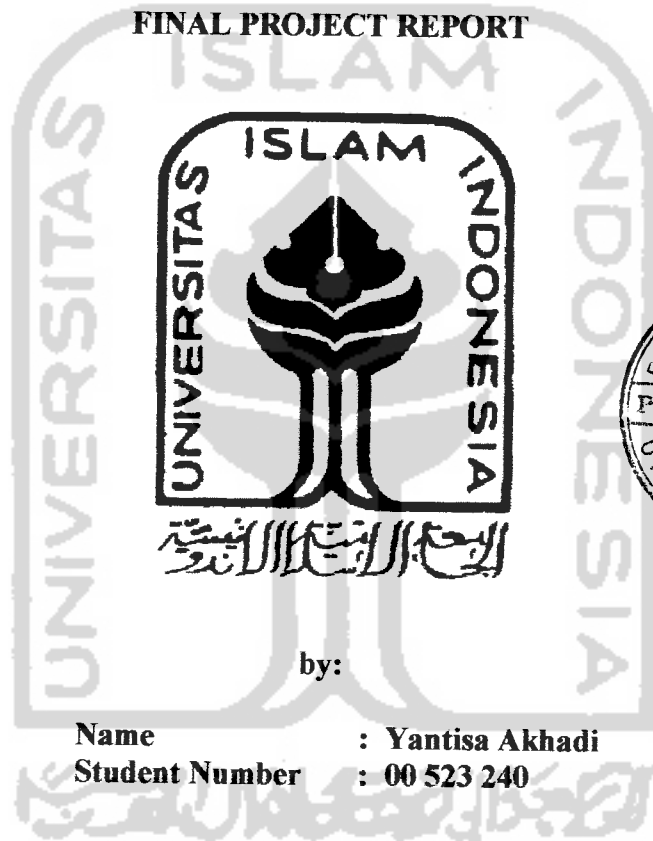


SUPERVISOR VALIDATION PAGE

APPLICATION OF DISTRIBUTED COMPUTING WITH OPENMOSIX ON COMPLEX NUMERICAL METHOD PROBLEM

FINAL PROJECT REPORT




by:


Name : Yantisa Akhadi
Student Number : 00 523 240

Yogyakarta, March 16, 2006

Supervisor I


(Fathul Wahid, ST. M.Sc)

Supervisor II


(Wawan Indarto, ST)

**GENUINENESS STATEMENT
OF FINAL PROJECT REPORT RESULT PAGE**

Hereby the undersigned,

Name : Yantisa Akhadi
Student Number : 00 523 240

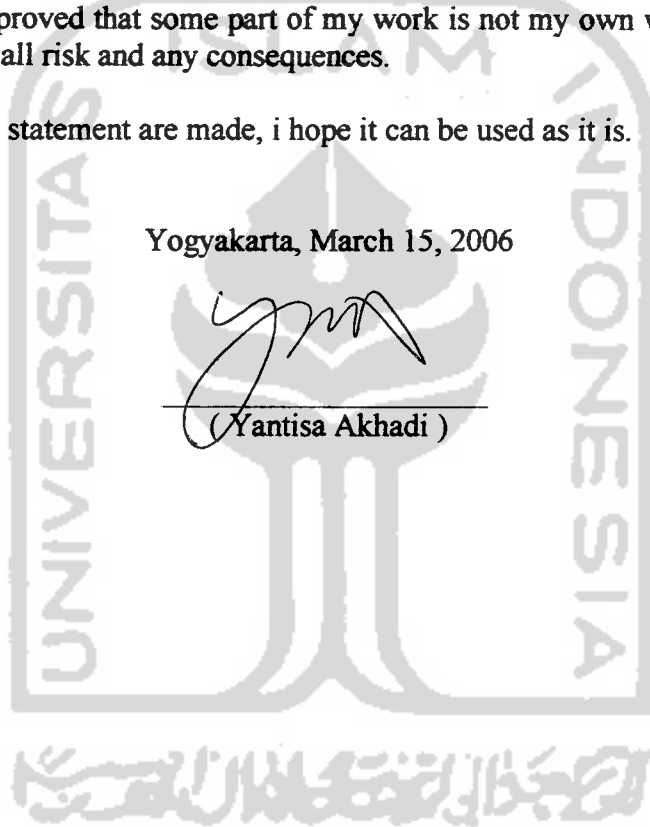
State that all component and content in this Final Project Report is my own work. If in other day proved that some part of my work is not my own work, then i will ready to accept all risk and any consequences.

That is how my statement are made, i hope it can be used as it is.

Yogyakarta, March 15, 2006



(Yantisa Akhadi)



EXAMINER VALIDATION PAGE

**APPLICATION OF DISTRIBUTED COMPUTING
WITH OPENMOSIX
ON COMPLEX NUMERICAL METHOD PROBLEM**

FINAL PROJECT REPORT

By:

Name : Yantisa Akhadi
Student Number : 00 523 240

Was Defended Before the Board of Examiners as Partial Fulfillment of the
Requirements to Obtain the Bachelor's Degree in Department of Informatics
Faculty of Industrial Technology Universitas Islam Indonesia

Yogyakarta, March 25, 2006

Examiner Team

Fathul Wahid, ST., M. Sc
Chief

Taufiq Hidayat, ST., M.Sc
Member I

Wawan Indarto, ST
Member II



Acknowledge,
Faculty of Industrial Technology
Universitas Islam Indonesia

H. Bachrun Sutrisno, M.Sc

DEDICATION



*To My Lovely Mother and Father,
for all the understanding, forgiveness and caring*

*To My Lovely Sister, Brother in Law, and Nephew,
for their support and care*

*To those who teach me lesson of life and knowledge,
may Allah repent you greatly*

*To my future wife, whoever you are,
i miss you a lot before i meet you dear*

*To all who appreciate knowledge,
let's spread what we know*

MOTTO

مَا وَدَّعَكَ رَبُّكَ وَمَا قَلَىٰ

*Tuhanmu tiada meninggalkan kamu dan tiada (pula) benci kepadamu
(QS. 93:3)*

*Kutinggalkan untukmu dua buah perkara, kamu tidak akan tersesat
selama kamu berpegang kepada keduanya, Kitabullah dan Sunnah*

*Rasul
(HR. Muslim)*



ACKNOWLEDGMENT



Assalamu' alaykum warrahmatullahi wabarakatuh

Praise to Allah SWT, for His mercy and blessing, so the compiler could finally finish this report. Shalawat and Salam to our beloved prophet Muhammad SAW, his family and his friends, who gives a lot of knowledge that guide us into the light.

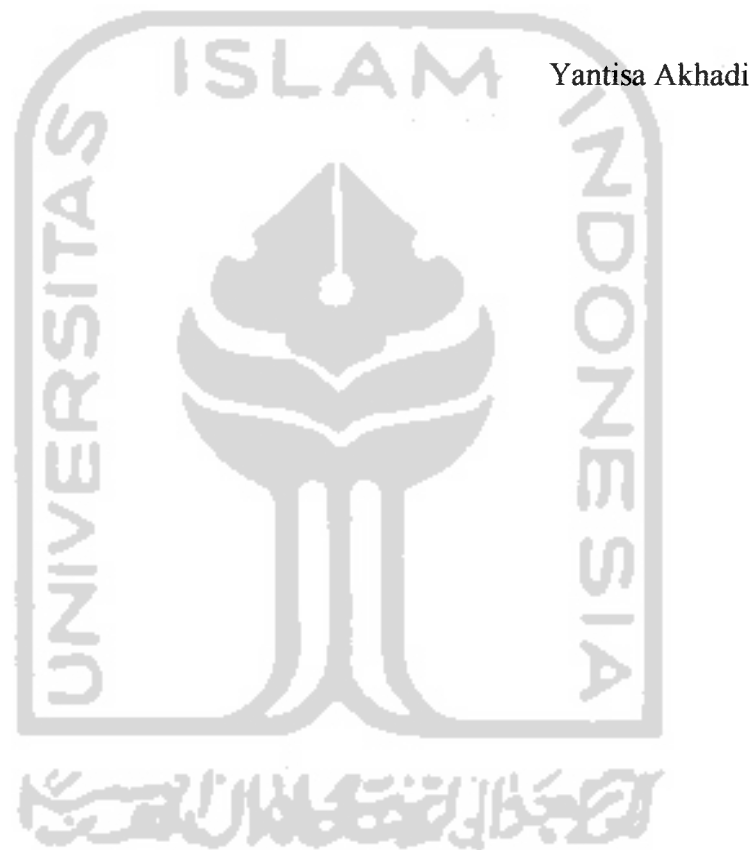
On the making and finishing this report, compiler would give appreciation to the following person, although they may not read this:

1. Mr. Ir. H. Bachrun Sutrisno, M.Sc, as the Dean of Faculty of Industrial Technology, Universitas Islam Indonesia Yogyakarta, who allowed the compiler to do this research and compile it.
2. Mr. Wawan Indarto, ST., as the Supervisor an the Head of Network Laboratory, for the critics, patience, and guidance while this report is still in idea until it finally come complete.
3. Mr. Fathul Wahid, ST., as the Supervisor, for comments and critics to improve the content of the research, especially during progress report session.
4. Mr. Yudi Prayudi, ST, as the Head of Informatics Departement, Faculty of Industrial Technology, Universitas Islam Indonesia, Yogyakarta, who gave ease to compiler during this research report compilation.
5. Mr. Supriyono, Mr. Erwin, and Mrs. Lizda for their support during the making of this report.
6. All my comrades+ on Pandega Karya 3C, Gesit, Doyo, Zaki, Agil, Anto and mas Indra, hope you all can passed soon.
7. All my colleagues at Computer Network Laboratory, Dini, Tetra, Jarwo, AP, Ryan, Nanda, Urip, Al, Iwell, Arif, and Yuda, finish your study guys, now i'm allowed to say that :p.
8. All my friends at Debating World, my teammate and soulmate Iponk, Adit, Nanang, Adjib, Adib, Wulan, Titis, Yuke, Umi, Aldi, Dilla, Ninus, Dhini, Adiz, Nisa-chan, Angga, Nunung, Tia, Kiki, Ade, Ratih, Yurisch, Rani, Qisthi, Dida, Rina, Tiara, Desi, Prima, Ester, Santi and Windi, not to forget all people who join JDF and Indodebaters.
9. All of my kids at Muhi, Delayota and PP Sunan Pandanaran, El, Nella, Chacha, Yayang, Icha, Winda, Asdi, Dewi, Dina, Agung, thanks honey
10. All my friends at KPLI Jogja, special to Iwan, Ryan, Willy SR, Jaya, Fathir, Yagus.

11. All my colleagues at Pasifik Satelit Nusantara, Wisnu, Daffe, mas Miko, mbak Ani, Mr. Zai, Mr. Iman, Mr. Iwan, Mr. Sutri, and more.
12. Everyone who support the compiler that cannot be mentioned one by one.

May Allah repent all of your kindness in this life and afterlife. Further contact can be delivered to iyan31@yahoo.com.

Yogyakarta, March 15, 2006



ABSTRACT

Numerical method is being widely used in almost every major engineering field. But its use also increase the demand for faster computation power. Few solution to those power is supercomputer. But the price of supercomputer creates big burden. Other alternatives come in from distributed computing world. Where cluster can be built using commodity hardware.

One of known implementation is using openMosix, a load balancing cluster, that even can be implemented on a live CD. This research tries to answer how is the performance on this cluster on complex problem of numerical method, especially on the field of big linear equation system, where there are millions or even billions of operation needed to be done.

The results of the research reveals that on heterogeneous environment, where there are computers with better specification would significantly reduce time needed to solve the problem compare to single computer and homogeneous environment.

Keywords : distributed computing, numerical methods, linux, openMosix, load balancing cluster.

TABLE OF CONTENTS

Title Page.....	i
Supervisor Validation Page.....	ii
Genuine Statement.....	iii
Examiner Validation Page.....	iv
Dedication.....	v
Motto Page.....	vi
Acknowledgment.....	vii
Abstract.....	ix
Table of Content.....	x
List of Tables.....	xiii
List of Figures.....	xiv
CHAPTER I PREFACE.....	1
1.1 Background.....	1
1.2 Problem Definition.....	2
1.3 Scope.....	2
1.4 Objectives.....	2
1.5 Benefits.....	3
1.6 Hypothesis.....	3
1.7 Methodology.....	3
1.7.1 Collecting Data.....	3
1.7.2 Design.....	4
1.7.3 Implementation.....	4
1.7.4 Testing.....	4
1.8 Report Outline.....	5
CHAPTER II THEORY.....	6
2.1 Numerical Method.....	7
2.1.1 General Introduction.....	7
2.1.2 Definition.....	8
2.1.3 Characteristic of Numerical Method.....	8
2.1.4 Application of Numerical Method.....	10
2.1.5 Area of Study.....	11
2.1.6 Computational Complexity Theory.....	15
2.2 Distributed Computing.....	16

2.3 Computer Cluster.....	20
2.3.1 Types of Computer Cluster.....	20
2.3.2 Cluster and Supercomputer.....	22
2.4 openMosix.....	22
2.4.1 Overview.....	22
2.4.2 Component of openMosix.....	23
2.4.3 openMosix Internal.....	24
2.4.4 Pros and Cons using openMosix.....	25
2.4.5 Linux Distribution with openMosix.....	27
CHAPTER III SOFTWARE REQUIREMENT.....	29
3.1 Analysis Method.....	29
3.2 Analysis Result	29
3.2.1 Data Analysis.....	29
3.2.2 Software Requirement.....	30
3.2.3 Hardware Requirement.....	31
CHAPTER IV SOFTWARE DESIGN.....	32
4.1 Design Method.....	32
4.2 Design Result.....	32
4.2.1 Network Design.....	32
Figures 4.1 Network Design.....	33
4.2.2 Software Design.....	33
4.2.2.1 Big Linear Equation System using Matrix Inverse.....	33
4.2.2.2 Big Linear Equation System using Backslash Division.....	35
CHAPTER V SOFTWARE IMPLEMENTATION.....	37
5.1 Scope.....	37
5.2 Implementation.....	37
5.2.1 Hardware Implementation.....	37
5.2.2 Software Implementation.....	40
5.2.2.1 openMosix Implementation.....	40
5.2.2.1 Numerical Method Software Implementation	43
CHAPTER VI SOFTWARE TEST & ANALYSIS.....	45
6.1 Performance Test.....	45
6.2 Result and Analysis.....	45
6.2.1 Base System Test.....	46
6.2.2 Homogeneous System Test.....	48
6.2.3 Heterogeneous System Test.....	50

CHAPTER VII CONCLUDING REMARK.....56
 7.1 Conclusions.....56
 7.2 Suggestions.....57
REFERENCES.....59



LIST OF TABLES

Table 6.1 Inverse Matrices on Base System

Table 6.2 Backslash Division on Base System

Table 6.3 Inverse Matrices on Homogeneous System

Table 6.4 Backslash Division on Homogeneous System

Table 6.5 Inverse Matrices on Heterogeneous System

Table 6.6 Backslash Division on Heterogeneous System



LIST OF FIGURES

Figures 4.1 Network Design

Figures 5.1 openMosix on Quantian Linux

Figures 6.1 KSysGuard Applet

Figures 6.2 Process Migration on Homogeneous System

Figures 6.3 Process Migration on Heterogeneous System

Figures 6.4 Overall Comparison on Inverse Matrices

Figures 6.5 Overall Comparison on Backslash Division



CHAPTER I

PREFACE

1.1 Background

Numerical method nowadays gains its popularity as a quick way to solve various problems in almost every aspect of science. From engineering, medics, chemistry and many more. This development is mainly supported by limitation of analytical method which is hard or near impossible to solve a complex problem, others aspect such as the growth of processing power of a computers also give significant contribution toward this.

But then, also known that special characteristic on solving mathematic problem using numerical method is there are many arithmetic operation and it is done repeatedly (over and over again). The more complex the problem that we are going to solve then the more operation needed thus more time will be sacrificed, due to limitation of a computer processing power, even in present day.

There is one quick way of solving the problem is by using super computer. But this way gives another problem which is how expensive a super computer can be, and even if available, still only limited number of people allowed to access it.

1.2 Problem Definition

From those background stated above we can take two major problems, which are :

1. A complex numerical method problem needs a lot of computer processing power and results a long time to solve a problem.
2. There is possibility of solving a complex numerical method problem using super computer to make faster execution but it gives another problem on cost and opportunity to use a such kind of machine.

1.3 Scope

The scope of this research are :

1. The process of computation is focused on complex numerical method problem, in this case to solve big linear equation systems
2. The optimization is done on the process of computation not on algorithm or source code of the program.

1.4 Objectives

The main objectives of this research is usage of distributed computing system with openMosix to reduce the time needed to solve complex numerical method problems.

1.5 Benefits

There are several benefits that can be gained from this research which is :

1. To give understanding on how actually powerful computation can be gained from group of regular computers.
2. To give overview on a method to accelerate computation process especially on complex numerical methods problem.
3. To show comparison of benefits between using single computation and distributed computing ones.

1.6 Hypothesis

This research tries to prove assumption that process of solving complex numerical method problems can be accelerated using distributed computing system, in this case openMosix.

1.7 Methodology

1.7.1 Collecting Data

Literature study is used as a method to search matter related to distributed computing and numerical method, these literature not only limited to printed

material but also from Internet where matter still in a digital form.

1.7.2 Design

On design, observation used as a method of knowing the environment where the implementation will be possibly used. These design then used as a basis for implementation, which produce requirements for the hardware and software . This design also used as a benchmark of distributed computing using openMosix to solve complex numerical method problems.

1.7.3 Implementation

Direct experiment is used a method of implementing a design that already produced from previous phase. Here, research directly involve with required hardware and software then configure it so it will works as intended.

1.7.4 Testing

Testing methods is using time comparison between those problems solved without openMosix and compared to problems solved using openMosix. From this result, the advantages of using openMosix can be seen.

1.8 Report Outline

The report outline of this research are presented below :

1. CHAPTER I Preface

This chapter gives overview on the background of the research, problems definition, what is the objectives of the research, benefits, hypothesis and how the research will be done.

2. CHAPTER II Theory

This chapter mainly discuss the ground theory that used in this research, from the perspective of numerical method and distributed computing itself, especially on openMosix technology that will be used.

3. CHAPTER III Software Requirement

This chapter discuss the analysis method on knowing the requirement to build the distributed computing system, from the perspective of software and hardware.

4. CHAPTER IV Software Design

On this chapter, focus is upon how software and network being designed, what kind of method is used, and how is the result of the design so that it can be starting point for the implementation to work.

5. CHAPTER V Software Implementation

Here, the chapter talks about how those design being implemented to build distributed computing environment using openMosix and also discuss on how the numerical method program run.

6. CHAPTER VI Software Test and Analysis

This chapter discuss what kind of test being chosen, how the implementation being tested and compared, between those environment without distributed computing and those with distributed computing. Then, the result of the test analyzed so the detailed comparison can be seen.

7. CHAPTER VII Conclusion and Suggestion

This chapter gives conclusions on the research which already performed, then discuss some suggestion for similar research or those interested doing further research. Other things, such as limitation, problems when performing the research also presented in this chapter.

5. CHAPTER V Software Implementation

Here, the chapter talks about how those design being implemented to build distributed computing environment using openMosix and also discuss on how the numerical method program run.

6. CHAPTER VI Software Test and Analysis

This chapter discuss what kind of test being chosen, how the implementation being tested and compared, between those environment without distributed computing and those with distributed computing. Then, the result of the test analyzed so the detailed comparison can be seen.

7. CHAPTER VII Conclusion and Suggestion

This chapter gives conclusions on the research which already performed, then discuss some suggestion for similar research or those interested doing further research. Other things, such as limitation, problems when performing the research also presented in this chapter.

CHAPTER II THEORY

2.1 Numerical Method

2.1.1 General Introduction

There are problems in mathematic world which cannot be solved or needs a long and hard way to be solved using analytical methods for example :

- a) Finding the integral of $\exp(-x^2)$ or also called error function
- b) Solving linear equation system on a matrix consist of thousand rows and column.
- c) Solving a general polynomial equation of degree five or higher, the Abel-Ruffini theorem states that there is no general solution in radicals to polynomial equations of degree five or higher.

Therefore alternatives methods should be used instead. At least there two alternatives known by mathematician[WIK06a] :

1. Using asymptotic analysis, which is a method of classifying limiting behavior, by concentrating on some *trend*.
2. Using numerical solution

This number two alternatives going to be further discussed in this research, since it is already used widespread to solve mathematical problem when analytical method is not enough.

2.1.2 Definition

Numerical method is a technique to solve problems which mathematically formulated using basic arithmetic operation, such as addition, division, and others. It is a fact that cannot be neglected that the golden era of numerical method started as PC(Personal Computers) widely spread and easy to use and get. Numerous scientist then begin to harness the cheap computation power of PC by using it in various field of science.

But important to be noticed that this method already used long ago before computer was found, as we can see from few names of its method, for example Newton's method, Gaussian elimination, and Euler's method, but the presence of computer, as a powerful arithmetic tools, accelerate the use of numerical methods greatly.

2.1.3 Characteristic of Numerical Method

There are several characteristic of Numerical Method :

- **Direct and Iterative method**

Some problems can be solved directly using algorithm that automates various task that usually solved manually in a long manner. This algorithm called direct method. Some examples are Gaussian elimination for solving systems of linear equations and simplex method in linear programming.

But most problem cannot be solved using direct method, in this case we use iterative method. Iterative method starts from a guess and finds successive approximation that hopefully converge to the solution.

- **Discretization**

In numerical method continuous problems must sometimes be replaced by a discrete problem whose solution is known to approximate that of the continuous problem, this process is called *discretization*. For example the solution of differential equation is a function. This function must be represented by a finite amount of data, for instance by its value at a finite number of points at its domain, even though this domain is a continuum[WIK06a].

- **The Generation and Propagation of Errors**

Errors on numerical method holds important values due the nature of numerical methods itself, in a way it use approximation and digital computers. The first errors that likely happen is round-off errors. This is happen because it is impossible to represent all real number in digital computers. The second errors is truncation errors which committed when an iterative method is terminated and the approximate solution differs from the exact solution. The third and last error, discretization errors happen because the solution of the discrete problem does not coincide with the solution of the continuous problem.

If one of those errors above happen, it will generally propagate through the calculation. This effect also called error propagation. And it leads to the notion of numerical stability. An algorithm is numerically stable if an error, once it is generated, does not give much impact during the calculation.

2.1.4 Application of Numerical Method

Numerical method never monopolized to be only used by mathematician,

the application of this method range from various field. Some examples are in civil engineering it helps the design of structure like bridges and building. In electronics engineering it helps to calculate the amount of current on a electronic scheme. Even in economy it can help to find solution for dynamic equilibrium economies.

2.1.5 Area of Study

The field of numerical analysis is divided in different disciplines according to the problem that is to be solved

- Computing values of functions

One of the simplest problems is the evaluation of a function at a given point. But even evaluating a polynomial is not straightforward: the Horner scheme is often more efficient than the obvious method. Generally, it is important to estimate and control round-off errors arising from the use of floating point arithmetic.

- Interpolation, extrapolation and regression

Interpolation solves the following problem: given the value of some unknown function at a number of points, what value does that function have at some other point between the given points? A very simple method

is to use linear interpolation, which assumes that the unknown function is linear between every pair of successive points. This can be generalized to polynomial interpolation, which is sometimes more accurate but suffers from Runge's phenomenon. Other interpolation methods use localized functions like splines or wavelets.

Extrapolation is very similar to interpolation, except that now we want to find the value of the unknown function at a point which is outside the given points.

Regression is also similar, but it takes into account that the data is imprecise. Given some points, and a measurement of the value of some function at these points (with an error), we want to determine the unknown function. The least squares-method is one popular way to achieve this.

- Solving equations and systems of equations

Another fundamental problem is computing the solution of some given equation. Two cases are commonly distinguished, depending on whether the equation is linear or not.

Much effort has been put in the development of methods for solving systems of linear equations. Standard methods are Gauss-Jordan

elimination and LU-factorization. Iterative methods such as the conjugate gradient method are usually preferred for large systems.

Root-finding algorithms are used to solve nonlinear equations (they are so named since a root of a function is an argument for which the function yields zero). If the function is differentiable and the derivative is known, then Newton's method is a popular choice. Linearization is another technique for solving nonlinear equations.

- Optimization

Optimization problems ask for the point at which a given function is maximized (or minimized). Often, the point also has to satisfy some constraints.

The field of optimization is further split in several subfields, depending on the form of the objective function and the constraint. For instance, linear programming deals with the case that both the objective function and the constraints are linear. A famous method in linear programming is the simplex method.

The method of Lagrange multipliers can be used to reduce optimization problems with constraints to unconstrained optimization problems.

- Evaluating integrals

Numerical integration, in some instances also known as numerical quadrature, asks for the value of a definite integral. Popular methods use one of the Newton-Cotes formulas (like the midpoint rule or Simpson's rule) or Gaussian quadrature. These methods rely on a "divide and conquer" strategy, whereby an integral on a relatively large set is broken down into integrals on smaller sets. In higher dimensions, where these methods become prohibitively expensive in terms of computational effort, one may use Monte Carlo or quasi-Monte Carlo methods, or, in modestly large dimensions, the method of sparse grids.

- Differential equations

Numerical analysis is also concerned with computing (in an approximate way) the solution of differential equations, both ordinary differential equations and partial differential equations.

Partial differential equations are solved by first discretizing the equation, bringing it into a finite-dimensional subspace. This can be done by a finite element method, a finite difference method, or (particularly in engineering) a finite volume method. The theoretical justification of these

methods often involves theorems from functional analysis. This reduces the problem to the solution of an algebraic equation[WIK06a]

2.1.6 Computational Complexity Theory

In computer science, computational complexity theory is the branch of the theory of computation that studies the resources, or *cost*, of the computation required to solve a given problem. This cost is usually measured in terms of abstract parameters such as time and space. *Time* represents the number of steps it takes to solve a problem and *space* represents the quantity of information storage required or how much memory it takes. There are often trade offs between time and space that have to be considered when trying to solve a computational problem. It often turns out that an alternative algorithm will require less time but more space (or vice versa) to solve a given problem. Time requirements sometimes must be amortized to determine the time cost for a well defined average case. Space requirements can be profiled over time, too, especially in consideration of a multi-user computer system.

The time complexity of a problem is the number of steps that it takes to solve an instance of the problem as a function of the size of the input (usually measured in bits), using the most efficient algorithm. To understand this

intuitively, consider the example of an instance that is n bits long that can be solved in n^2 steps. In this example we say the problem has a time complexity of n^2 . Of course, the exact number of steps will depend on exactly what machine or language is being used. To avoid that problem, we generally use Big O notation. If a problem has time complexity $O(n^2)$ on one typical computer, then it will also have complexity $O(n^2p(n))$ on most other computers for some polynomial $p(n)$, so this notation allows us to generalize away from the details of a particular computer[WIK06d]

2.2 Distributed Computing

Distributed computing is a science which solves a large problem by giving small parts of the problem to many computers to solve and then combining the solutions for the parts into a solution for the problem[KIP06]. Distributed computing was developed with a goal of sharing computation power among computers. By this method, process tends to solve quicker compare to single process.

There are various hardware and software architectures which is used for distributed computing. At the hardware level, all of processor must be connected

on the same network, this network can be in a form of separated devices or being printed onto a circuit board. And on the software level, there will be system which manage those process running using some sort of communication system.

Below are types of distributed computing architectures[WIK06b] :

- Client-server — Smart client code contacts the server for data, then formats and displays it to the user. Input at the client is committed back to the server when it represents a permanent change.
- 3-tier architecture — Three tier systems move the client intelligence to a middle tier so that stateless clients can be used. This simplifies application deployment. Most web applications are 3-Tier.
- N-tier architecture — N-Tier refers typically to web applications which further forward their requests to other enterprise services. This type of application is the one most responsible for the success of application servers.
- Tightly coupled (clustered) — refers typically to a set of highly integrated machines that run the same process in parallel, subdividing the task in parts that are made individually by each one, and then put back together to make the final result.

- Peer-to-peer — an architecture where there is no special machine or machines that provide a service or manage the network resources. Instead all responsibilities are uniformly divided among all machines, known as peers.
- Service oriented — Where system is organized as a set of highly reusable services that could be offered through a standardized interfaces.
- Mobile code — Based on the architecture principle of moving processing closest to source of data
- Replicated repository — Where repository is replicated among distributed system to support online / offline processing provided this lag in data update is acceptable.

There are broad example where this system implemented and used nowadays, from medics, security to not so important ones, here are some currently active projects :

- Fight AIDS@Home (<http://fightaidsathome.scripps.edu/>)

The objective of this project is to discover new drugs for AIDS by trying to find candidate drugs that have the right shape and chemical characteristic to block HIV protease. This project completed Phase I on

May 21st, 2003 where almost 60.000 computers completed 1.400 years of computing to process over 9 million tasks.

- Einstein@Home (<http://einsteinathome.org/>)

In 1916, Albert Einstein in his General Theory of Relativity predict the existence of gravitational waves, but this wave cannot be detected at that time due to engineering limitation. Based on that, this project searches for spinning neutron stars(also called pulsars), which are likely to emit gravitational waves, using data from LIGO (Laser Interferometer Gravitational Wave Observatory, located in USA) and GEO gravitational wave detector. Currently, there are more than 160.000 host joined this project.

- Project RC5 (<http://www1.distributed.net/rc5/>)

This interesting project is coordinating and maintaining the RC5 servers that are needed to distribute key blocks to all of the participating client programs, this key block is trying to be solved through the use of brute force attack. On 19 October 1997, this project found the correct solution for the RSA Labs 56-bit secret-key challenge, and it took 250 day to found it. Then, on 14 July 2002, this project found the winning key for the RSA Labs 64-bit secret-key challenge, where it took 1.757 days and more than

300.000 individual to locate. As of 3 December 2002, the project working on 72-bit RSA Labs secret-key challenge.

2.3 Computer Cluster

A computer cluster is a group of loosely coupled computers that work together closely so that in many respects it can be viewed as though it were a single computer[WIK06c]. It is known that there is a lot of idle processor load on computers nowadays, on the other side sometimes processor load can be so high, for example when compiling hundreds line of C++ code or encoding movies from a handy cam to MPEG-4 format. This is where the idea of cluster is, which to spread loads among all available computers, using the resources that are free on the other machines.

2.3.1 Types of Computer Cluster

There are three types of cluster commonly know today, which is :

- Fail-over cluster

Also known as high-availability cluster, this cluster consist two or more network connected computers which are backup each other, so that if one of the computers is fail or down, the other computers will replace that

computers. This redundant action is done to improve availability and eliminate a single point of failure, especially on a condition where 99,99% uptime is crucial. On Linux OS, Linux-HA(<http://linux-ha.org>) is a software project which provide such availability.

- **Load balancing cluster**

As for the name, this cluster manages workloads, distribute it into nodes which is least busy compare to others. This cluster main goal is to improve performance, although sometimes it is combine with fail-over cluster, so that two benefits, improved availability and improved performance can be gain. There are some implementation of load balancing cluster, commercial one such as Moab cluster suite or Maui Cluster Scheduler and free software such as openMosix or Linux Virtual Server.

- **High Performance Computing Cluster**

This type of cluster splitting a computational task across many different nodes. Machines which join in this kind of cluster are configured specially to provide extreme performance by running custom program to maximize parallelism in this cluster. Usually those custom program use library for example PVM(Parallel Virtual Machine), MPI(Message Passing Interface), or AFAPI(Aggregate Function API).

2.3.2 Cluster and Supercomputer

Supercomputers are usually built by a selected number of vendors, such as Cray, IBM and Fujitsu, and only certain company and organization afford to buy such machine due to the price which are very expensive. On the other hand, lots of universities, organization and institution also need high performance machine. One of the best alternatives is to build cluster of computers, where its hardware are available everywhere. Nowadays, even some of these cluster able to compete with supercomputer on TOP500, a project to list 500 most powerful computer system in the world.

2.4 openMosix

2.4.1 Overview

openMosix is a software package that turns networked computers running GNU/Linux into a cluster. It will automatically balance the load between different nodes of cluster, and nodes can join or leave the running cluster without disruption of service. The load is spread out among nodes according to their connection and CPU speeds.

openMosix come in a form of Linux-kernel patch, so that it can easily integrated into system who use the Linux-kernel, especially 2.4 version, while 2.6 version is actively developed. As it is part of kernel a user's program, files, setting and other resources will work without any further changes.

2.4.2 Component of openMosix

There are three main component that formed openMosix :

1. Process Migration

Using openMosix, process which migrate can be seen, because each process has its own Unique Home Node(UHN) where it gets created. Migration here means that a process is splitted in two parts, user part and a system part. The user part will be moved to a remote node while the system part will stay on the UHN. This system-part is sometimes called the deputy process: this process takes care of resolving most of the system calls.

2. The openMosix File System(oMFS)

oMFS is a feature of openMosix which allows administrator to access remote filesystems in a cluster as if they were locally mounted. The filesystems in the other nodes can be mounted on /mfs.

3. Direct File System Access(DFSA)

Both Mosix and openMosix provide a cluster-wide file-system (MFS) with the DFSA-option (Direct File-System Access). It provides access to all local and remote file-systems of the nodes in a Mosix or openMosix cluster.

2.4.3 openMosix Internal

openMosix consists of a mechanism called Preemptive Process Migration (PPM) which is a set of algorithms for adaptive resource sharing. openMosix works in a decentralized way, where there is no central control or master/slave relationship between the nodes. Scalability is achieved by randomness in the system control algorithm, where each node does not attempt to determine the overall state of the cluster or any particular node. Each node sends information about its available resources to a randomly chosen subset of other nodes. So one node know the status of some neighbors not all of them.

But the interesting point is the mathematical model for scheduling algorithm comes from the fields of economics research. As it is known that every nodes usually heterogeneous one, where there are differences in use of memory, CPU, network latency, and others, then openMosix try to see those differences

into single homogeneous “cost”. From here jobs are then assigned to the machine where they have the lowest cost, just like a market-oriented economy. [KRB05]

2.4.4 Pros and Cons using openMosix

Despite of many benefits when we use openMosix, there are also some weaknesses on using openMosix, below are comparison pros and cons using openMosix.

Pros of openMosix :

- No extra packages are required.
- No code changes to your application are required.
- Simple to install/configure.
- On a Red-Hat based system/distro, installing openMosix is as simple as typing: `# rpm -Uvh openMosix*.rpm`
- Well integrated with openAFS.
- Port to IA-64
- oMFS has been improved much since plain MFS.
- It is a clustering platform with more than 10 products based on it: openMosixView, openMosixWebView, openMosixApplet, RxLinux, PlumpOS, K12LTSP, LTSP and many others.

- openMosix is a product developed by the users themselves so it's more close to the user by definition.
- Node autodiscovery/fail-over daemon already implemented in the user land tools via multicast messaging.
- Aliases for hosts with multiple interfaces.
- Basic routing available (in the rare case where true multicast routing is undesirable).
- Cluster Mask allows to specify to which nodes a given process can migrate.

Cons of openMosix :

- Kernel dependent.
- Shared memory issues.
- Security issues.
- There are issues with Multiple Threads not gaining performance.
- Performance would not be gained when running one single process such as web browser on an openMosix Cluster: the process won't spread itself over the cluster. Except of course the process will migrate to a better performance machine.

2.4.5 Linux Distribution with openMosix

There are several Linux distribution that already includes openMosix in their kernel. This Linux distribution is designed so that we can harness the ability of openMosix right after boot-up in a form of Live CD or installable distribution.

Here is the list :

- ClusterKnoppix (<http://bofh.be/clusterknoppix/>)

As known from its name, this distribution is developed based on famous Knoppix project. It has several features such as terminal server, so it can use PXE, DHCP and tftp to boot linux clients via network, includes Cluster Management tools, and new nodes can automatically join the cluster. Unfortunately, this project which started by Wim Vandersmissen, stopped its development on 2004, since its latest release was version 3.6 on 31st August, 2004.

- Quantian (<http://dirk.eddelbuettel.com/quantian.html>)

Quantian is an extension of Knoppix and ClusterKnoppix, where its provide some features from both distribution. What make this distribution different from both distribution above is there a large number of programs of interest to applied or theoretical workers in quantitative or data-driven

fields. Some example are R, Octave, GSL(GNU Scientific Library), Quantlib, Scientific and Numeric Python and Maxima. The most recent version is 0.7.9.1, which released on December 11, 2005 in a form of DVD.

- CHAOS (<http://midnightcode.org/projects/chaos/>)

The objective of this project is to create platform capable of large-scale ad-hoc cluster deployment and to be the fastest, most compact, secure and straight-forward openMosix cluster. There are some security improvement that differs with another distribution by included IPSEC tunnels for all cluster communications, state aware packet filtering for each node, and others.

- Gary's openMosix Floppy (GoMF) (<http://gomf.sourceforge.net>)

This distribution is a single floppy openMosix Linux mini-distro designed to quickly add CPU/Memory resources to an openMosix cluster. It includes the auto discovery daemon and some user tools like mosctl, mosrun, mosmon, mps, setpe.

CHAPTER III

SOFTWARE REQUIREMENT

3.1 Analysis Method

The method used to analyze the software requirement is mainly based on literature study ranging from the software how-to written by the author of the software, various article in internet which explain how to implement the software until some tutorial written based on other people experience. The reason why the compiler choose this method because this is commodity software that built by other people or group and not something that built from scratch to fulfill certain needs.

The compiler also used simple observation to some network in offices and computer laboratory in environment common in Indonesia, where this kind of environment is the target of where the research will be implemented.

3.2 Analysis Result

3.2.1 Data Analysis

As already become common fact that numerical methods are used mostly in university, especially in those university that taught engineering study, and

some research institution nowadays. Other fact from simple observation is almost every institution stated above have networked computers in a form of Local Area Network(LAN) in a form of office computers or computer laboratory for teaching purposes.

Based on those two fact, openMosix is design to be implemented on a LAN. So the requirement is built from general knowledge of where people will use numerical methods and common topology exist in that area.

3.2.2 Software Requirement

The minimum requirement so that the research can be fully running as expected are :

- Linux kernel 2.4.26 with openMosix patch, all node must run openMosix with the exact version.
- KDE 3.2, as a user-friendly environment desktop.
- openmosixview
- GNU Octave 2.1

Due to the matter of practicality, all of software required above are already available on Quantian Linux Distribution version 0.5.9.2, it is the latest CD version released on July 2004, since start from that version all Quantian

distribution are released in DVD format. While nowadays, the use of DVD is still not as much as CD.

3.2.3 Hardware Requirement

There are minimum requirements that must be fulfilled so that openMosix can be implemented which are :

- A PC with Pentium III or Celeron Processor 700 MHz, 128 MB RAM, 8 MB VGA, Bootable CDROM drive and a Monitor.
- Network Interface Card 10/100 Mbps
- 5 or 8 port 10/100 switch

Those hardware mentioned above are minimum standard that owned by most of universities and institution in Indonesia.

CHAPTER IV SOFTWARE DESIGN

4.1 Design Method

On this chapter, there are two things that will be discussed, on how the numerical method software being developed and how are the network diagram for the system itself being designed. On the point of numerical method software, the process of design will focus more on giving analysis of the software that will be used in testing the openMosix system, since the focus of this research is on implementing openMosix system.

4.2 Design Result

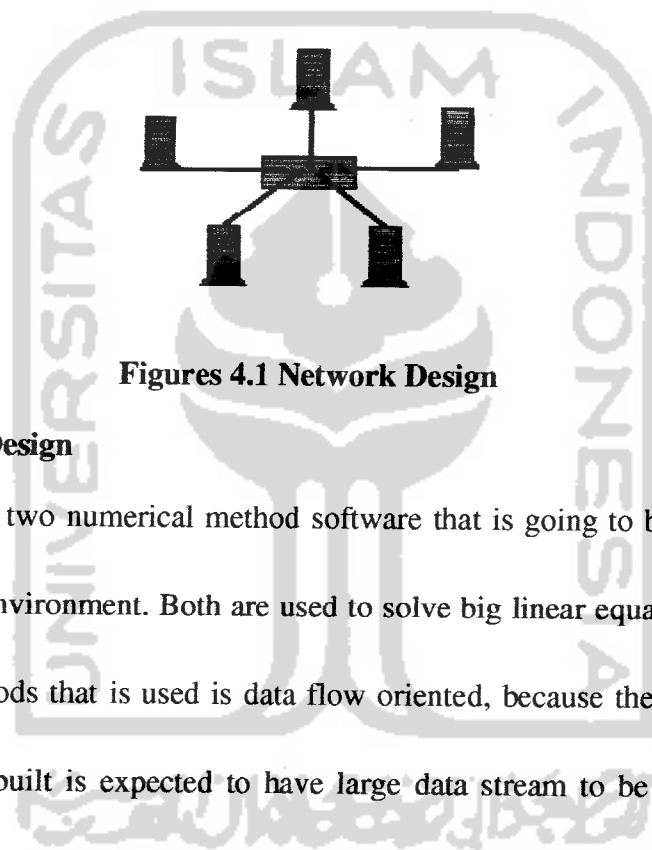
4.2.1 Network Design

On this sub chapter, focus will be on topology used in implementation of openMosix. As it is known that openMosix does not have dependency to a certain topology, so any topology as far as it can connect few nodes then it is alright for openMosix to use it.

Most of Local Area Network(LAN) in Indonesia are using star topology nowadays, the use of bus topology is deprecated, as on star topology it is more

easier to build. Based on this openMosix will be using this topology with the assumption of five nodes on each LAN, because this is the minimum amount of port that is common in a network switch.

And so the diagram of network will be like this :



Figures 4.1 Network Design

4.2.2 Software Design

There are two numerical method software that is going to be used to test the openMosix environment. Both are used to solve big linear equation systems.

The methods that is used is data flow oriented, because the software that are going to be built is expected to have large data stream to be processed by openMosix system.

4.2.2.1 Big Linear Equation System using Matrix Inverse

For $n \times n$ matrix, the inverse matrices can be found using Gauss-Jordan elimination methods, where :

$$[A | I] \text{ -- Gauss-Jordan elimination --> } [I | A^{-1}]$$

For a clearer explanation, see example on how to get inverse matrices from the matrix below :

$$A = \begin{bmatrix} 1 & 1 & 2 \\ 0 & 1 & 2 \\ 2 & 1 & 1 \end{bmatrix}$$

Solution :

$$\left[\begin{array}{ccc|ccc} 1 & 1 & 2 & 1 & 0 & 0 \\ 0 & 1 & 2 & 0 & 1 & 0 \\ 2 & 1 & 1 & 0 & 0 & 1 \end{array} \right] \begin{array}{l} \\ R_3 - 2R_1 \\ \end{array} \quad \left[\begin{array}{ccc|ccc} 1 & 1 & 2 & 1 & 0 & 0 \\ 0 & 1 & 2 & 0 & 1 & 0 \\ 0 & -1 & -3 & -1 & 0 & 1 \end{array} \right] \begin{array}{l} R_1 - R_2 \\ \\ R_3 + R_2 \end{array}$$

$$\left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & -1 & 0 \\ 0 & 1 & 2 & 0 & 1 & 0 \\ 0 & 0 & -1 & -2 & 1 & 1 \end{array} \right] \begin{array}{l} \\ \\ -R_3 \end{array} \quad \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & -1 & 0 \\ 0 & 1 & 0 & -4 & 3 & 2 \\ 0 & 0 & 1 & 2 & -1 & -1 \end{array} \right] \begin{array}{l} R_2 - 2R_3 \\ \\ \end{array}$$

By using the inverse matrices, linear equation systems can be solved through characteristics of matrix. One of matrix characteristic is that A times A^{-1} will result identity matrix.

$$AA^{-1} = A^{-1}A = I$$

Other characteristic of matrix is if a matrix A times I will result matrix A itself, as shown below

$$AI = IA = A$$

Based on those two characteristic, linear equation system $Ax=B$, can be solved using the following way :

$$\begin{aligned} Ax &= b \\ A^{-1}Ax &= A^{-1}b \\ Ix &= A^{-1}b \\ x &= A^{-1}b \end{aligned}$$

4.2.2.2 Big Linear Equation System using Backslash Division

Instead of using inverse matrices, matlab/octave also provide faster and more accurate way, which is using backslash division or matrix left division. According to Matlab manual, it is stated that backslash division is computed using Gaussian elimination. The reason why this method provide faster way because it is reduce the process to produce the inverse matrices first, and directly process the matrix.

Gaussian elimination is implemented to solve linear equation systems by forming upper triangle, where all of coefficients below main diagonal are zero, and then solve it using backward substitution. To form those triangle, Gaussian elimination methods use some elementary row operation, which are :

1. Switching position between row of equation
2. Change an equation by multiplying a row of equation

3. Multiply an equation with non zero constant

Let see an example below, on how it is implemented. Suppose there are three equation :

$$3x_1 + 4x_2 + 3x_3 = 20 \quad (\text{A})$$

$$x_1 + 5x_2 - x_3 = 8 \quad (\text{B})$$

$$6x_1 + 3x_2 + 7x_3 = 33 \quad (\text{C})$$

1. Eliminate x_1 from equation (B) and equation (C) :

$$3x_1 + 4x_2 + 3x_3 = 20 \quad (\text{A})$$

$$(\text{B}) - \frac{1}{3} (\text{A}) \Rightarrow \frac{11}{3} x_2 - 2x_3 = \frac{4}{3} \quad (\text{D})$$

$$(\text{C}) - 2 (\text{A}) \Rightarrow -5x_2 + x_3 = -7 \quad (\text{E})$$

1. Eliminate x_2 from (E) equation :

$$3x_1 + 4x_2 + 3x_3 = 20 \quad (\text{A})$$

$$\frac{11}{3} x_2 - 2x_3 = \frac{4}{3} \quad (\text{D})$$

$$(\text{E}) + \frac{15}{11} (\text{D}) - \frac{19}{11} x_3 = -\frac{57}{11} \quad (\text{F})$$

2. Calculate x_3 , x_2 , and x_1 using backward substitution :

From (F) we get $x_3 = 3$. Insert x_3 value into (D) and we get $11/3x_2 = 4/3+6$
 $= 22/3$. So we get $x_2 = 2$. Insert x_3 and x_2 value into (A) to get x_1 value: $3x_1$
 $= 20 - 8 - 9 = 3$, $x_1=1$. So, the the answer for the linear equation system
 above is (1,2,3).

CHAPTER V

SOFTWARE IMPLEMENTATION

5.1 Scope

On this chapter, all of previous design will be implemented and then compared based on the time consumed and computer configuration used. But there are few scope of the implementation :

1. Test will be conducted and monitored based on time consumed to solve complex numerical methods problems
2. All operating systems parameters, openMosix configuration, network configuration will be kept default. No interference in term of load balancing process. Few changes, if there any, will be number of computers join the computation node.

5.2 Implementation

Here, the previous design implemented in a form of environment suitable to test various hardware configuration and different complexity of software.

5.2.1 Hardware Implementation

There are few configuration that implemented to form different scenario for testing, which are :

1. Base System Configuration

This configuration is implemented as a benchmark for the other test. As a tool of comparison, this configuration implemented on the minimum requirement available at the test environment. And this configuration only used on single stand alone computer. Here is the specification for the configuration :

- Processor : Intel Celeron 800 MHz, 128 KB L2 Cache
- Memory : 128 MB SDRAM
- Graphic Card : Trio 3D/2X 8 MB
- Monitor : GTC 15"
- CDROM : 52X CDROM Drive

Hard disk is not necessarily installed because all testing configuration will be using linux live CD, Quantian.

2. Homogeneous Cluster Configuration

On this configuration, there are five computers connected through a switch. Each computers have almost exact same configuration. The specification for each computers is

- Processor : Intel Celeron 800 MHz, 128 KB L2 Cache
- Memory : 128 MB SDRAM

- Graphic Card : Trio 3D/2X 8 MB
- Monitor : GTC 15"
- CDROM : 52X CDROM Drive
- NIC : 10/100 MBps

Additional peripheral :

- 16 port switch

This configuration also does not need hard disk.

3. Heterogeneous Cluster Configuration

While on this configuration there are few hardware specification differences, since it try to simulate an environment with various hardware configuration. There are five computers, where two computers are having different specification. Three computers using same specification which is

- Processor : Intel Celeron 800 MHz, 128 KB L2 Cache
- Memory : 128 MB SDRAM
- Graphic Card : Trio 3D/2X 8 MB
- Monitor : GTC 15"
- CDROM : 52X CDROM Drive
- NIC : 10/100 MBps

The fourth computer using specification below

- Processor : Intel Pentium 4 2.26 GHz

- Memory : 128 MB DDR RAM (96 MB available)
- Graphic Card : Shared 32 MB
- Monitor : GTC 15"
- CDROM : 52X CDROM Drive
- NIC : 10/100 MBps

And finally the fifth computer having the best specification as mentioned below

- Processor : Intel Pentium 3 866 MHz
- Memory : 128 MB SDRAM
- Graphic Card : Trio 3D/2X 8 MB
- Monitor : GTC 15"
- CDROM : 52X CDROM Drive
- NIC : 10/100 MBps

Additional peripheral :

- 16 port switch

5.2.2 Software Implementation

5.2.2.1 openMosix Implementation

These are steps needed to implement all the software to form the environment for the testing :

1. Boot the computer using Quantian Linux Live CD

2. When boot already done, there will be a boot prompt. On this boot prompt type the following command

```
boot : knoppix vsync=60
```

The above command will set vertical refresh rate manually into 60 Hz, and allowed unrecognized monitor, which is used in this testing to maximize its screen resolution until 1024 x 768, otherwise, it will only give 800 x 600 screen resolution.

3. The next step is open a terminal console and become super user by typing

```
$ su
```

4. Then set the network address using following command

```
# ifconfig eth0 192.168.0.1 netmask 255.255.255.0
```

On the other node the range of IP will start from 192.168.0.2 until 192.168.0.5 with the same netmask.

5. And then make the openMosix auto discovery daemon listen on the network interface, so that one node will automatically found other node, by typing

```
# omdisc -i eth0
```

6. Run Octave by typing octave on the terminal console

```
# octave
```

7. Start the openmosixview by click openmosixview icon on the left side of icon bar.
8. On the openmosixview window, click the openmosixmigrmon icon, so that the migration process able to be seen
9. Also start KSysGuard applet to monitor cpu load and memory usage.
10. Last but not least also start KSnapshot to take entire screen snapshot.

If all of those step successfully entered, it should give appearance below :



Figures 5.1 openMosix on Quantian Linux

The upper right windows show openmosixview, a tools to give monitor on load of every node connected on a openMosix systems, it also show openMosix-speed of each node, number of memory provide, and number of CPU on each node.

While the big window on the left is openmosixmigmon, used as sign if there is any migration process that happen, between home node into another node. Home node shown by the picture of penguin in the middle and the other node shown by other penguins around it. The black dot around the home node are the representation of each process in home node, that might will migrate if the load in home node are too big.

5.2.2.1 Numerical Method Software Implementation

The source code to solve big linear equation systems using matrix inverse in .m programming language is presented below :

```
t1=cputime();           % Start the timer based on cputime
n=2000;                % Matrix size

A=floor(rand(n)*10);   % Randomly create matrix a
b=floor(rand(n,1)*10); % Randomly create matrix result b
x=inv(A)*b;           % Solve it using inverse division
t2=cputime();         % Record cputime when it already computed
time=t2-t1            % Print how much time needed to solve
```

While the source code to solve big linear equation systems using backslash division is presented below :

```

t1=cputime();           % Start the timer based on cputime
n=2000;                 % Matrix size

A=floor(rand(n)*10);    % Randomly create matrix a
b=floor(rand(n,1)*10); % Randomly create matrix result b
x=A\b;                 % Solve it using backslash division
t2=cputime();          % Record cputime when it already computed
time=t2-t1             % Print how much time needed to solve

```

Here is the analysis for the source code above. First we start the counter by putting the start time, which is the `cputime` into variable `t1`. And then we set the matrix size on a variable `n`. The function `rand()` is used to generate matrix `A` which is a $n \times n$ matrix, but because the generated value is between 0 and 1, so it is multiplied by 10 and rounded using `floor()` function, in order to get random integer between 0 and 9. Similar step happens on matrix `b`, only in this matrix the size is $n \times 1$. Next, matrix `x` is calculated using backslash division.

CHAPTER VI

SOFTWARE TEST & ANALYSIS

6.1 Performance Test

After all implementation is set, then the testing procedure will be started. This will become black-box testing, because it will be more focus upon functionality of software, whether it will run as expected or not. This testing procedure is using five size of elements based on the matrix size, which are 100, 500, 1000, 2000 and 3000. The code that will be used is as what already mentioned on previous chapter. And three different configuration also will be tested, to gain better understanding on openMosix implementation, and its impact toward complex numerical method problems.

The only measurement in this test is how long does a test can be solved, using configuration provided. To increase the accuracy of the result each test are repeated five times, then average time needed to solve a process also presented.

6.2 Result and Analysis

Below are the result of entire test and also the analysis of the test

6.2.1 Base System Test

This test only executed on a single computer, with the specification already mentioned on chapter V page 38, and used as benchmark or comparison purposes. The result is shown in table 6.1 below.

Table 6.1 Inverse matrices on base system

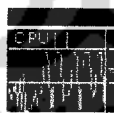
MSize	ESize	1 st Test	2 nd Test	3 rd Test	4 th Test	5 th Test	6 th Test
100	10.000	0.050	0.010	0.010	0.020	0.010	0.02
500	250.000	1.130	0.930	0.920	0.900	0.910	0.96
1000	1.000.000	6.500	6.460	6.150	5.650	6.270	6.21
2000	4.000.000	42.480	40.710	42.160	42.190	42.360	41.98
3000	9.000.000	134.870	135.040	134.540	132.990	133.740	134.24

At matrix with the size of 100 and 500 the process goes very quick, on the 1000, where computer must process 1.000.000 element and do a lot more repetition, also shows no significant difficulties. But when the size of matrix increase two times to 2000 the time needed to solve those linear equation system increase more than seven times. On the matrix size of 3000, the times needed to solve increase more than three times compare to previous size.

Table 6.2 Backslash division on base system

MSize	ESize	1 st Test	2 nd Test	3 rd Test	4 th Test	5 th Test	Average
100	10.000	0.110	0.020	0.020	0.020	0.020	0.04
500	250.000	0.690	0.450	0.450	0.430	0.430	0.49
1000	1.000.000	2.410	2.390	2.400	2.420	2.400	2.4
2000	4.000.000	15.400	14.970	15.070	15.290	15.380	15.22
3000	9.000.000	46.190	46.940	47.230	47.640	45.870	46.77

Same thing also happen on backslash division method. Other fact that can be shown in those table is as the size of matrix grown in a step of thousands, the times needed to compute all the element are grown exponentially. The table also shown that backslash division really gives more efficient solution compare to inverse matrices methods. Picture below shown what happened on the test.

**Figure 6.1 KSysGuard applet**

The graphic above, a screen shot of KSysGuard, is captured after backslash division process finished its five times testing on matrix size of 2000. The small peak on the left shown CPU load for matrix size of 1000, the average load needed are slight above 50%, and five higher peak on the right shows that for a moment for each test the CPU load have reach 100%.

6.2.2 Homogeneous System Test

The result is shown in table 6.3 below.

Table 6.3 Inverse matrices on homogeneous system

MSize	ESize	1 st Test	2 nd Test	3 rd Test	4 th Test	5 th Test	Average
100	10.000	0.150	0.020	0.020	0.020	0.020	0.05
500	250.000	1.010	0.930	0.920	0.900	0.900	0.93
1000	1.000.000	7.530	5.840	5.970	5.970	5.980	6.26
2000	4.000.000	48.160	43.630	42.640	47.130	43.160	44.94
3000	9.000.000	136.100	135.940	135.940	135.150	131.840	134.99

In a contrast on what first expectation on how cluster system would make process would go faster, this test shows how the migration also needs time to process the data and on communicating to the other nodes.

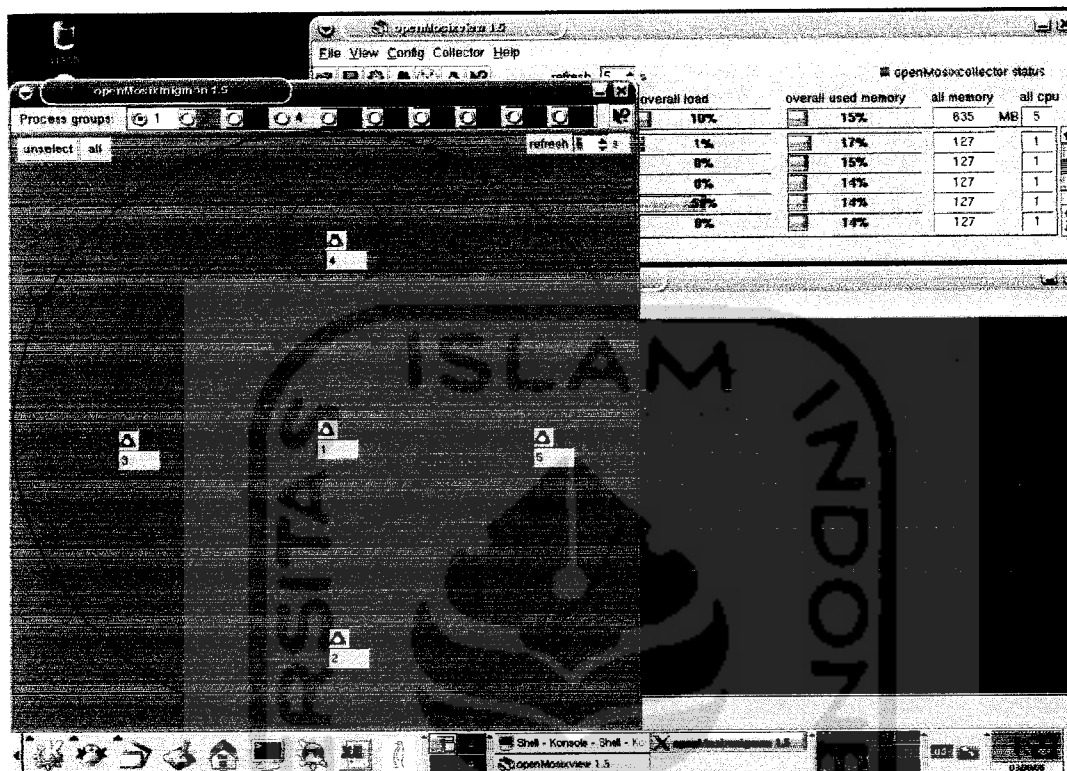


Figure 6.2 Process Migration on Homogeneous System

As we can see from the picture above, the black dot around the penguin is PID(Process Identification) of process that exist in home node, and four other penguin that surround it is another openMosix node detected. It is also shown a process migration, where we can see one black dot moving into node number four or top penguin. It means there is one process from home node which migrate into node number four. At the openmosixview the load of computer number four also increased.

Table 6.4 Backslash division on homogeneous system

MSize	ESize	1 st Test	2 nd Test	3 rd Test	4 th Test	5 th Test	6 th Test
100	10.000	0.150	0.010	0.020	0.010	0.020	0.04
500	250.000	0.730	0.560	0.480	0.450	0.440	0.53
1000	1.000.000	2.830	2.470	2.400	2.440	2.410	2.51
2000	4.000.000	19.890	15.450	15.600	14.850	15.200	16.2
3000	9.000.000	50.060	48.070	46.100	48.050	48.110	48.08

The same thing also happen on backslash division, there is slight increase on the time needed to solve the equation. Although the increase is quite small from few millisecond until few seconds but it prove longer time are needed for openMosix on a homogeneous system in an exchange of lesser load on home node.

6.2.3 Heterogeneous System Test

The results is shown in table 6.5.

Table 6.5 Inverse matrices on heterogeneous system

MSize	ESize	1 st Test	2 nd Test	3 rd Test	4 th Test	5 th Test	6 th Test
100	10.000	0.220	0.020	0.030	0.020	0.020	0.06
500	250.000	1.120	0.950	0.950	1.040	0.280	0.87
1000	1.000.000	7.080	5.850	1.970	1.820	1.870	3.72
2000	4.000.000	14.760	45.140	14.340	36.940	14.480	25.13
3000	9.000.000	147.66	138.04	146.14	135.17	135.55	140.51

This test is the most interesting of all test, since it really shows the advantages of using distributed computing, in this case openMosix. From the matrix size of 1000, there are quite significant improvement especially on the third test where there is dramatic decrease from 5 seconds to almost 2 seconds. It shows that process goes 50 % faster. Compare to base system, the average times needed decreased from 6.26 seconds to 3.72 seconds.

On the matrix size of 2000, the results sometimes fluctuate from the lowest 14.340 seconds until the highest 45.140 seconds. But on average there are significant decrease from 41.98 seconds on base system to 25.13 seconds on heterogeneous system.

The only lower results happen on the matrix size of 3000, not only the average time needed slightly increase from 134.24 seconds in base system to 140.51 in heterogeneous system. Not only that, during test on the matrix size of 3000, the home system experience several times crash. It appears because the unstable condition that result from several factor start from minimum memory condition, because since the system is Live CD, then quite amount of memory needed to run the process and as temporary storage. Other factor because there are few additional application executed and adding the load on home node, such as

openmosixview, openmosixmigmon and KSnapshot.

Table 6.6 Backslash Division on Heterogeneous System

MSize	ESize	1 st Test	2 nd Test	3 rd Test	4 th Test	5 th Test	Average
100	10.000	0.130	0.010	0.020	0.020	0.020	0.04
500	250.000	0.480	0.480	0.480	0.450	0.450	0.47
1000	1.000.000	0.790	0.800	0.770	0.810	0.810	0.8
2000	4.000.000	17.750	18.630	5.260	5.200	12.860	11.94
3000	9.000.000	49.510	47.970	47.860	51.960	50.650	49.59

The significant improvement happened on the matrix size of 1000 using backslash division, where dramatic decrease from 2.4 seconds into 0.8 seconds, in another word it cut times to only 30% compares to the base system.

On the matrix size of 2000, there is quite decrease on the third and fourth test. This fluctuate result range from the lowest point 5.200 seconds into the highest 18.630 seconds. While on base system, the lowest results are 14.970 seconds and the highest 15.400 seconds. On the average it gives slight decrease from 15.22 seconds to 11.94 seconds.

Same behavior as inverse matrices on the matrix size of 3000 also happen on backslash division. It gives slight increase of times. The average time of base system is 46.77 seconds, while on heterogeneous system 49.59 seconds. It appears although migration did happen but not the octave process, instead on other small

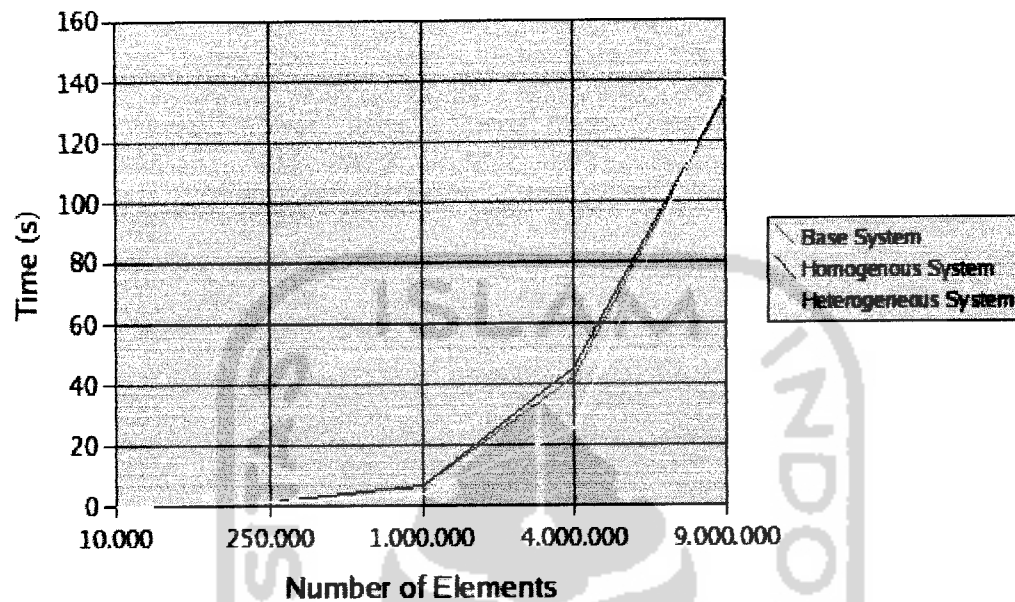
process. There is alternatives on both cases(inverse matrices and backslash division), which is by using manual migration on process with high load, but that is not the purpose of this research, since this research will only used default configuration provided.



Figure 6.3 Process Migration on Heterogeneous System

Another interesting fact we can see from the snapshot above, which taken on the implementation, there are four process that migrate into the fourth node, the best node exist on the network. It also makes the load on the fourth node increase to 66%, while other node give average load of 20%.

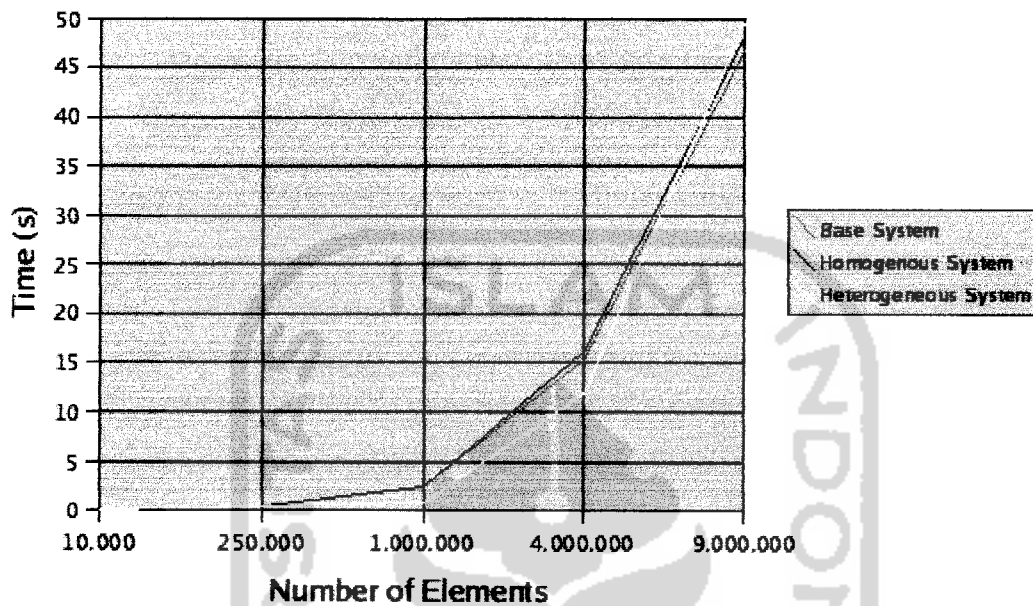
Inverse Matrices Comparison



Figures 6.1 Overall comparison on inverse matrices

As comparison, the results of all of the research is presented on the chart, where it shown that heterogeneous system give most advantages compares to other methods. The best result for heterogeneous system happened on matrix size of 2000 or where number of elements is 4.000.000. While the worst result for heterogeneous system happened on matrix size of 3000, where the process did not migrate to the best node.

Backslash Division Comparison



Figures 6.2 Overall comparison on backslash division

Similar result also happen on backslash division method, as we can see from the charts, where heterogeneous system proved to best method. Although it gives bad result on the matrix size of 3000, where the migration not happened. The chart also proved that backslash division is faster compare to inverse matrices method, because time needed to solve the big linear equation is much more efficient using backslash division.

CHAPTER VII

CONCLUDING REMARK

7.1 Conclusions

From all of research that already implemented there are few point of conclusion, which are :

1. Load balancing have obvious advantages on network where exist better specification computers compare to home node, so that the process which need a lot of processing power can easily migrate to that better computers. By this scenario also, the computation power of high specification in a node can really be optimized and shared using heterogeneous system cluster.
2. If there are two condition between single stand alone system and homogeneous system with the same specification, the preferred choice is single stand alone, since it would not result much advantages as it seen from comparison from base system and homogeneous system on previous chapter. Unless there are few process on home node that each process need lot of computation power, then homogeneous system are more preferred.
3. Although automatic migration sometimes enough to accommodate the

computing need of certain node, manual migration will give better results.

Since the user know what process that really need computing power of other node, and what process that enough to be computed on home node.

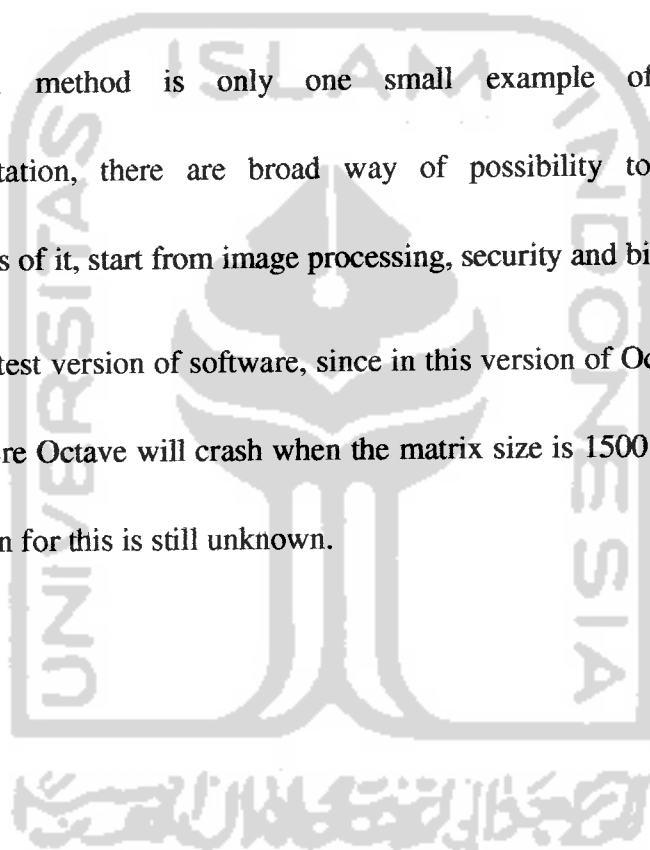
This is why on some test on heterogeneous system on previous chapter some process would migrate, some other would not.

7.2 Suggestions

There are also some suggestion regarding further research on this subject, which are :

1. The use of live CD should only be considered if the installation of real linux system with openMosix kernel patch not authorized or not available. Since live CD often caused unstability that can impact on system crash due to the high usage of system memory. Other alternatives by adding more system memory.
2. Other alternatives than manually migrate the process is by lower the home node openMosix speed to the minimum. So that process tend to migrate to other node, this will give home node more chance to process other applications.

3. There also a lot of way to tune openMosix to deliver it maximum performance, such as changing the network parameters, the use of better form of networking hardware, and many more. This is not yet to be explored in this research.
4. Numerical method is only one small example of openMosix implementation, there are broad way of possibility to harness the advantages of it, start from image processing, security and bioinformatics.
5. Try the latest version of software, since in this version of Octave there are bugs, where Octave will crash when the matrix size is 1500 and 2500, the true reason for this is still unknown.



REFERENCES

- [BUY04] Buytaert, K. The openMosix HOWTO.
<http://howto.x-tend.be/openMosix-HOWTO/>, accessed on December 29, 2005.
- [MUN03] Munir, R. *Metode Numerik*. Bandung: Informatika, 2003.
- [PEA06] Pearson, K. *Distributed Computing*,
<http://www.distributedcomputing.info/index.html>, accessed on January 6, 2006.
- [PRE02] Pressman, Roger S., *Rekayasa Perangkat Lunak: Pendekatan Praktisi (Buku I)*. Yogyakarta: Andi Offset, 2002.
- [SAH05] Sahid. *Pengantar Komputasi Numerik dengan MATLAB*. Yogyakarta: Andi Offset, 2005.
- [TRI98] Triatmodjo, B. *Metode Numerik*. Yogyakarta: Beta Offset, 1998.
- [WIK06a] Wikipedia contributors, *Numerical analysis*,
http://en.wikipedia.org/w/index.php?title=Numerical_analysis&oldid=32955403, accessed January 6, 2006.
- [WIK06b] Wikipedia contributors, *Distributed Computing*,

http://en.wikipedia.org/w/index.php?title=Distributed_computing&oldid=33523528, accessed January 9, 2006.

[WIK06c] Wikipedia contributors, *Computer Cluster*,

http://en.wikipedia.org/w/index.php?title=Computer_cluster&oldid=3821646, accessed January 8, 2006.

[WIK06d] Wikipedia contributors, *Computational complexity theory*,

[http://en.wikipedia.org/w/index.php?title=Computational complexity theory](http://en.wikipedia.org/w/index.php?title=Computational_complexity_theory), accessed February 15, 2006.

