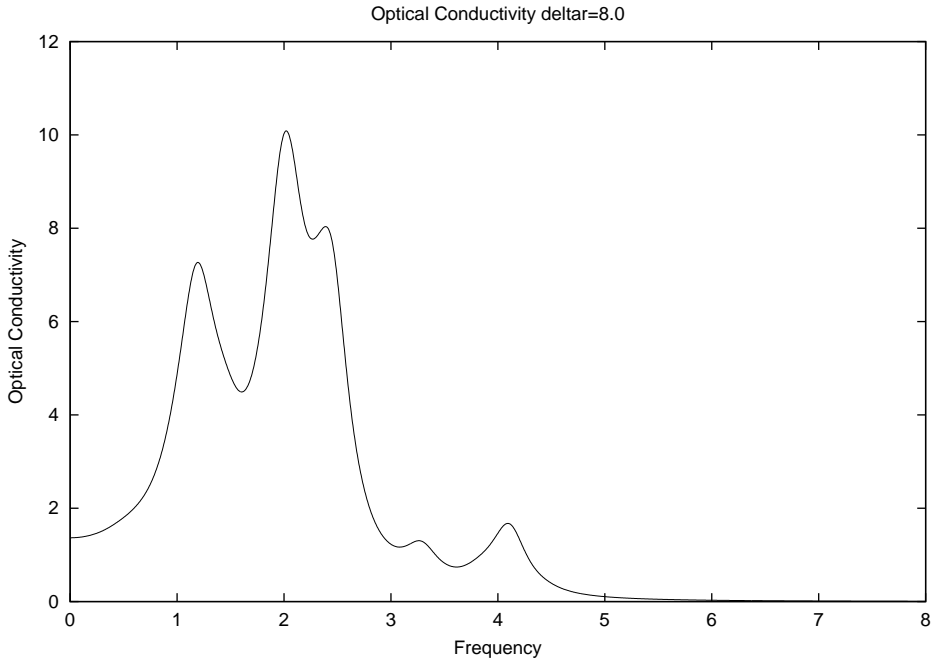
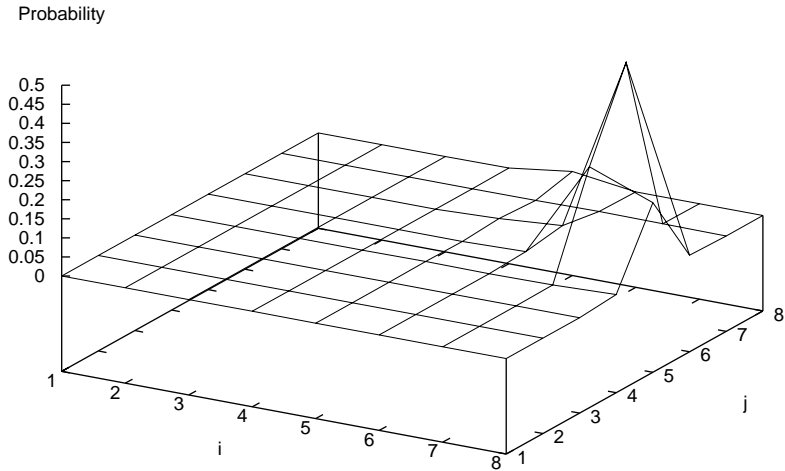
8.4 No Disorder, $\delta = 0$

Probability of Finding Electrons as a Function of Position and Disorder, $\delta = 0.0$



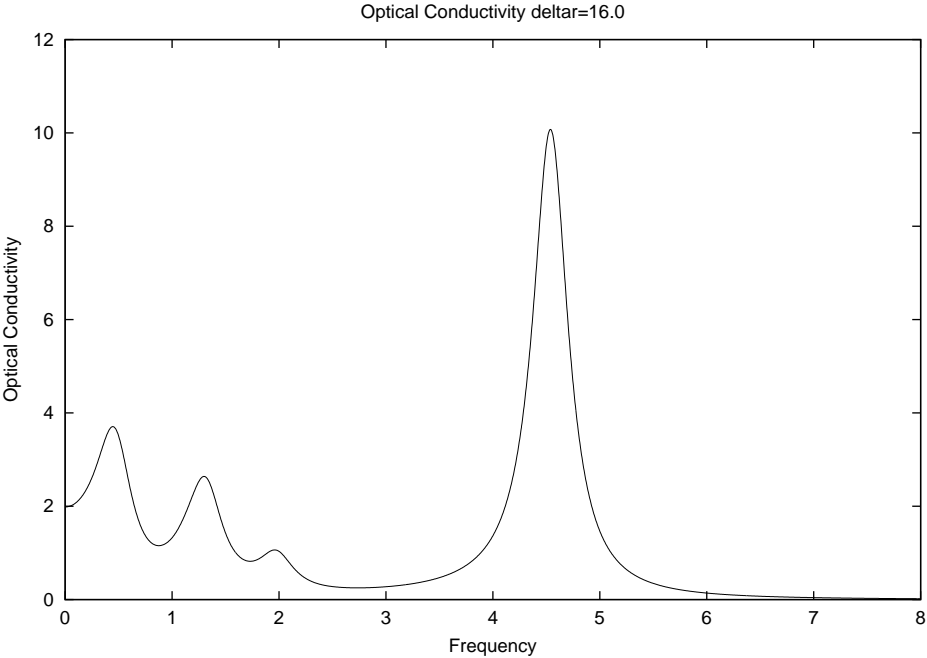
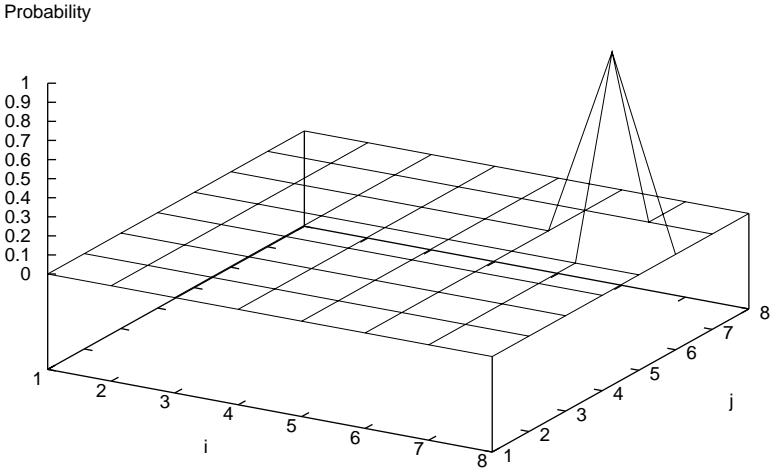
8.5 $\delta = 8$

Probability of Finding Electrons as a Function of Position and Disorder, $\delta=8.0$



8.6 $\delta = 16$

Probability of Finding Electrons as a Function of Position and Disorder, $\delta=16.0$



8.7 Parallel Solution for Optical Conductivity

Since creating the graphs depicted above for optical conductivity required multiple runs of the same program with different arguments within a set range I found this problem to be perfectly parallelizable. This means that I could distribute an even load over every processor. The way this was accomplished was by writing a short shell script that simply divides the range by 15 for the 15 processors and then rsh's to each machine and runs through the range of arguments it as been allotted. The master then takes these subsets of the results and combines them before graphing. This technique proved to increase the run time by nearly 15 as it should.

9 Summary

I believe this project was successful because we now have the framework lane for more, larger scale, computation projects. I wish we would have had time to run the code I wrote with larger lattice sizes. The graphs depicted above are only for 8×8 lattices which leads to 64×64 matrices. I enjoyed this project because it is very exciting to see a program that usually takes a very long time on a single machine complete execution much faster.

References

- [1] *The Beowulf Project at GSFC*. Available on www.beowulf.org.
- [2] *Netperf networking benchmarking*. Available on www.netperf.org.
- [3] Toon Moene. *Predicting the Future with Linux - Numerical Weather Forecasting*.
- [4] *The Scalapack Poster* Available on www.netlib.org.
- [5] The dd man pages.
- [6] Robert G. Brown *So, You Want to Build a Beowulf? Parts I of II*.
- [7] Thomas L. Sterling, John Salmon, Donald J. Becker, and Daniel F. Savarese. *How to Build a Beowulf*. The MIT Press, 1999
- [8] Douglas Eadline. *Cluster Quick Start(DRAFT)*. Available on www.xtrememachines.com.
- [9] Anderson, Bai, Bischof, Demmel, Dongarra, Du Croz, Greenbaum, Hammarling, McKenney, Ostrouchov, and Sorensen *LAPACK Users's Guide* Society for Industrial and Applied Mathematics, 1992
- [10] *blacs.errata*. Available on www.netlib.org.
- [11] William Gropp, Ewing Lusk, and Anthony Skjellum. *USING MPI Portable Parallel Programming with the Message-Passing Interface*. The MIT Press, 1994