

DOKUZ EYLÜL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCE

**PLATFORM DEVELOPMENT FOR PARALLEL
OPERATION OF SINGLE BOARD COMPUTERS**

by
Kübra KARADAĞ

September, 2017

İZMİR

PLATFORM DEVELOPMENT FOR PARALLEL OPERATION OF SINGLE BOARD COMPUTERS

**Thesis Submitted to The Graduate School of Natural and Applied Sciences of
Dokuz Eylül University Mechatronics Engineering for The Degree of Master of
Science in Mechatronic Engineering, Mechatronics Engineering Program**

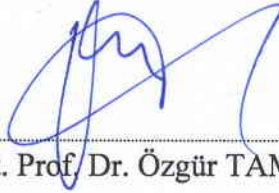
**by
Kübra KARADAĞ**

September, 2017

İZMİR

M.Sc THESIS EXAMINATION RESULT FORM

We have read the thesis entitled “**PLATFORM DEVELOPMENT FOR PARALLEL OPERATION OF SINGLE BOARD COMPUTERS**” completed by **KÜBRA KARADAĞ** under supervision of **ASIST. PROF. DR. ÖZGÜR TAMER** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.



Asist. Prof. Dr. Özgür TAMER

Supervisor



Assoc. Prof. Dr. Levent ÇETİN

(Jury Member)



Asist. Prof. Dr. Nalan ÖZKURT

(Jury Member)



Prof. Dr. Emine İlknur CÖCEN

Director

Graduate School of Natural and Applied Sciences

ACKNOWLEDGMENT

I would like to thank my supervisor Assist. Prof. Özgür TAMER for his technical and mental support throughout my thesis. And, I would like to thank my family who standing with love behind me and giving me priceless support.

Kübra KARADAĞ



PLATFORM DEVELOPMENT FOR PARALLEL OPERATION OF SINGLE BOARD COMPUTERS

ABSTRACT

Recent improvements in technology like mobile devices, Internet of things equipment or scientific and industrial applications generated large amounts of data and necessity of high performance hardware to process this data. Besides, because technological improvements, the processing of large programs, the transformation of enormous data, and the increase of systems which have the simultaneous data flow from interconnected discrete elements have become an inevitable part of daily life. Even though there exist expensive servers there is always a need for all these needed are required fast, affordable, scalable, efficient and flexible solutions. At this point, High-performance computing and parallel programming can meet at this bottleneck to solve data-rich, large-scale programs. Beowulf clustering is one of the high-performance computing approaches which is provides multiprocessing platform to run programs into as divided tasks parts concurrently. In this thesis, a scalable Beowulf cluster consisting of single board computers (SBC) were built and were evaluated its performance is evaluated. Single board computers are preferred as a computing node because of its credit card size, affordability and its ability to meet to the required performance that is needed. The infrastructure operating system of the platform was developed based on Linux operating system. Python was selected preferred as the development environment programming language and MPI was used to the (message passing interface) and MPI4py were used to carry out parallel operations. The platform was tested with problems programs with different characteristics and results were evaluated compared according to with previous studies in this field. Scalable speed up tendency is observed from test results.

Keywords: MPI, parallel computing, single board computer

TEK KART BİLGİSAYARLARLA PARALEL İŞLEM YAPABİLMESİ İÇİN PLATFORM GELİŞTİRİLMESİ

ÖZ

Mobil aygıtlar, bilimsel ve endüstriyel uygulamalar ya da nesnelerin interneti çalışmaları gibi teknolojiye son gelişmeler, bu verileri işlemek için büyük miktarda veri ve yüksek performanslı donanım gerekliliğini yarattı. Aynı zamanda, teknolojik gelişmeler nedeniyle, büyük programların işlenmesi, yoğun data transferleri ve birbirine bağlı eleman unsurlardan eşzamanlı veri akışı olan sistemlerin artması, günlük hayatın kaçınılmaz bir parçası oldu. Bir çözüm yöntemi olarak pahalı sunucuların varlığının yanında, bunların hepsine, hızlı, ekonomik, ölçeklenebilir, verimli ve esnek çözümler gereklidir. Bu noktada, yüksek performanslı bilgi işlem ve paralel programlama, veri açısından zengin, büyük ölçekli programları çözmek için bu darboğazı karşılayabilir. Beowulf kümeleme, programları, bölünmüş görev parçaları halinde çalıştırmak için çoklu işleme platformu sağlayan yüksek performanslı bilgi işlem yaklaşımlarından biridir. Bu tezde, tek kart bilgisayarlardan (SBC) oluşan ölçeklenebilir bir Beowulf kümesi oluşturulmuş ve onun performansı değerlendirilmiştir. Tek kartlı bilgisayarlar, kredi kartı boyutları, uygun fiyatlı olması ve gerekli performansı karşılaması nedeniyle hesaplama düğümü olarak tercih edilmiştir. Platformun altyapı işletim sistemi Linux işletim sistemine dayalı olarak geliştirilmiştir. Python programlama dili olarak seçilmiş ve MPI (mesaj geçiş ara yüzü) ve MPI4py paralel işlemleri gerçekleştirmek için kullanılmıştır. Platform, programlarla test edilmiş ve sonuçlar, bu alandaki daha önceki çalışmalara göre değerlendirilmiştir. Ölçeklenebilir hızlanma eğilimi test sonuçlarından gözlenmiştir.

Anahtar kelimeler: MPI, paralel programlama, tek kart bilgisayar

CONTENTS

	Pages
THESIS EXAMINATION RESULT FORM.....	ii
ACKNOWLEDGEMENTS	iii
ABSTRACT	iv
ÖZ	v
LIST OF FIGURES	ix
LIST OF TABLES	xi
 CHAPTER ONE – INTRODUCTION	1
 CHAPTER TWO – THEORETICAL BACKGROUND.....	5
 2.1 Parallel Computing.....	5
2.2 Flynn’ s Taxonomy.....	6
2.3 Network Topology.....	7
2.3.1 Bus Topology.....	8
2.3.2 Ring Topology.....	8
2.3.3 Star Topology.....	9
2.3.4 Mesh Topology.....	9
2.3.5 Hybrid Topology.....	10
2.3.6 Tree Topology.....	11
2.4 Parallel Programming.....	11
2.5 Distributed Computing.....	16

2.6 Beowulf Cluster.....	17
2.7 Single Board Computer.....	19
2.6 Amdahl' s Law.....	19
CHAPTER THREE – HARDWARE.....	21
3.1 Base Equipment of Thesis	21
3.1.1 Super Pi.....	21
3.1.2 Switch.....	23
3.2 Design of Thesis	24
CHAPTER FOUR- SOFTWARE	26
4.1 Linux.....	26
4.2 Operating System.....	26
4.3 Python.....	27
4.4 MPI	27
4.5 MPI4py.....	29
4.6 Secure Shell (SSH).....	29
CHAPTER FIVE – IMPLEMENTATION.....	31
5.1 Method.....	31
5.2 Matrix Multiplication Test.....	33
5.3 Calculation of Pi Test	34

5.3.1 Formula for Calculation of Pi.....	34
5.3.2 Sequential Program.....	35
5.3.3 Point to Point Program.....	36
5.3.4 Collective Communication.....	38
5.4 Sobel Filter Test Programs.....	39
 CHAPTER SIX– RESULTS	45
6.1 Matrix Multiplication Test.....	45
6.2 Calculation of Pi Test	48
6.3 Sobel Filter Test.....	53
 CHAPTER SEVEN – CONCLUSION	57
 REFERENCES	58

LIST OF FIGURES

	pages
Figure 2.1 Bus topology schema.....	8
Figure 2.2 Ring topology schema.....	9
Figure 2.3 Star topology schema	9
Figure 2.4 Mesh topology schema	10
Figure 2.5 Hybrid topology schema	10
Figure 2.6 Tree topology schema	11
Figure 2.7 Shared memory processing schema.....	12
Figure 2.8 Thread processing schema.....	13
Figure 2.9 Distributed memory processing schema	14
Figure 2.10 Data parallel processing schema	14
Figure 2.11 Hybrid processing schema	15
Figure 2.12 Single program multiple data processing schema.....	15
Figure 2.13 Multiple program multiple data processing schema	16
Figure 2.14 Beowulf Cluster example.....	18
Figure 2.15 Example of characteristic of speedup under Amdahl's Law.....	20
Figure 3.1 Super pi layout.....	21
Figure 3.2 Super pi specifications.....	22
Figure 3.3 The switch that used in thesis	24
Figure 3.4 Built platform in thesis.....	24
Figure 6.1 Execution time reduction.....	46

Figure 6.2 Comparison of speed up between Amdahl's law and matrix multiplication test results.....	47
Figure 6.3 Comparison time of results.....	47
Figure 6.4 Execution time reduction of point to point test program.....	48
Figure 6.5 Error of point to point test program.....	49
Figure 6.6 Execution time reduction of collective test program.....	49
Figure 6.7 Error of collective test program.....	50
Figure 6.8 Comparison of Time Reduction.....	50
Figure 6.9 Error comparison of test programs.....	51
Figure 6.10 Comparison of pi results.....	51
Figure 6.11 Comparison of speed up between Amdahl's law and Pi calculation test results.....	52
Figure 6.12 Baboon face – unprocessed Sobel filter test image.....	54
Figure 6.13 Baboon face - Sobel filtered test image.....	54
Figure 6.14 Execution time reduction graphic of sobel test programs	55
Figure 6.15 Comparison of speed up between Amdahl's law and Sobel filter test results	56

LIST OF TABLES

	Pages
Table 3.1 Specifications of Switch.....	23
Table 3.2 The cost table of platform.....	25
Table 4.1 General Used Communicators.....	28
Table 4.2 Point to Point Communicators.....	29
Table 4.3 Broadcast Communicators	29
Table 5.1 Basic program for copy-id operation.....	33
Table 5.2 The sequential program for calculation pi.....	36
Table 5.3 The point to point program for calculation pi.....	37
Table 5.4 The collective program for calculation pi	38
Table 5.5 Sobel filter operator-weight_x.....	39
Table 5.6 Sobel filter operator-weight_y.....	39
Table 5.7 Sobel filter sequential test program.....	40
Table 5.8 Sobel filter parallel test program.....	42
Table 6.1 Standard deviation values of matrix multiplication test programs' results.....	45
Table 6.2 Comparison time of results.....	46
Table 6.3 Standard deviation values of pi calculation test programs' results.....	48
Table 6.4 Amdahl's law comparison of pi calculation test programs.....	52
Table 6.5 Standard deviation values of Sobel filter test programs' results.....	53
Table 6.6 Amdahl's law comparison of Sobel filter test programs.....	55

CHAPTER ONE

INTRODUCTION

Since more than five decades, technological developments have been growing up gradually. Instinct of understanding life lead humanity to discover the nature and to simulate it to improve life conditions. All these endeavours have triggered to design giant, complex structures with massive infrastructures, to create simulation programs which use tremendous amount of data to analyse problems, to implement control systems etc. (Barney, 2017).

Nowadays because of technological developments, massive programs and systems have become common. For this reason, processing is taking so much time when traditional sequential algorithms are preferred for software design. Many different researches have been conducted in this field. First, developers focused on designing better processors to implement calculation efficiency and reduce the processing time. Then, idea of parallelism came out. Parallel computing systems and parallel programming paved the way of new era of computing (Pacheco, 2011).

After twenty-five years from the first electronic computer named as ENIAC (Electronic Numerical Integrator and Compute) had announced, Intel 4004 was on the market as the first semiconductor microprocessor in 1971 (intel co., n.d.). It can be said that, this development was one of the important milestone of the computing history. It paved the way for building portable, compact design which transforms itself from room-size computer to a single board computer (SBC).

Initial parallel systems were first proposed in 1958 by IBM Cogni and Daniel Slotnick, who introduced the idea that numerical calculations could be done in IBM research (Wilson, 1994). When it came to 1960, the concept of parallelism with the design of parallel architectures that could be programmed in conjunction with E. V. Yevreinov from the Novosibirsk Institute of Mathematics (IMN) gained a new

dimension (Wilson, 1994). In 1964, Daniel Slotnick developed large-parallel machines for use at Lawrence Livermore National Laboratories (Wilson, 1994). In 1983, Goodyear Aerospace developed the Massively Parallel Processor (MPP) for NASA Goddard. In 1985, David Gelner Linda set the foundations for a parallel programming system (Wilson, 1994). In 1986, the Parallel Virtual Machine (PVM) project was developed to enable the use of software required for distributed computers. In 1993, IBM ran the first SP1 Powerparallel system based on the RISC RS / 6000 processor (Akçay & Erdem, 2010) (Wilson, 1994).

Besides improving the hardware, researchers also developed new programming methodologies that are based on running tasks at the same time in multiple processors. At the beginning of parallelism era, in 1967 Gene Amdahl and Daniel Slotnick published an article and introduced "Amdahl's Law" that concerned about limits and possibilities of parallel operating at the AFIPS Conference (Amdahl, 1967). High-performance computing went through many different architectural eras to boost-up speed and scalability of systems. Vector processing systems and Beowulf clustering which are two different approaches to computing gave way to the improvements. UNIX based and shared memory systems kind AIX, HP-UX, Solaris, and IRIX which are RISC infrastructures were produced through Vector Processing Systems (IBM, 1996). And thanks to Beowulf Clustering, more affordable computing elements like x86 architectures based Linux led to the standardization of systems (Sterling, 2001). These both approaches paved the way to produce more reachable and standard systems (Snell, 2014).

Nowadays, supercomputing has developed so much that a list of the top 500 supercomputers in the world is being prepared and this list is updated every 6 months. According to the recently announced, list Sunway TaihuLight, developed by China National Parallel Computer Engineering and Technology Research Centre (NRCPC), has got the best performance in the world. In the test with the Linpack benchmark, the performance of this system was measured as 93 petaflops (Top500 List - June 2017, 2017).

Since the first SBC that is “Dyna-micro” has been launched, the biggest increase in popularity of SBC has paved the way by ArduinoTM products which allow easy using to hobbyist and beginner developer (Build "dyna-micro" an 8080 microcomputer , n.d.) (Arduino , n.d.). After ArduinoTM had begun to dominate the market, at 2008 BeagleBone’s engineers created the BeagleBoard that is low cost and open source (BeagleBone, n.d.). In 2006 in benefit of kids to encourage programming, Raspberry Pi was designed by The University of Cambridge’s Computer Laboratory (Upton, 2011). Its 25 \$ cost made Raspberry Pi to the most known and preferable one of single board computers (Ortmeyer, 2014).

When single board computers have become low cost and small size, these positive factors made it use as computer for parallel operation platform. There are lots of studies in this cluster made by single board computers field. The most known and the very first one is Joshua Kiepert’s project named as “Creating a Raspberry Pi-Based Beowulf Cluster” (Kiepert, 2013). Kiepert used 33 Raspberry Pi board to construct the platform. And he tested the platform with pi calculation programs with using execution time reduction as a benchmark. We could see from the results of this study reduction in the processing time that depends on processors number is noticeable.

“Iridis-pi: a low-cost, compact demonstration cluster” is another study made by Simon J. Cox and his team (Cox, et al., 2014). This project consists of 64 Raspberry Pi Model B. Cox’s team show off some different benchmark tried on and noticeable one of results is scalability of platform is good at large size programs but at small size problems network overheads.

In this thesis, a distributed platform was developed for parallel operations. The main aim of this thesis is building a system which runs parallel programs and analyses its performance with various benchmarks. Through this study, we aim to improve knowledge skill based on experimental experiences.

In scope of these morals, a cluster computer, composed of single board (SBC) computers is built. There are many reasons to choose SBCs as a node. The most important of these reasons is that SBCs are cheaper than other alternatives. Besides, due to its small structure, it provides to construct a portable structure. Moreover, the fact that it does not need to have a cooling system which means extra cost. Besides all these features, having a performance like an average personal computer's is its significant plus too.

The built platform was tested by massively operational programs. First, basic matrix multiplication algorithm was used for testing the system to detect any mistake of platform. Secondly, pi calculation algorithm was run for testing between parallel programming approaches to determine the most effective one. Afterward, Sobel edge detection algorithm of image processing were run as sequential and parallel programs for testing. Finally, the results were examined and conclusions were made.

In the following chapters, thesis's main objects will be explained. At the second section, theoretical background information is given. The peripheral and software descriptions of the platform were expressed in hardware, which is the third part, and software, which is the fourth part, respectively. Then, in implementation chapter, information of how software configurations and testing process carried out is given. Chapter of results focus on test programs outcomes. Finally, in the seventh chapter, the evaluation of the results is given.

CHAPTER TWO

THEORETICAL BACKGROUND

In this chapter, theoretical explanations about project's background will be presented. These explanations contain hardware structure and software structure information. The main headlines of parallel computing and programming will be introduced briefly for preparing the infrastructure.

2.1. Parallel Computing

In the simplest sense, parallel computing could be defined as processing of divided parts of program concurrently.

The steps of solving a problem by parallel programming is given as follows (Barney, 2017):

- The problem is divided into independent pieces that can be solved.
- The separated tasks are sent to the parallel processing elements.
- The processing specified in the processing element is performed.
- The results are combined for the final operations.

The parallel computing systems consist of a computing unit with multiple cores or a number of computers connected by a network (Barney, 2017).

There are basic four models of parallel computing: bit-level parallelism, instruction-level parallelism, data level parallelism and task level parallelism. Bit-level parallelism is done by increasing the word size which is the amount of information that the processor can execute per cycle. Instruction-level parallelism is done by re-ordering

of the program so that the program can be combined into groups which are then executed in parallel without changing the result of the program. Data-level parallelism is based on sending discrete data across computing nodes to run at the same time. Task parallelism divides programs into different tasks which run with same or different data and sends these groups to computing nodes. (David E. Culler, 1999).

Besides boost up the time performance, parallel computing is a very proper approach to design real natural problems which are simultaneous, complex and influenced by each other. It gives a chance to solve problem within several dimensions at the same time and provide to boost up the probability of reaching real solutions.

2.2. Flynn's Taxonomy

There are many ways to define and classify computing platforms on behalf of their main characteristic structure and working mechanisms. One of them and the most well known one is the Flynn's Taxonomy (Flynn, 1966).

Flynn's taxonomy is proposed by Michael J. Flynn in 1966 (Flynn, 1966). This classification model divides architectures into main 4 categories. Classification is made along to the two factors which are instruction stream and data stream.

Flynn's taxonomy divides architecture mainly as;

- SISD single instruction, single data: This machine structure logic is based on traditional computers and can compute just consecutive processes. A data stream is run sequentially by single instruction in a cycle clock (Akhter, 2006).

- SIMD single instruction, multiple data: This structure allows computing multiple data at the same time by a single instruction stream. Digital signal processing, image processing, and multimedia applications such as audio and video are examples of the fields in which these machines are performing (Akhter, 2006).
- MISD multiple instruction, single data: This machine is mainly used as a theoretical structure because generally, multiple instruction streams work better with multiple data. It provides to divide single data to multiple instructions to conduct many tasks on partial data (Akhter, 2006).
- MIMD multiple instruction, multiple data: This machine is the most preferred one of the parallel computing structures. It runs multiple data stream on multiple instruction streams. Processing units are used multiple tasks to run multiple data. This structure used for complex programs which have many parameters and calculation parts to solve the problem (Akhter, 2006).

In this thesis MIMD is used because of the structure has multiple discrete processing nodes (SBCs) and we tried to run one massive program with multiple tasks on this platform. Its advantage is being proper for many programs that have complex logic and tremendous data.

2.3. Network Topology

In parallel computing field, there are many types of computing system approach. One of them is the multiple processing elements compilation. Considering the system, these processing elements are called as nodes and connecting lines which connect them each other. Network topology is the structure organization of the nodes of a computer network.

There many types of network topologies. Mainly network topology recognizes six basic topologies:

2.3.1. Bus Topology

In this very popular topology, connectivity is provided by a single backbone line. All nodes connect this main bus line which can be called as the communication line. Nodes send and receive data to each other via this bus line which has two endpoints.

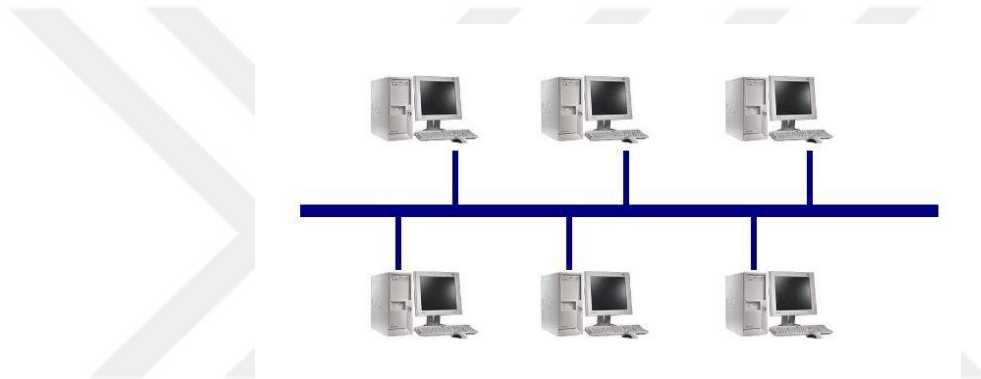


Figure 2.1 Bus topology schema

2.3.2. Ring Topology

In this topology, all nodes connect two of other nodes with links to build a circular structure. All nodes connect just two of another node which are neighbour previous and next ones. These partial lines provide to uninterrupted circular data transmission. Sent data travel all around the circuit until received by other node or nodes.

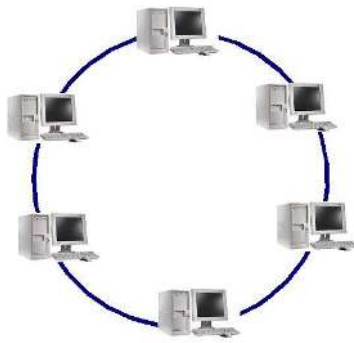


Figure 2.2 Ring topology schema

2.3.3. Star Topology

In this topology, a centre point connects the all nodes to each other. This point is the called the host and end of the all the communication terminals

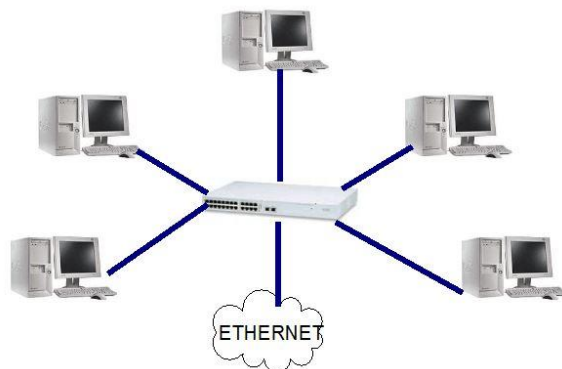


Figure 2.3 Star topology schema

2.3.4. Mesh Topology

When mesh topology is considered there is not any structural position advantages or disadvantages between any nodes. All nodes connect to each other in the network. Besides higher connectivity level but also there are many redundant paths.

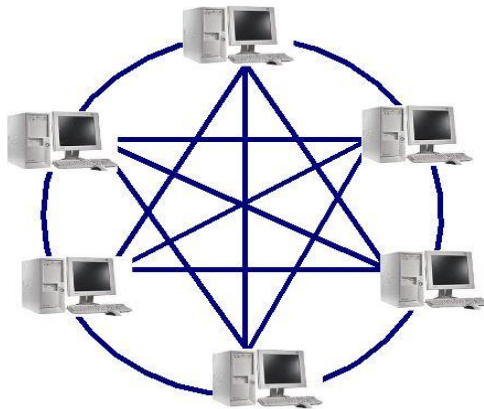


Figure 2.4 Mesh topology schema

2.3.5. Hybrid Topology

Network nodes' interconnections do not have any structural similarities at this topology. Computing equipment connect each other depends on processing necessities.

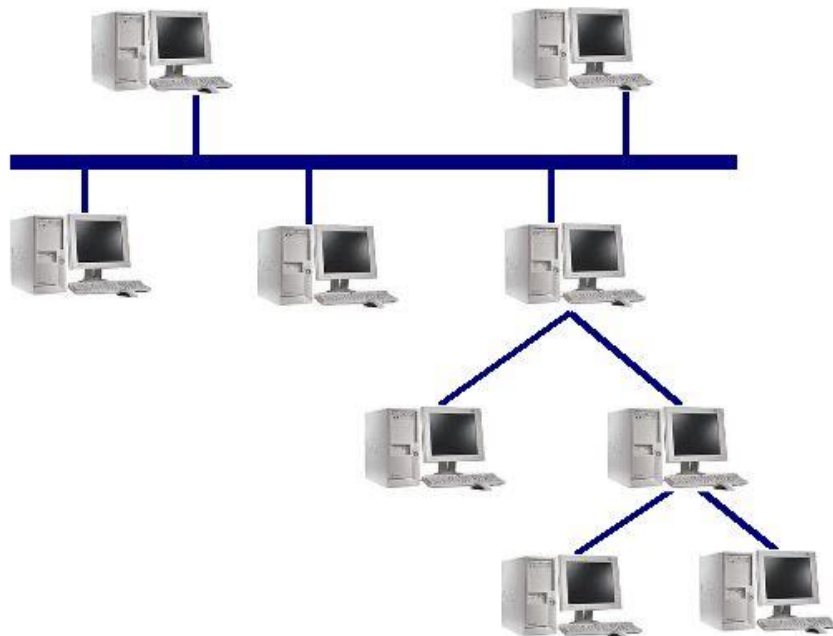


Figure 2.5 Hybrid topology schema

2.3.6. Tree Topology

Tree topologies are the most common network structure. Connection between nodes are very similar with the tree topology. One root is divided into minimum two branches and these branches are divided into many segments too. For all branches, there is only one root to provide connection to springs.

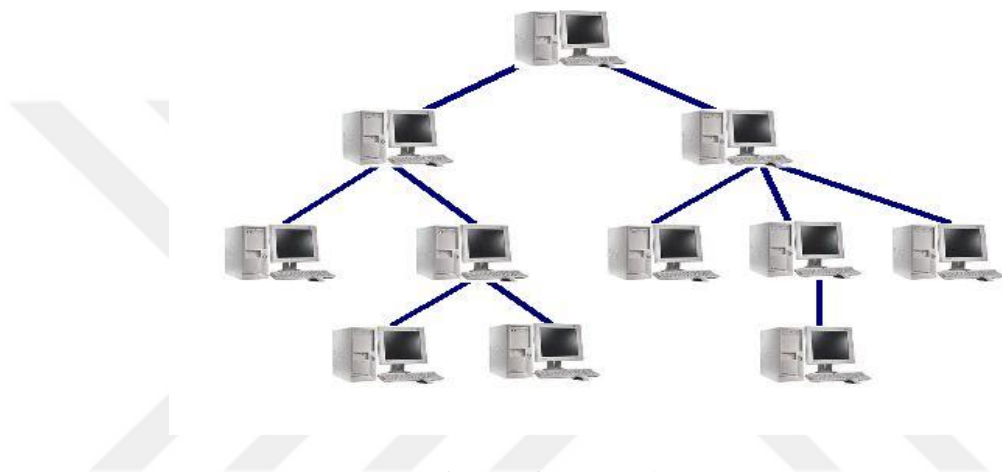


Figure 2.6 Tree topology schema

In this thesis, the star topology selected for building structure because it allows connecting nodes via a controllable switch and this feature gives advantage of flexibility.

2.4. Parallel Programming

Parallel programming is the strategy of the parallel computing that considers software and hardware specifications of the processing system. . The problem type and the computer architecture must be considered for proper computing.

Computing architecture's memory structure and processing patterns effect considerably on logic of parallel programming. For avoiding to lost packages and fault in data flow, shared or distributed memory architecture must have to be dealt in different parallel programming strategies. Parallel programming models divide into many types which are;

- **Shared memory:** In this programming model, tasks share a common address space, which they read and write to asynchronously. Sharing process may occur some problems such as race conditions, deadlocks or confusions about to access memory. The locks/semaphores are used to solve these problems (Barney, 2017).

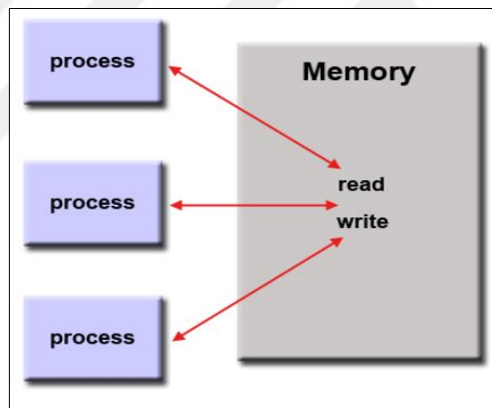


Figure 2.7 Shared memory processing schema (Barney, 2017)

- **Threads:** In this model, the massive task is divided into smaller tasks which such a way as to allow the tasks to be performed. This model is another version of the shared memory model (Barney, 2017).

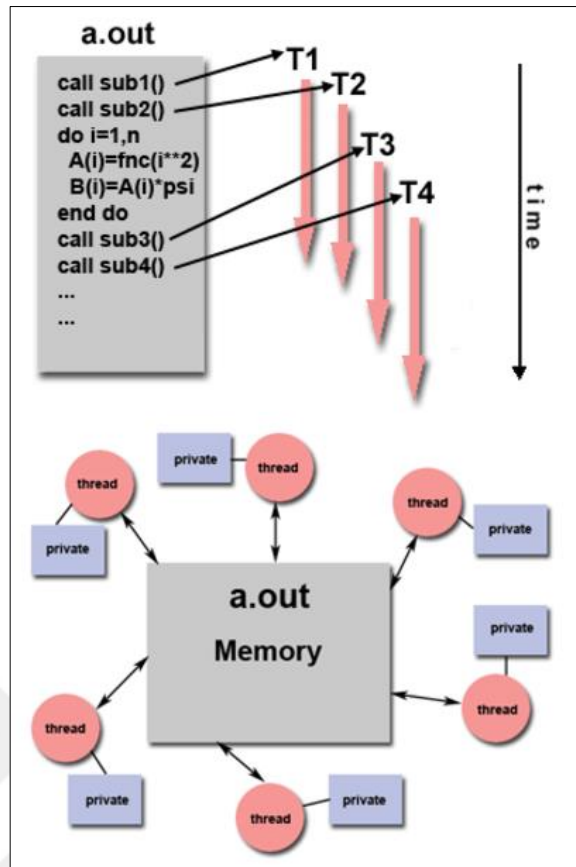


Figure 2.8 Thread processing schema (Barney, 2017)

- Distributed memory / message passing: In this model, during processing data sets use only their own memory. This model allows running multiple tasks on the same computing units or many different computers (Barney, 2017).

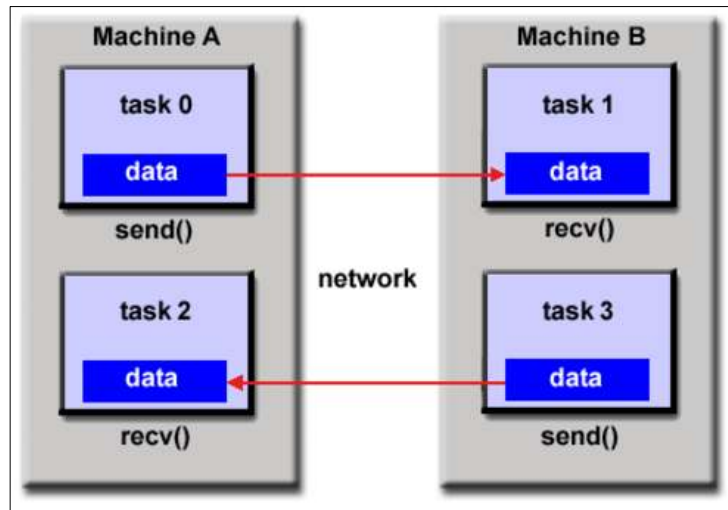


Figure 2.9 Distributed memory processing schema (Barney, 2017)

- Data parallel: this model run parallel programs based on data set. multiple tasks run on distinct parts of the single data set simultaneously (Barney, 2017).

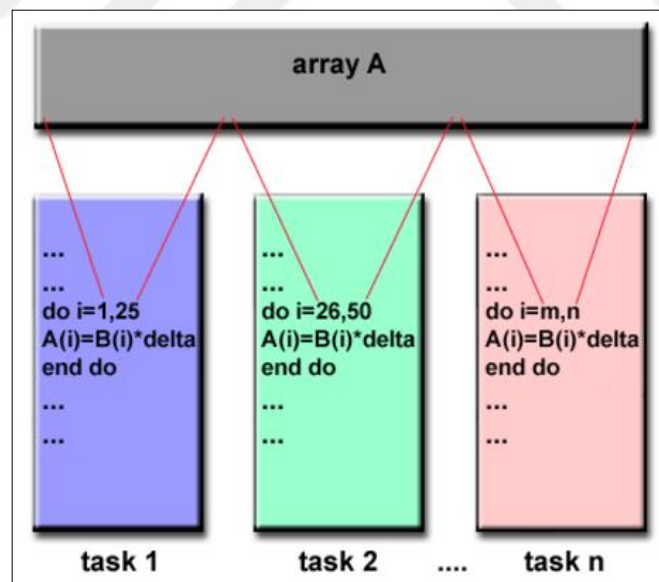


Figure 2.10: Data parallel processing schema (Barney, 2017)

- Hybrid: This model uses at least two various kinds of programming models. The combination of the message passing model (MPI) with the threads model (OpenMP) is a preferred one (Barney, 2017).

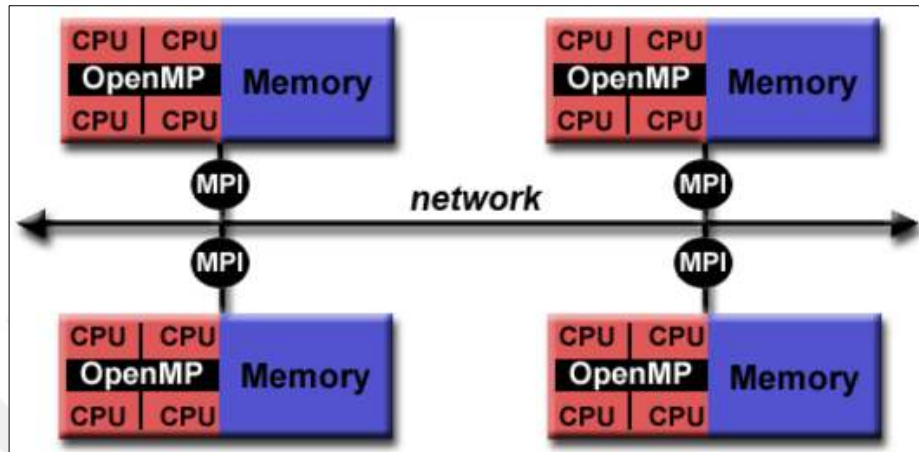


Figure 2.11 Hybrid processing schema (Barney, 2017)

- Single program multiple data (SPMD): This allows running multiple data on distinct parts of single program which is designed properly to have divided task sessions (Barney, 2017).

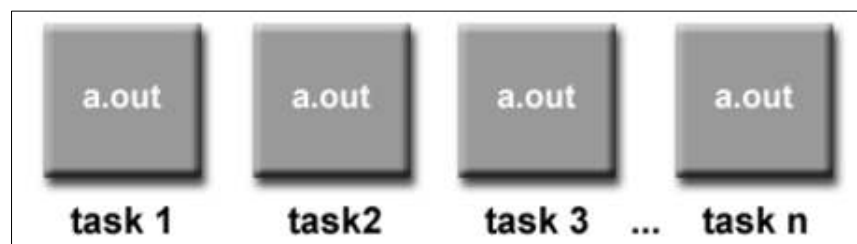


Figure 2.12 Single program multiple data processing schema (Barney, 2017)

- Multiple program multiple data (MPMD): This model allows running multiple data on multiple programs in various combination that necessary to solve problem (Barney, 2017).

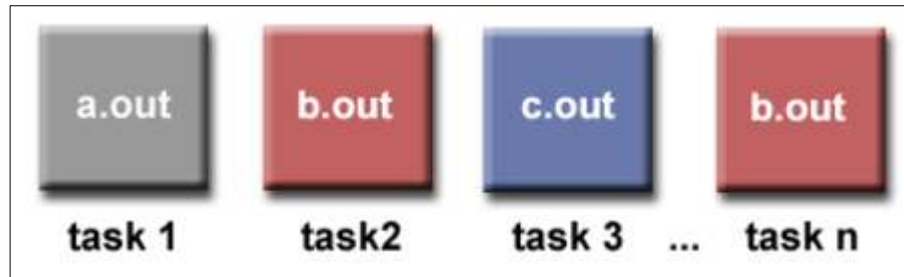


Figure 2.13 Multiple program multiple data processing schema (Barney, 2017)

This thesis scopes on distributed memory architecture because of the structure based on SBCs which have individual memory on their own and so using message passing techniques to communication. Considering that, this topic will be covered in more detail.

2.5. Distirubuted Computing

A distributed system is complementary of computing parts which could be alike computers. These computing parts are sharing none of the equipment as memory, processing unit etc. Besides of sharing no physical parts, nodes being connected each other via a network. This system benefits from the message passing which is the technique of communication and data sharing.

Distributed computing systems can be used for many different purposes. Because of the differences of the required calculation method and the system must be designed differently. For this reason, Distributed computing systems can be installed in two different ways, peer to peer architecture and server- client architecture. Peer to peer is a structure in which each computer node is equal in terms of operation. In the other

architectural structure, the server-client server acts like master node and manages operations and other slave nodes.

As mentioned before, in distributed systems, computing nodes use message passing. The message passing must be done correctly in order for the system to be able to perform a healthy operation. The transmitted packages, which may be any signal, data or command, must be marked by a reference by the sender. This marking contains the knowledge of which recipient should take the packages.

2.6. Beowulf Cluster

At NASA 's The Goddard Space Flight Centre, in 1994, Beowulf Cluster Project which is based on Linux, was produced by Donald Becker, Thomas Sterling, Jim Fischer (Becker, 1999). The main specialities of this project are having free software and of the shelf hardware. From 1994 first launch of it, now there are lots of Beowulf cluster platform all around the world (Brooks, n.d.).

This project has mainly focus on decreasing the budget of building a parallel platform. And according to Dr. Bernard Brooks “This was made possible by two recent developments in technology: firstly, the introduction of cheap Intel and Intel clone microprocessors that could perform respectably compared to DEC's Alpha CPU, Sun's SPARC and UltraSPARC lines, and other high-performance CPUs, and secondly, the availability of capable open-source operating systems, most notably Linux. “(Brooks, n.d.).



Figure 2.14 Beowulf Cluster example (Zinner, 2012)

The original Beowulf parallel workstation, prototyped by NASA, combined sixteen 486DX PC's with dual Ethernet networks, 0.5 GB of main memory, and 20 GB of storage, and providing up to eight times the disk I/O bandwidth of conventional workstations (Hawick, Grove, & Vaughan, 1999). Since the Beowulf design uses commodity hardware components and freely available systems software, NASA's project has demonstrated how the price/performance ratio of this route is attractive for many academic and research organisations. One of the most difficult tasks in designing and commissioning a Beowulf cluster is tracking the cost/performance benefits from the multitude of different possible configuration options (Hawick, Grove, & Vaughan, 1999).

Beowulf clusters have many advantages. One of them is its affordable cost. With technological developments, personal computers have got cheaper prices and that makes them usable as nodes for Beowulf clusters. Therefore, the total cost of building super computing platforms is decreasing. Another preferable specialty of Beowulf clusters is scalability. The main structure of it allows adding processing nodes. Maintaining the connection of Beowulf cluster with hub or switch make it easy to change the design of its topology for many kinds of processes.

All these matters that were mentioned before show that the Beowulf is affordable, flexible and scalable.

2.7. Single Board Computer

Single board computer (SBC) is an integrated circuit that consist of providing structure for many peripheral job (audio, HDMI) in one motherboard.

Although its small size, a single board computer can be used as a normal computer. A single board computer has a microprocessor and peripheral parts which can meet the requirements of a personal computer. SBCs provide abilities for developing systems for educational or professional studies (Cox, et al., 2014).

SBCs were made possible by increasing density of integrated circuits. It has a design without connectors and bus driver circuit that allows minimizing its price. And this design prevents reliability problems which stem from connections (Rosch, 1999).

Nowadays single board computer sector is on top of the technological interest (Global Market Insights, Inc., 2017). Besides techno-developer hobbyists have much attention of them. Especially, being affordable and having satisfactory performance depends on its price make SBC reachable and preferable.

There are many types of SBC on the market to meet different demands. BeagleBone, PandaBoard, Raspberry Pi, Banana Pi, Odroid, The Parallela Board, Nano Pi Neo are the examples of these credit card-sized single board computers.

2.8. Amdahl's Law

Amdahl's law can be used to calculate how much a computation can speed up by running part of it in parallel (Amdahl, 1967).

The focus of this law is the calculation of speed up from making a comparison to serial and parallel part of the program. Its formula depends on comparing these parts' fractions and number of processors (Barney, 2017).

Introducing the number of processors performing the parallel fraction of work, the relationship modelled by Amdahl like (Barney, 2017).

$$\text{speedup} = \frac{1}{\frac{P}{N} + S} \quad (2.1)$$

- P= parallel fraction
- N= number of processors
- S= serial fraction

Amdahl's law provides an upper bound on the speedup that can be obtained by a parallel program: if a fraction r of the original, serial program isn't parallelized, then we can't possibly get a speedup better than $1/r$, regardless of how many processes/threads we use. In practice, many parallel programs obtain excellent speedups. One possible reason for this apparent contradiction is that Amdahl's law doesn't take into consideration the fact that the unparallelized part often decreases in size relative to the parallelized part as the problem size increases (Pacheco, 2011).

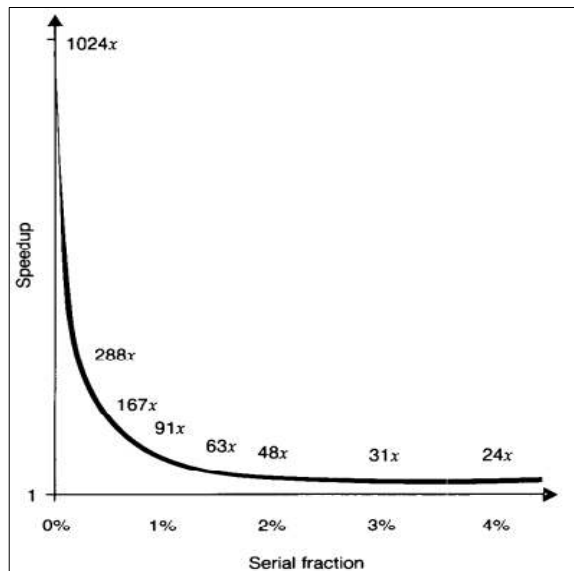


Figure 2. 15 Example of characteristic of speedup under Amdahl's Law (Gustafson, 1988)

CHAPTER THREE

HARDWARE

In this thesis, the main focus is creating a cluster resembling Beowulf with single board computers to implement parallel computing thesis and build an educational training platform for students who are interested in parallel programming. For these matters, 17 single board computers, 1 gigabit ethernet switch have been used for compiling the platform.

In the previous section, this equipment's will be introduced with specific details.

3.1. Base Equipment of Thesis

3.1.1 *Super Pi*



Figure 3.1 Super pi layout (File:Foxconn SuperPI top.JPG, 2016)

The most remarkable equipment is the single board computer for sure. In this thesis, single board computer which could have been called as a node of Beowulf cluster, is performing like a real computer. In this scope if giving some information about single board computer is needed;

In this work, a SBC called Super Pi is used as the nodes of the cluster. Producer company The Foxconn claims that (Foxconn Super Pi, 2016) "The Foxconn Super Pi is essentially a Foxconn redesigned Banana Pi. The overall design shows clear relationship with the Banana Pi but the PCB layout and onboard connector positions

are mostly different.” Super Pi is equipped with 1 GB shared DDR3 and dual-core (ARM Cortex-A7) (Specification, n.d).

For more details of this product layout and specification table given below.

CPU	ARM® Cortex™-A7 Dual-Core
GPU	ARM® Mali400MP2 Complies with OpenGL ES 2.0/1.1
Memory(SDRAM)	1GB DDR3 (shared with GPU)
Onboard Storage	SD(Max.32GB) / MMC card slot SATA port
Onboard Network	10/100/1000 Ethernet
Video Input	A CSI input connector allows for the connection of a designed camera module
Video Outputs	HDMI, CVBS, LVDS/RGB
Audio Output	3.5 mm Jack and HDMI
Power Source	5 volt via MicroUSB (DC In Only) and/or MicroUSB (OTG)
USB 2.0 Ports	2 (direct from Allwinner A20 chip)
Buttons	Reset button: Next to MicroUSB connector Power button: Next to Reset button Boot button (Optional): Next to HDMI connector
LED	Power LED & RJ45
Remote	IR (Optional)
OS	Andrord 4.2 and Linux etc.OS
Weight	48g
Size	91.5 mm x 60 mm

Figure 3.2 Super pi specifications (Specification, n.d)

Besides SBCs the SD cards necessary for platform because of SBC’ s need an extra memory for operating system. Owing to that, for operating system and extra storage for after works, 16 GB micro SD card was preferred.

The operating system installed on memory cards and conducted through memory card adapters.

3.1.2 Switch

The switch is used as the communication infrastructure. Modem is used for IP distributing. These of two gives the communication infrastructure for single board computers.

Table 3.1 Specifications of the switch (Hewlett Packard Enterprise, 2016)

Specifications	HPE OfficeConnect 1920 24G Switch (JG924A)
I/O Ports and Slots	24 RJ-45 auto-negotiating 10/100/1000 ports (IEEE 802.3 Type 10BASE-T, IEEE 802.3u Type 100BASE-TX, IEEE 802.3ab Type 1000BASE-T) 4 SFP 100/1000 Mbps slots (IEEE 802.3u Type 100BASE-FX, IEEE 802.3z Type 1000BASE-X) Supports a maximum of 24 autosensing 10/100/1000 ports plus 4 SFP 100/1000 slots
Additional Ports and Slots	1 RJ-45 console port to access limited CLI port
Physical Characteristics	
Dimensions	17.32(w) x 6.81(d) x 1.73(h) in (44 x 17.3 x 4.4 cm) (1U height)
Weight	4.96 lb (2.25 kg)
Memory and Processor	MIPS @ 500 MHz, 32 MB flash, 128 MB SDRAM; packet buffer size: 512 KB
Performance	
100 Mb Latency	<5 μ s
1000 Mb Latency	<5 μ s
Throughput	41.7 Mpps (64-byte packets)
Routing/Switching Capacity	56 Gbps
Routing Table Size	32 entries (IPv4), 32 entries (IPv6)



Figure 3.3 The switch that used in thesis (HPE, n.d)

3.2 Design

All of the equipment given above were connected to each other as shown in the figure. For start up the initialization and following to process, connection between user and platform are provided through One monitor, keyboard and mouse.

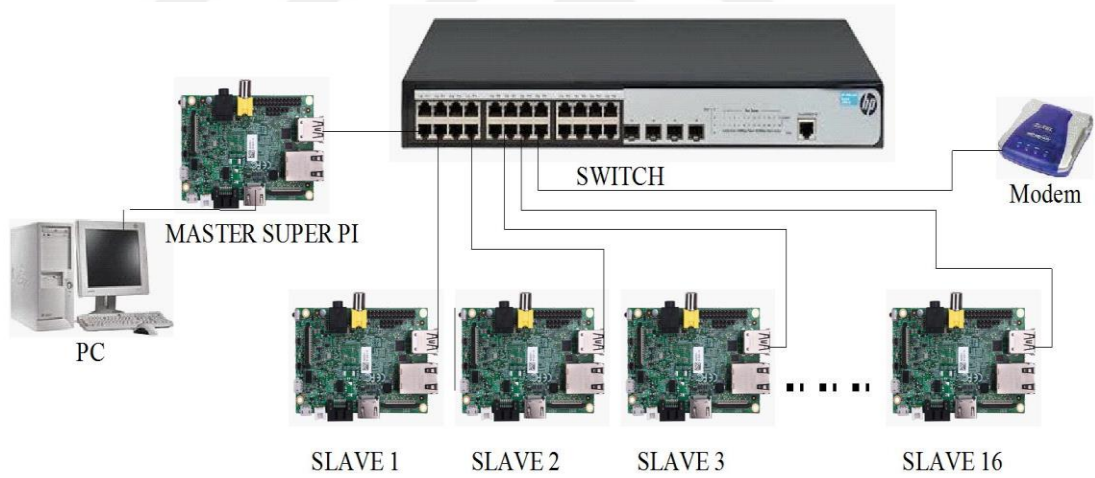


Figure 3.6 Built platform in thesis

As mentioned before at this thesis, one of the advantages of this built platform is its affordable price because of using SBC (Super pi) as processing unit. As seen at Table 3.2 (This table arranged depending on 2017 July market data) price of Super pi is 59,90 TL (LeMaker Super Pi , 2017). If we make a basic comparison between Super pi and Raspberry pi 3 which is one of very popular SBC for a while, we can see that super pi is almost 3 times cheaper than Raspberry pi 3 (Raspberry Pi 3, 2017). Thus,

we can see the correctness of our assertion about affordability in the direction of this market data.

Table 3.2 The cost table of platform

EQUIPMENT	PRICE	QUANTITY	COST
SUPER PI	59, 90 TL	17	1018,3 TL
SWITCH	844 TL	1	844 TL
SD CARDS AND ADAPTERS	29,99 TL	17	509,83 TL
ETHERNET CABLES	8,99 TL	17	152,83 TL
POWER ADAPTERS	10 TL	17	170 TL
TOTAL COST			2694,96 TL

CHAPTER FOUR

SOFTWARE

4.1 Linux

Linux is one of the most popular operating systems in the world. Its popularity stems from its design that is stable and basic but beside of these, the main reason of its reign is it is open source. These features make it preferable for users and company at the market (Sterling, 2001).

In the scope of clustering, Linux is the most proper operating system in any way. Flexible design makes it suitable for design distributed structure. The partial system may offer something special for its own structure and Linux can meet this needs with its easily changeable and arrangeable open source (Sterling, 2001).

Linux is a memory kernel that can easily be compiled to 600 KB of compressed disk. This feature provides using on embedded systems. Being small and simple make Linux proper operating system for Beowulf because smaller structure prevents to bug problems that occur every time of adding code line on the source tree. And small kernel is more likely to be stable that is what Beowulf searching for too. (Sterling, 2001)

In the scope of all these reasons, almost all single board computers have their own Linux-based operating systems.

4.2 Operating System

In this work, due to the fact Super Pi single board computers' resemblance of Banana Pi, Raspbian for Banana Pi operating system which based on Linux, has installed.

Besides of this main reason, Raspbian-like operating systems are much preferable than the others because it gives larger space to find implementation and problem-solving examples (Kiepert, 2013).

4.3 Python

In this thesis, the Python programming language is preferred for writing test programs. One of the reasons why the Python language is chosen is ease of use. Python is a user-friendly language with the modular structure. Modularity allows code use for many times. Python has simple but useful syntax and high-level data structure (Dalcin, The MPI for Python, 2017). Python has become a preferred language because of its easy syntax and fast application capability.

The Python interpreter and comprehensive standard library are available without any charge in source or binary form. Python is a flexible language and proper to be an extension for many areas. This flexibility supports it to easily add new functions or data types which are created with C or C ++ (Dalcin, The MPI for Python, 2017).

4.4 Mpi

In this thesis, Message passing interface (Mpi) is used for data transferring to SBC's. MPI is a specification that provides communication with addresses. The sent data goes through the address directly to recipient processor address to realize processing and this main flowchart repeats during the operation. (Barney, Message Passing Interface (MPI), 2016). This specification is released by MPI Forum for avoiding many problems that developers face with in the past. Lisandro Dalcin evaluated MPI like that (Dalcin, MPI for python, 2015) “message-passing has proven to be an effective one. This paradigm is specially suited for (but not limited to) distributed memory architectures and is used in today’s most demanding scientific and engineering application related to modelling, simulation, design, and signal processing.”

MPI is a specification that provides communication with addresses. The sent data goes through the address directly to recipient processor address to realize processing and this main flowchart repeats during the operation (Barney, Message Passing Interface (MPI), 2016).

MPI uses the communicators for implementations. The base and most uses communicators class is the comm class. Through comm class many kind of parallelization codes can be used.

For parallelization, MPI provides some different methods. One of these methods is Point to Point communication. At this communication method, there is a sequential transferring between nodes. Node, processor or master computer, send data to other node and wait to feedback of receiving information from the node. After this, processing keeps going sequentially.

The other communication style is collective communication. With this method, server node broadcasts data to the other nodes. Data received by the related nodes. This method is beneficial in terms of time saving. Because master doesn't waste time for sending data which are sent at the same time to all slaves. And master does not need to get any acknowledgment for skip to next tasks.

MPI has 256 communicators for different processing but next section the most used ones are given for information.

Table 4.1 General Used Communicators

COMMANDS	EXPLEMENTATIONS
MPI_Init	Starter of the program
MPI_Finalize	Used for finishing the program
MPI_Comm_size	Used for obtained number of process
MPI_Comm_rank	Used for obtained to process rank

Table 4.2 Point to Point Communicators

COMMANDS	EXPLEMENTATIONS
comm_send	Used for sending data
comm_recv	Used for receiving data

Table 4.3 Broadcast Communicators

COMMANDS	EXPLEMENTATIONS
comm_scatter	Scatters data from one process to all processes
Comm_gather	Gathers data from all processes to one process
comm_bcast	Sends data from one process to all processes
Comm_barier	Synchronizes all processes

4.5 Mpi4py

Mpi4py is a package that used for supporting MPI standards when working with python programming language (Dalcin, MPI for python, 2015).

Mpi4py provides binding of the MPI standard for the Python programming language. It ensures to access multiple processing nodes which needed to use by Python. Through Mpi4py many communicators or codes are not needed to use. The programs start and stop to work automatically. For example, MPI_init that is used for initialization and MPI_Finalize that stops the program aren't needed when Mpi4py is using (Dalcin, The MPI for Python, 2017).

4.6 Secure Shell (SSH)

SSH, or Secure Shell, is a communication protocol for remote filing, backup and data exchange. The most important feature that separates SSH from other communication protocols is focusing more security on communication.

SSH uses 2 kinds of encryption when providing communication. First one is the public key and the other is the private key. These keys allow to communicate between channels. Another pleasant thing about SSH is that you do not have to do password / username processing once you start secure communication.

In this thesis, SSH is used for communication between SBCs for parallel programming. More detailed information on how the connection is established in the implementation section is given.



CHAPTER FIVE

IMPLEMENTATION

In this chapter, implementation of thesis' s steps will be introduced. This session contains information about the configuration of software, designing, and execution of test programs.

5.1 Method

For the first step of setting up node Raspbian for Banana Pi operating system was installed on the memory card.

At the first time boot up operations, basic configurations which contain storage configurations, SSH activation, password and hostnames changing must be performed.

The platform need a parallel operation program and tools also. As mentioned before, the platform that built has distributed structure and so this it needs PVM or message passing techniques for executing communication operations. Message passing technique was chosen because in the literature there are so many comparable studies which were using it. Due to this reason Mpi and Mpi4py programs were and have installed on single board computer.

The most common API for programming distributed-memory systems is message-passing. In message-passing, there are (at least) two distinct functions: a send function and a receive function. When processes need to communicate, one calls the send and the other calls the receive. There are a variety of possible behaviours for these functions. For example, the send can block or wait until the matching receive has started, or the message-passing software can copy the data for the message into its own storage, and the sending process can return before the matching receive has started. The most common behaviour for receives is to block until the message has been received. The most commonly used message-passing

system is called the Message-Passing Interface or MPI. It provides a great deal of functionality beyond simple sends and receives (Pacheco, 2011).

After this step, the first configured single card computer memory card was ready to be copied as a basic configured card to all cluster nodes.

Accordingly each of the slave nodes were assigned static unique IP numbers to individualize them. Then hostnames and hosts were changed with new ones which describes themselves with their number.

For realize these, hosts, hostname and interfaces file which are in etc/network direction were changed with new names and IP numbers.

The next step was SSH configurations for none encrypted communication. The Secure Shell (SSH) protocol is preferred for providing communication between the master node and the slave nodes. The proper login credentials were entered to the master node for making a no encrypted connection to the slave nodes as needed. To accomplish this, we added SSH credentials to our master node. In the master node, the SSH key was generated. Then from master node to slave node pinging operation was done. At this moment, properly working slave node recommended as the key for login so previously generated log-in credential was entered to the slave node. After this operation, login of the slave without password was done.

The ssh-copy-id operation was done to ensure that all nodes are in not encrypted contact with the master node. In order realizing this a basic program called as copyid.sh which is on below were written.

Table 5.1 Basic program for copy-id operation

```
copyid.sh
#!/bin/bash
IP_HEADER="192.168.1.1"
ip_list=(01 02 04 05 06 07 08 09 10 11 12 13 14 15 16 17 25)
for i in ${ip_list[@]}
do
    ssh-copy-id bananapi@${IP_HEADER}$i
done
```

In the following step, a file was created in which IP addresses are written for use by Mpi when performing parallel operations. The file is a basic text file which contains the IP addresses in order that used by MPI for the rank number. According to this file, the first order number is recognized as a master and its rank is zero.

After all these steps, a test programs for parallel operation was written to evaluate cluster performance.

5.2 Matrix Multiplication Test

In this thesis, the test programs are developed for evaluating the performance of the cluster computer. Master-slave operation has two kinds of nodes. One of them is a master node, which runs base processing parts. It initializes the programs, sends data, sending process depends on parallelization design, and receives data to find out the results.

In matrix multiplication test program, the master node creates a massive random array and divide this array into the number of slave nodes. Then it sends divided operations to the slaves for computing. After slaves have performed necessary operations, all results are sent to the master node for printing. Better note these;

- Slaves compute the square of elements of divided array.
- Master does distribute operations and monitor nodes only.

First, the test program was run on only one single board computer. Then program was run on 1 master with 2, 4, 8, 16 single board computer slaves respectively. The running operation was conducted with the mpiexec command. That command contains the file of ip numbers which used for operation and number of slave size and the test program file. The files must be written with its directions. .

5.3 Calculation of Pi

In the next parts, formulation of the test program and its execution models which depend on different communication approaches will be presented.

5.2.1 Formula for Calculation of Pi

There are many approaches for calculating pi in the literature. The formula that was used in this thesis is given below. this formula was chosen because of its simple and iterative mathematical expression. The formula benefits from the sum of discrete rectangular areas, using n intervals to compute an approximate pi value (Module for Monte Carlo Pi, n.d).

$$\pi = \int_0^1 \left(\frac{4}{1+x^2} \right) dx \approx \frac{1}{n} \cdot \sum_{i=0}^{n-1} \frac{4}{1+\left(\frac{i+0.5}{n}\right)^2} \quad (5.1)$$

Using the above formula, pi calculation has run with 3 different python programs. All calculations were repeated for 60 times to reach average values. There are 3 values as output from all programs. These;

- Calculated pi value
- Error according to calculated value
- Calculation time

Programs:

- Sequential program: Program only run on one SBC.
- Collective program: Program is Broadcasted to SBCs and results are collected accordingly
- Point to Point Program: The master card sends and receives individual data to each slave . In Here, for continuing to other tasks, the master node waits for the feedback from the recipient. after receiving the master, the feedback is received.

Each collective and Point-to-Point Programs were worked on 4 separate sets of 2 - 4 - 8 - 16 slaves. As a result, there were 9 different sets of data, 1 consecutive, 4 collective and 4 point to point.

5.2.2 Sequential Program

The program is run sequentially on a single SBC to calculate Pi. First, the necessary libraries are imported and the parameters are defined. At the calculation section, formula parts are computed n times in a loop. in the end, result, error and processing time are calculated and printed.

In the error calculation part, math.pi has used. If highlighting is needed for clear understanding, the math.pi gives a (Python software foundation, n.d.) pi value that has 15 digits after point and its value is 3.141592653589793.

Table 5.2 The sequential program for calculation pi

```
#Initialization of The Program

import math
import time
Start_time=time.time()
sum=float(0.0)
n=10000000                                # Iteration number

#Calculation part of program

def Pi_Function(n):
    global sum
    for i in range (n):
        x=(1.0/n)*(i+0.5)
        sum+=4.0/(1.0+x**2)
    return sum*(1.0/n)
Pi=Pi_Function(n)
ERROR=abs(Pi-math.pi)                    # Calculation of Error using math.pi
that has 15 digit after the point

# Finalization of program

End_time=time.time()
Elapssed_Time=End_time-Start_time        # Calculation of processing
time
print(Pi,ERROR,Elapssed_Time)
```

5.2.3 Point to Point Program

The formula is carried out by mpi4py using point to point communication operators. As mentioned before, main processes are send and receive. Firstly, MPI.COMM_WORLD communicator is defined to use these group of operators. After detection of the rank and the size of the platform, iteration number and other needed parameters as using for flag or counting number are defined.

At calculation part, if rank equals to zero it means that, program runs at the master processor, then it recognizes and calculates values which are needed to be sent. Master node stores these into the buffer. After preparation is done, respectively with processors' ranks, data packages are sent. Finally, to continue the process, the master node needs to get acknowledgment from the receiver node. Once receiving feedback comes up, it means that data transformation is done and master could run next task

(The issue explained over the master node but flowchart is same for every node who send to data to other with comm. send operator.).

Once processing is done, result data is sent back to the master by the slave node in the same way. Master node receives data and does finalization of the program or performs another task.

Table 5.3 The point to point program for calculation pi

```
#Initialization of The Program
from mpi4py import MPI
import time
import math
comm=MPI.COMM_WORLD
rank=comm.rank
size=comm.size
n=10000000
rankvalue=1
sum=0.0
k=0.0
#Calculation part of program
if rank ==0:                                #Master's part
    for x in range(1,size):                 #Distributing boundary values for
partial                                     calculation
        a=((rankvalue-1)*(n/(size-1))+1)
        b=(rankvalue*(n/(size-1)))
        c=(a,b)
        Start_time=time.time()
        comm.send(c,dest=x)
        rankvalue=rankvalue+1
    for y in range (1,size):
        data=comm.recv(source=y)
        sum=sum+data
    Pi= sum * (1 / n)
    ERROR=abs (Pi - math.pi)

# Finalization of program
    Elapssed_time= time.time() - Start_time
    print (Pi, ERROR, Elapssed_time)

else :                                      # Slave's part
    c=comm.recv(source=0)
    a,b=c
    for i in range(a, b):                   # Implementation of
formula
        x= (1/n) * (i + 0.5)
        k+=4.0/(1.0+x**2)
    data=k
    comm.send(data, dest=0)
```

5.2.4 Collective Communication

In the collective communication part, broadcast type communication was preferred. Beginning of the program is very similar to the point to point communication program because collective communication and point to point communication contexts belong to the same communicator that is MPI.COMM_WORLD.

Table 5.4 The collective program for calculation pi

```
#Initialization of The Program
from mpi4py import MPI
import math
import time
comm=MPI.COMM_WORLD
size=comm.size()
rank=comm.rank()
Start_time=time.time()

#Calculation part of program
def Pi_Function(n, start=0, step=1):
    sum=0.0
    for i in range (start,n,step):
        x=(1.0/n)*(i+0.5)
        sum+=4.0/(1.0+x**2)
    return sum*(1.0/n)

if rank ==0:          # Master's part
    n=10000000

else :                # Slave's part
    n= None
#Distributing boundary values for partial calculation
n=comm.bcast(n, root=0)
Pi_partial= Pi_Function(n, rank, size)
Pi= comm.reduce(Pi_partial, op=MPI.SUM, root=0)

# Finalization of program
if rank==0:          # Master's part
    ERROR=abs(Pi - math.pi)
    End_time=time.time()
    Elapssed_Time=Start_time-End_time      #Calculation of processing
time
    print (Pi, ERROR, Elapssed_Time)
```

After initialization, calculation function is designed as a sequential program. Then, from the master node, iteration number is broadcasted to all processors via the comm.bcast operator. In this operation, it is well being to know that the sent data is the

same for all processors. The calculation has executed between boundaries which have computed with rank and size numbers.

Once calculation has been done, results are summed up with MPI.SUM operator and this sum is reduced by the comm.reduce operator. And finalization task is run by the master node.

5.3 Sobel Filter Test Programs

Digital image processing methods are software methods used to make an image better or to rearrange with an image for different purposes. The Sobel filter is also a method of providing edge detection in an image using the following weight x and weight y matrices. The matrices are multiplied by the pixel values of the image being traversed on the image. The processed values are summed and the result is written in the middle pixel. In this method, pixels' sharpness weight is determined by using neighbor relations (Gonzales & E., 2002).

-1	0	1
-2	0	2
-1	0	2

Table 5.5 Sobel filter operator-weight_x

1	2	1
0	0	0
-1	-2	-1

Table 5.6 Sobel filter operator- weight_y

This method requires a lot of processing level, and these processes are suitable for parallel processing because they are independent processes from each other. For these reasons, this method's algorithm is chosen for testing the platform performance.

The Sobel Filter program is organized in two ways. One of these is the sequential program. In this program, pixel values are taken after reading the image, and these values are subjected to the Sobel filter operation with the weight_x and weight_y matrices. The obtained new pixel values are replaced by image's pixel values.

Table 5.7 Sobel filter sequential test program

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from PIL import Image, ImageFilter
import math
import time
def pixel(image, x, y):
    try:
        return max(image.getpixel((x, y)))
    except IndexError as e:
        return min(image.getpixel((x, y)))
def run():
    image= Image.open("baboon.bmp")
    Sobel_Filter(image)
    image=image.convert('RGB')
    image.save('RESULTIMAGEseq.bmp')
def Sobel_Filter(image):
    gx = [[-1, 0, 1],
          [-2, 0, 2],
          [-1, 0, 1]]
    gy = [[1, 2, 1],
          [0, 0, 0],
          [-1, -2, -1]]
    width, height = image.size
    gradient_magnitudes = [[0 for x in range(0,width)] for y in
range(0,height)]
    for y in range(1, (height-1)):
        for x in range(1, (width-1)):
```

```

        Weight_y = (
            gy[0][0] * pixel(image, x - 1, y - 1) +
            gy[0][1] * pixel(image, x, y - 1) +
            gy[0][2] * pixel(image, x + 1, y - 1) +
            gy[2][0] * pixel(image, x - 1, y + 1) +
            gy[2][1] * pixel(image, x, y + 1) +
            gy[2][2] * pixel(image, x + 1, y + 1)
        )
        Weight_x = (
            gx[0][0] * pixel(image, x - 1, y - 1) +
            gx[0][2] * pixel(image, x + 1, y - 1) +
            gx[1][0] * pixel(image, x - 1, y) +
            gx[1][2] * pixel(image, x + 1, y) +
            gx[2][0] * pixel(image, x - 1, y - 1) +
            gx[2][2] * pixel(image, x + 1, y + 1))

        gradient_magnitude =
math.ceil(math.sqrt(pow(Weight_x, 2) + pow(Weight_y, 2)))
        gradient_magnitudes[y][x] = gradient_magnitude
    for x in range(0, width):
        for y in range(0, height):
            gradient_magnitude = gradient_magnitudes[y][x]
            pixel_value =
int(math.floor(gradient_magnitude%255))
            image.putpixel((x,y),pixel_value)

    tic=time.time()
    run()
    delta=time.time()-tic
    print("Time Elapsed:", delta)

```

In the second program, parallel test program, image pixel values are generated, then this pixel array is divided by the number of processors in the platform, and Sobel filter is applied on this partial image. Finally, the new pixel values are sent to the master processor and a new image is created.

Table 5.8 Sobel filter parallel test program

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from mpi4py import MPI
import time
import numpy as np
from PIL import Image, ImageFilter
import math
comm = MPI.COMM_WORLD
rank = comm.rank
size = comm.size
def find_num_blocks(n):
    if n == 3:
        return 1, 2
    elif n == 5:
        return 2, 2
    elif n == 9:
        return 2, 4
    elif n == 17:
        return 4, 4
def Pixels(image):
    height, width = image.size
    image_pixels = np.zeros((height, width))
    for x in range(1, (height-1)):
        for y in range(1, (width-1)):
            image_pixels[x,y]=max(image.getpixel((x, y)))
    return image_pixels
def Sobel_Filter(image, n):
    height, width = image.shape # Get figure shape
    num_row_blocks, num_col_blocks = find_num_blocks(n)
    partw = int(width / num_col_blocks) # Block size
    parth = int(height / num_row_blocks)
    # Initialize gradients
    gradient_magnitudes = np.zeros((height, width))
    processed_image = Image.new('RGB', (height, width), 'black')
    processed_image.save('empty_image.bmp')
    record_table = dict()
```

```

if rank == 0: # If the master...
    # Send image blocks
    mark = 1
    for x in range(0, num_row_blocks):
        for y in range(0, num_col_blocks):
            sendarray = image[x * parth: (x + 1) * parth , y *
partw: (y + 1) * partw ]
            comm.send((sendarray, mark), dest=mark)
            record_table[str(mark)] = (x + 1, y + 1)
            mark = mark + 1

    # Receive filtered image blocks
    for k in range(1, size):
        gradient_magnitude, mark = comm.recv(source=k)
        mark = str(mark)
        x, y = record_table[mark]
        gradient_magnitudes[(x - 1) * parth: x * parth, (y - 1) *
partw: y * partw] = gradient_magnitude
    else: # Else, it is slave..
        (image, mark) = comm.recv(source=0)
        def Sobel_Filter2(image):
            block_height, block_width = image.shape
            filtered_image = np.zeros((block_height, block_width))
            gx = np.array([[ -1, 0, 1],
                           [ -2, 0, 2],
                           [ -1, 0, 1]])
            gy = np.array([[ 1, 2, 1],
                           [ 0, 0, 0],
                           [ -1, -2, -1]])
            for x in range(1, (block_height-1)):
                for y in range(1, (block_width-1)):
                    Weight_x = np.sum(image[x - 1:x + 2, y - 1:y + 2]
* gx)
                    Weight_y = np.sum(image[x - 1:x + 2, y - 1:y + 2]
* gy)
                    gradient_magnitude = math.ceil(math.sqrt(Weight_x
** 2 + Weight_y ** 2))
                    filtered_image[x, y] = gradient_magnitude
            return filtered_image

```

```

        filtered_image = Sobel_Filter2(image)
        comm.send((filtered_image, mark), dest=0)
    for x in range(height):
        for y in range(width):
            pixel_value = int(math.floor(gradient_magnitudes[x, y] %
255))
            processed_image.putpixel((x, y), pixel_value)
        return processed_image
def run():
    image = Image.open("baboon.bmp") # Open image
    image_pixels = Pixels(image) # Get pixels in an array
    processed_image = Sobel_Filter(image_pixels, size) # Apply sobel
masks
    processed_image.save("RESULTIMAGE.bmp")
    tic = time.time()
    run()
    print('Time elapsed: ', time.time()-tic)

```

CHAPTER SIX

RESULTS

In this chapter, the results of the test programs will be exhibited and evaluation of them will be done.

6.1 Matrix Multiplication Test

When single board computers work, there are lots of processing behind the scene. At the embedded system, detection or avoiding this kind of processes are not easy when special processing keeps going.

Due to these reasons, to minimize these effects and evaluate proper results, every step was repeated for 60 times and standard deviation of execution times were calculated and shown at Table 6.1.

Table 6.1 Standard deviation values of matrix multiplication test programs' results

	1 SBC	2 SLAVES	4 SLAVES	8 SLAVES	16 SLAVES
Standard Deviation Value	0,102	0,218	0,113	0,489	2,851

As seen the graphic on below, there is a recognizable decrease on the processing time and noticeable speed up which is proportional to the number of slaves.

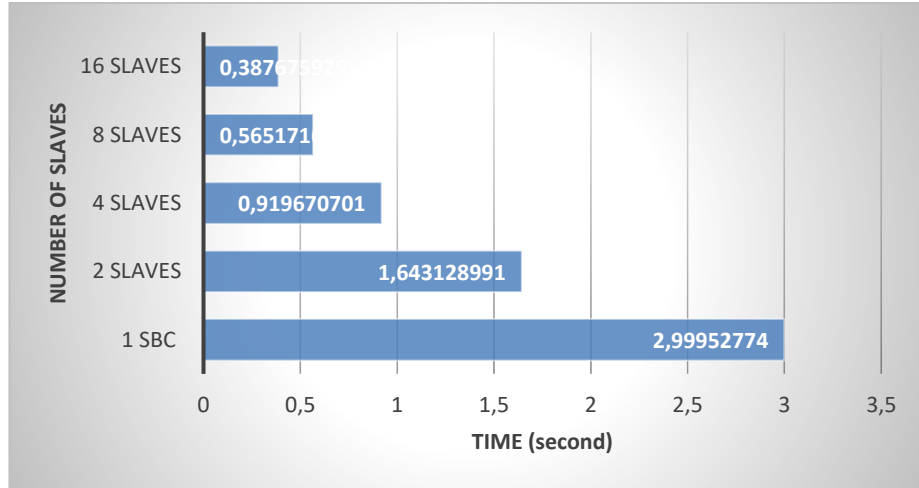


Figure 6.1 Execution time reduction

In this project, in scope of written test program parallel fraction is evaluated as % 93 and serial fraction is % 7. These fractions detected by using total time and divided parts' processing time comparison. The total execution time of program is 2,99952774 second. Beside, the execution time of the parallelized part of program is 2,7815928 second. Assume that in sequential program the whole program's processing time is (Time.main), divided part's processing time is (Time.parallelpart) and the parallel fraction is (Fraction.parallel); the formula of relationship between them could be written as on below;

$$\text{Fraction.parallel} = (\text{Time.parallelpart}) * 100 / (\text{Time.main}) \quad (6.1)$$

When the formula conducted with given values, the parallel fraction found as 92,73% which could assume as 93%. When Amdahl's Law was conducted for this proportions for all of cluster sizes. The comparison table and graphic were presented on Table 6.1. below.

As seen on below at table and comparison graphic, calculated and measured result nearly meet each other.

Table 6.2 Comparison time of results

	2 SLAVES	4 SLAVES	8 SLAVES	16 SLAVES
AMDAHL'S LAW	1,87	3,31	5,37	7,8
TEST RESULTS	1,83	3,26	5,31	7,74



Figure 6.2 Comparison of speed up between Amdahl's law and matrix multiplication test results

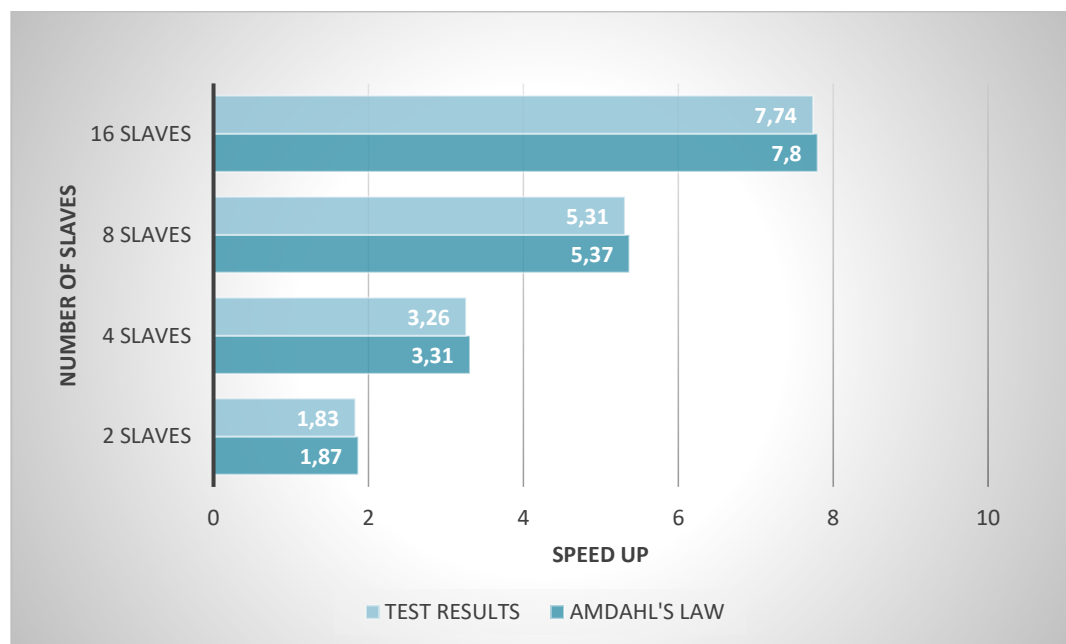


Figure 6.3 Comparison time of results

6.2 Pi Calculation Test

The point to point test programs' results are given below. The test programs run 50 times for every size of platform and standard deviation of execution times were calculated and shown at Table 6. . First graphic is Figure 6.3 that shows the execution time reduction depends on number of slaves which means size of processing platform. As seen from graphic there is a noticeable logarithmic reduction size by size.

Table 6.3 Standard deviation values of pi calculation test programs' results

	SEQUENTIAL	POINT TO POINT				COLLECTIVE			
	1 SBC	2 SLA VES	4 SLA VES	8 SLA VES	16 SLA VES	2 SLA VES	4 SLA VES	8 SLA VES	16 SLA VES
Standard Deviation Value	0,3	0,365	0,182	1,433	0,543	0,157	0,111	0,471	0,548

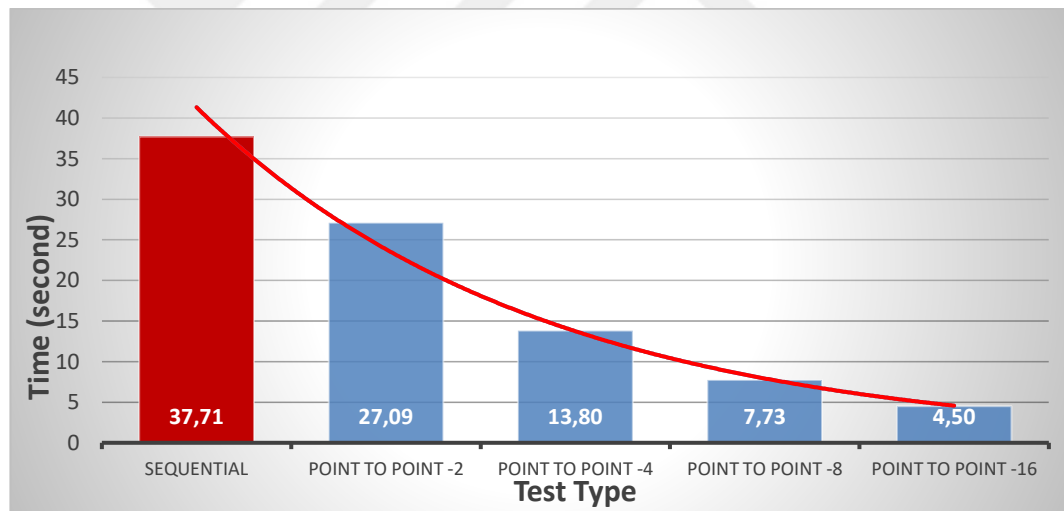


Figure 6.4 Execution time reduction of point to point test program

Error graphic of point to point test program results showed consistent within itself. Every node that is added on the system were increased in calculation success.

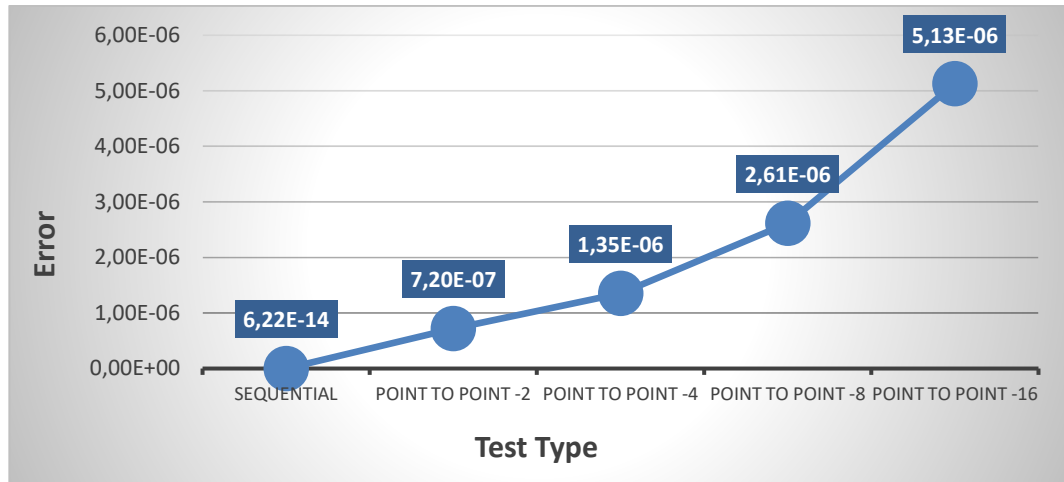


Figure 6.5 Error of point to point test program

At figure 6.5 the execution time reduction of collective test programs' results are given. Likely point to point test program results, there is a satisfied speed up could see clearly.

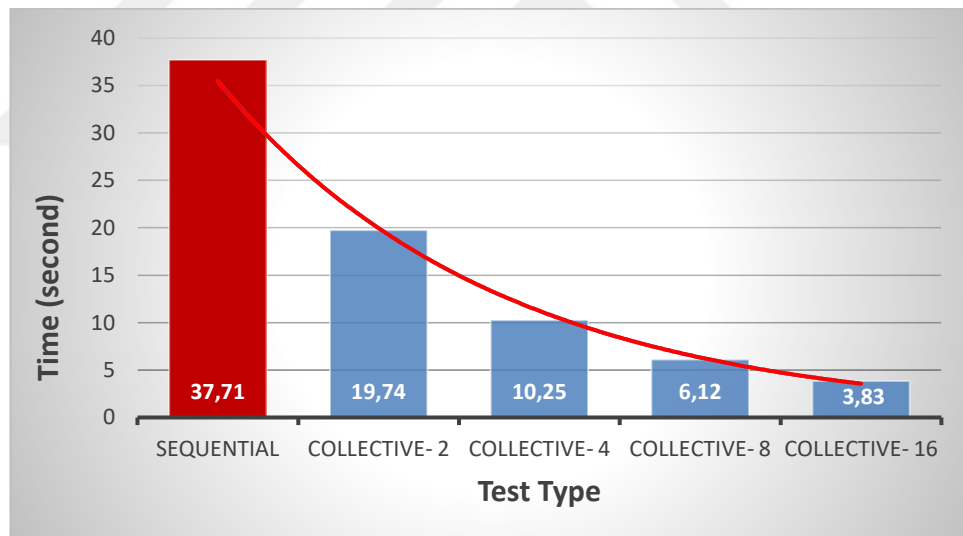


Figure 6.6 Execution time reduction of collective test program

Error results of collective test program is given below at figure 6.6. The rate of reduction in error after each node insertion in the platform that has been showing at the graphic is a very noticeable.

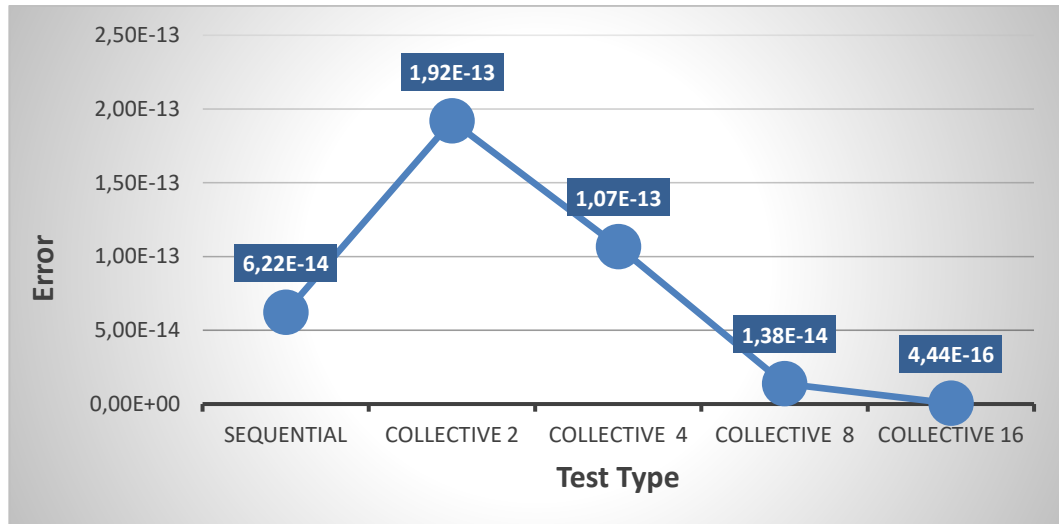


Figure 6.7 Error of collective test program

From graphics on below we can see the comparison between sequential, collective and point to point programs.

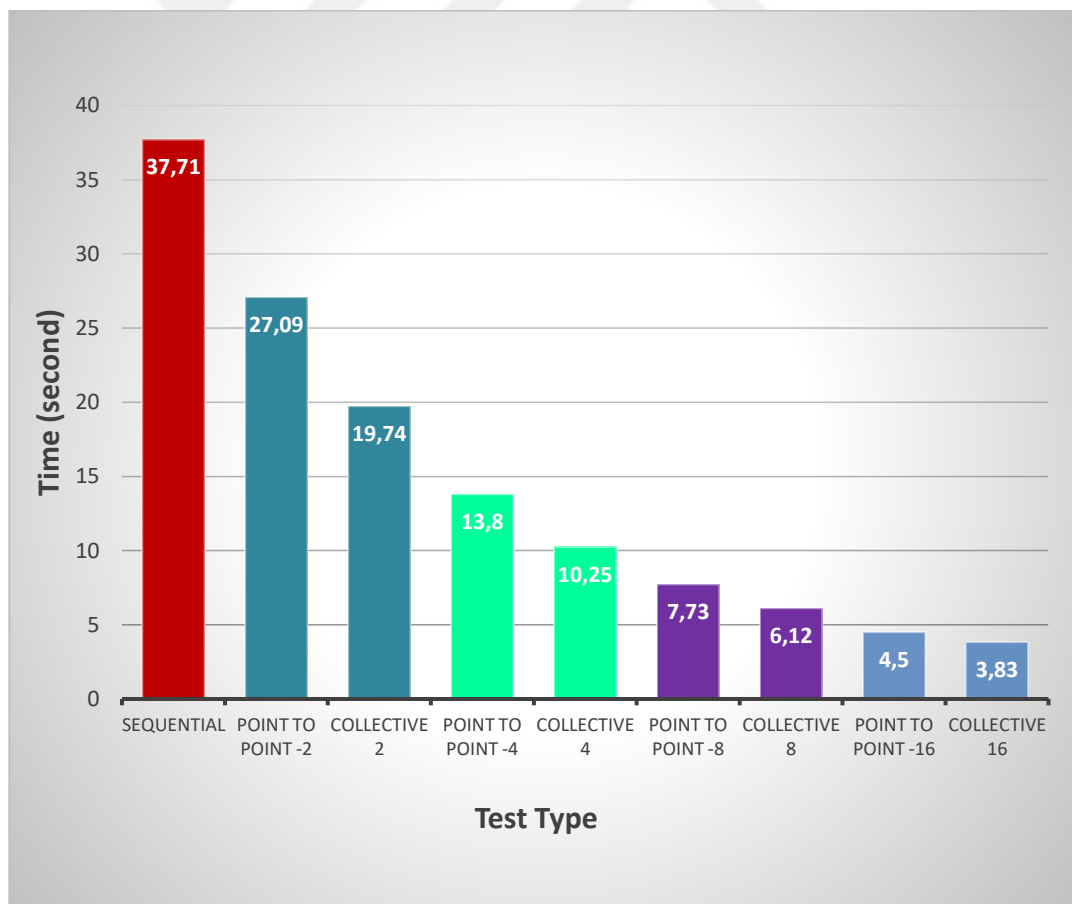


Figure 6.8 Comparison of Time Reduction

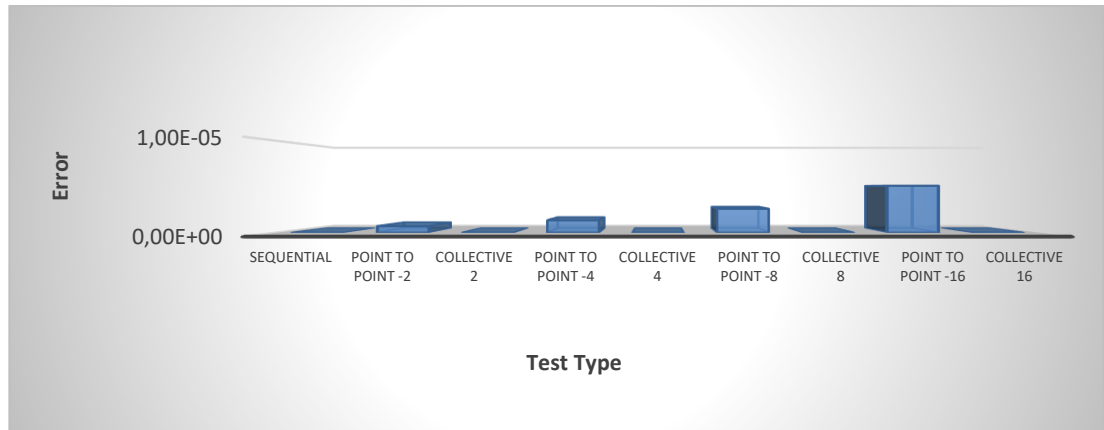


Figure 6.9 Error comparison of test programs

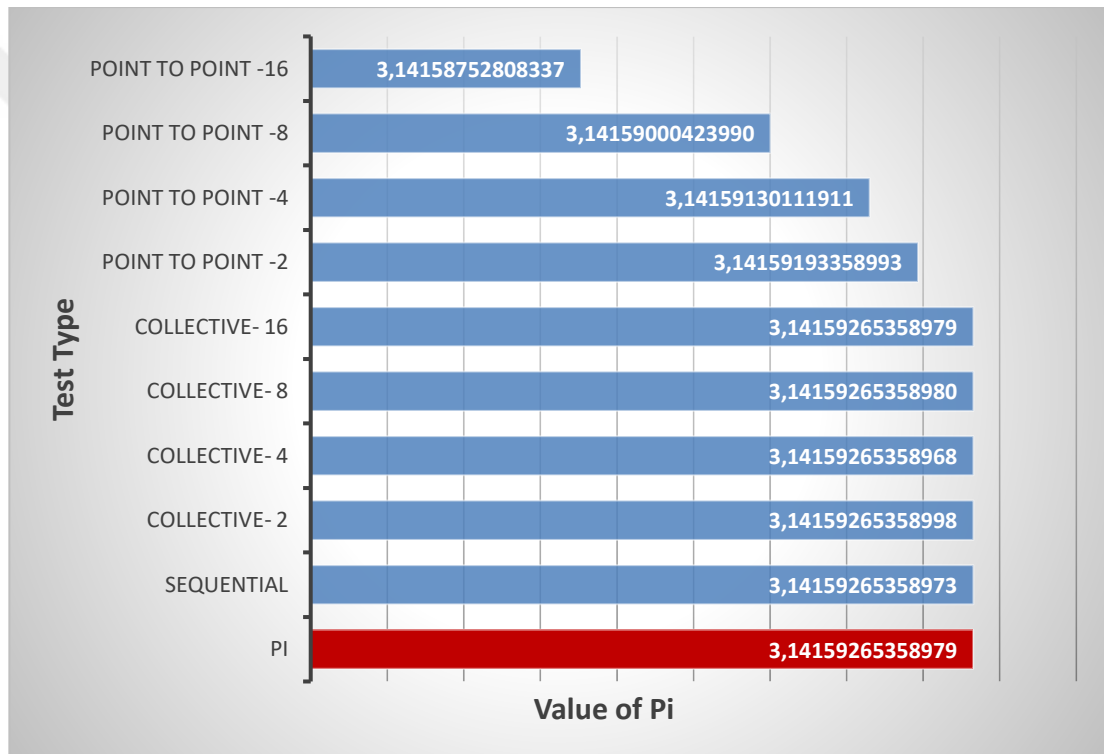


Figure 6.10 Comparison of pi results

The results evaluated with Amdahl's law and are shown on below. The parallel fraction of calculation of pi test program is almost 100% because of the number of iteration is very high. However, for evaluation performance of test programs, the fraction was assumed as 99.9 % and calculation of speed up was done with it.

Table 6.4 Amdahl's law comparison of pi calculation test programs

	2 SLAVES	4 SLAVES	8 SLAVES	16 SLAVES
AMDAHL' S LAW	1,98	3,88	7,48	13,9
COLLECTIVE	1,91	3,68	6,16	9,85
POINT TO POINT	1,39	2,73	4,89	8,38

The results show satisfied reduction in calculation time. The difference between collective and point to point test programs at processing time reduce stems from flow chart of processing.

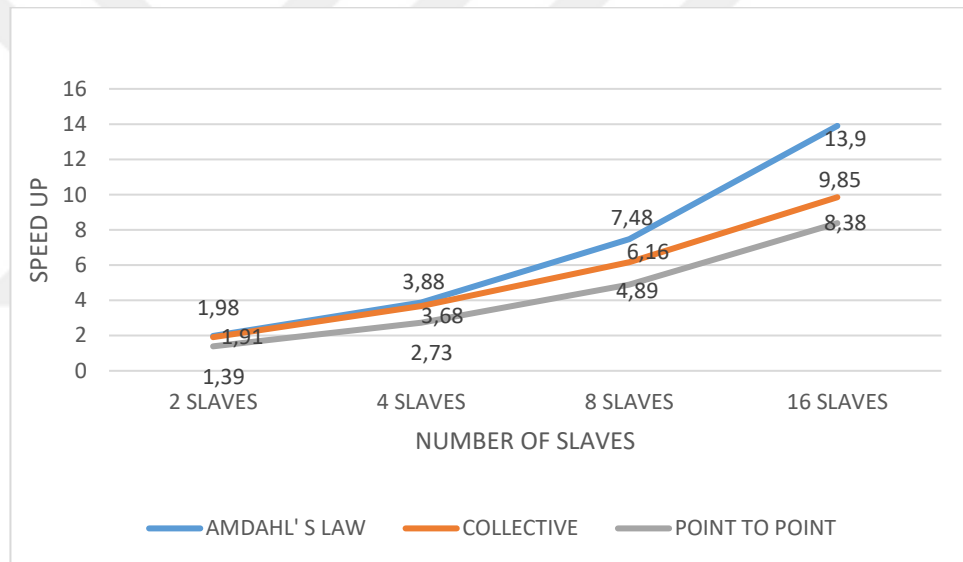


Figure 6.11 Comparison of speed up between Amdahl's law and Pi calculation test results

As mentioned before, point to point test program's algorithm based on sending and receiving data are working sequentially. The master uses comm.send to send data to slave nodes. This code first sends data to the receiver node's buffer storage and wait for the recipient acknowledgement to send next one. This blocking communication style caused wasting time. Beside of this comm.bcast is send all data at the same time to slave nodes and doesn't demand any information from them. Because of these differences between logic of communication style, we can say that collective program's success is expected result.

If we compare the error results, we can say that collective test program is working better than point to point program. At collective tests added slave nodes boosted up the results' success gradually. But at the point to point program we see the diverse difference, the added slave nodes are caused decrease on reaching a healthy result. The increase of the divided parts rises up the probability of loss of packets used in the calculation.

If we evaluate carefully without comparing between collective and point to point programs, we see the scalable reduction tendency depends on the number of slaves from each of their graphics (Figure 6.3 & Figure 6.5). These graphics are very similar to Gustafson's graphic (Gustafson, 1988) which characteristic of speed up under Amdahl's law that is given at chapter 2 (Figure 2.15). From these references, we can say that the result of speed up for two of test programs meet expectations.

6.3 Sobel Filter Test Program Results

The process was carried out in 5 different platform size: sequential and parallel platform with 2-4-8-16 slaves. The test programs run 50 times for every size of platform and standard deviation of execution times were calculated and shown at Table 6. . As the test image, baboon face which is commonly used for image processing testing was preferred. Result image is given on below at Figure 6.11.

Table 6.5 Standard deviation values of Sobel filter test programs' results

	1 SBC	2 SLAVES	4 SLAVES	8 SLAVES	16 SLAVES
Standard Deviation Value	0,102	0,828	3,467	1,87	0,454

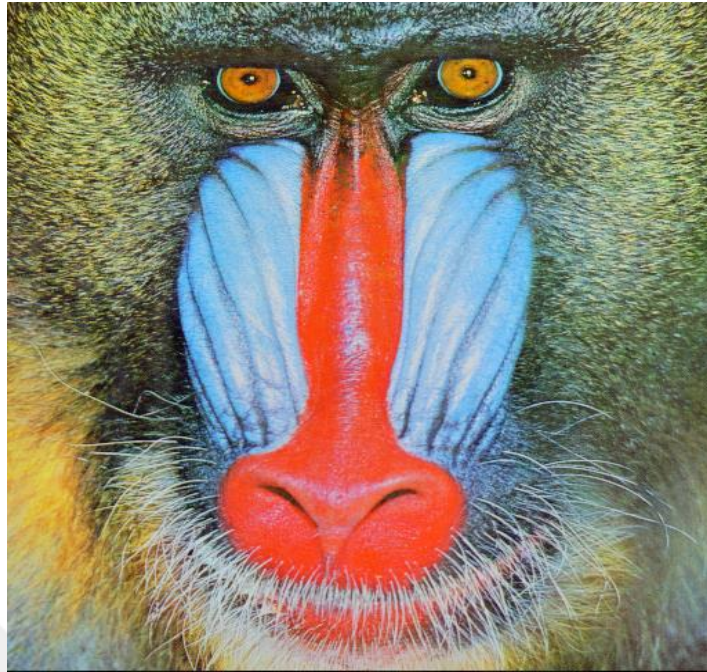


Figure 6.12 Baboon face – unprocessed Sobel filter test image



Figure 6.13 Baboon face - Sobel filtered test image

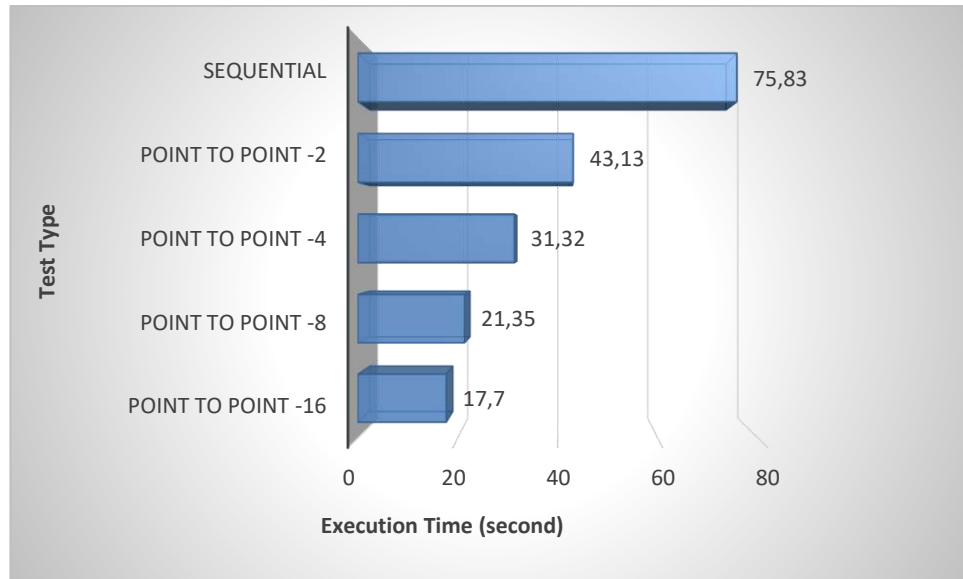


Figure 6.14 Execution time reduction graphic of sobel test programs

In this test program, because of algorithm structure the parallel fraction is lower than previous test programs. The converting processes and creating new image from new pixels processes were conducted by master and these parts run as serial. The total execution time is 75,83 second and parallel execution time nearly 65,81 second. Therefore, parallel fraction is evaluated as nearly 87% and serial fraction is nearly 13%. These fractions detected by using total time and divided parts' processing time comparison. As mentioned before, the formula of relationship between total and parallel execution time could be written as on below;

$$\text{Fraction. parallel} = (\text{Time. parallepart}) * 100 / (\text{Time. main}) \quad (6.2)$$

Table 6. 6 Amdahl's law comparison of Sobel filter test programs

	2 SLAVES	4 SLAVES	8 SLAVES	16 SLAVES
AMDAHL'S LAW	1,77	2,88	4,19	5,42
TEST RESULTS	1,76	2,42	3,55	4,28

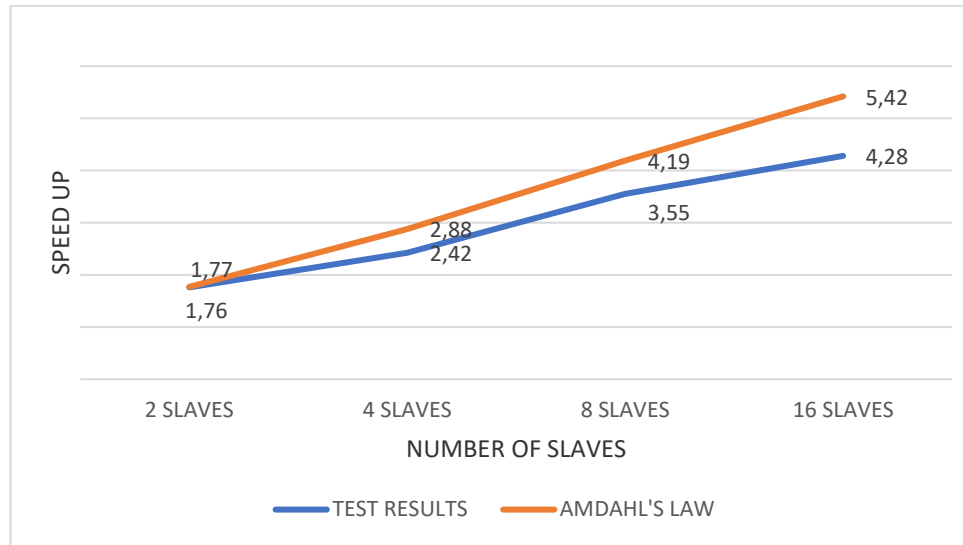


Figure 6.15 Comparison of speed up between Amdahl's law and Sobel filter test results

As seen at Table 5.7 and Table 5.8, there are differences in software codes sequential and parallel programs, these cause a delay of 0.03 seconds, but this value is so negligible when considering these programs. Therefore, these nonsense differences were not considered.

With the number of processors added, an increase in processing speed was observed each time. It was observed that the speed increase during the process gradually decreased as the image size was small. This is an expected result for parallel processing. In the work called as "Parallel approach of Sobel Edge Detector on Multicore Platform", it is stated that as the image size increases, the performance ratio increases (N.E.A.Khalid, 2011). When this reference and Amdahl's law are taken into consideration, an appropriate relationship is found between the results. The results are consistent with the Amdahl curve. On the other hand, the differences between Amdahl's law calculation and test results are directly proportional to the number of processors added to the platform. Actual speed up is less than Amdahl's law calculation because it is expected that performance in real systems is slightly slower due to non-ideal system conditions (Amdahl, 1967).

CHAPTER SEVEN

CONCLUSION

Consequently, we can see the expected result of the thesis which is a considerable reduction at runtime for every node that is added to parallel computing.

Changes in runtime are nearly proportional to the number of nodes. That shows measurable parallelism. In this context, the cluster can be considered as a scalable platform. As seen, for each added node has contributed almost a proportionally speed up. This feature gives a clue about the relationship between the size and the performance of the cluster.

These results show that extremely cheap single board computer as Super Pi can be used as a processor in cluster and it can be give a successful performance results.

This low costed platform can be used like a high-performance computer in many needed fields such as image processing, proving massive mathematical theorems, predicting weather conditions etc. which have many iterative processes and divided parts to solve. And we also benefit from them for experimental parallel computing platform at schools, HPC for the project which has shoestring budget or hobby material for who interests. In any event, we foresee that this kind of platforms will become much more popular into the near future.

REFERENCES

- Akçay, M., & Erdem, H. A. (2010). Paralel Hesaplama ve MATLAB Uygulamaları. *Akademik Bilişim*. Muğla: Muğla Üniversitesi.
- Akhter, S. &. (2006). *Multi-core programming 33*. Hillsboro: Intel press
- Amdahl, G. M. (1967). Validity of the single processor approach to achieving large scale computing capabilities. *AFIPS' 67 - spring joint computer conference* (pp. 483-485). ACM.
- Arduino . (n.d.). *About Us*. Retrieved July 28, 2017, from www.arduino.cc: <https://www.arduino.cc/en/Main/AboutUs>
- Barney, B. (2016). *Message Passing Interface (MPI)*. Retrieved May 15, 2017, from computing.llnl.gov: <https://computing.llnl.gov/tutorials/mpi/#What>
- Barney, B. (2017). *Introduction to Parallel Computing*. Retrieved June 07, 2017, from https://computing.llnl.gov/tutorials/parallel_comp/
- BeagleBone*. (n.d.). Retrieved July 28, 2017, from beagleboard.org: <http://beagleboard.org/bone-original>
- Becker, D. J. (1999). *How to Build a Beowulf: A Guide to the Implementation and Application of PC Clusters*. MIT Press.

Brooks, D. B. (n.d.). *LoBoS History*. Retrieved July 28, 2017, from Lobos:
<https://www.lobos.nih.gov/LoBoS-history.shtml>

Brown, R. G. (2004). *Engineering A Beowulf-style Compute Cluster*. Duke University Physics Department.

Build "dyna-micro" an 8080 microcomputer . (n.d.). Retrieved July 28, 2017, from [www.classiccmp.org:http://www.classiccmp.org/cini/pdf/re/Dyna-micro%200576.pdf](http://www.classiccmp.org/cini/pdf/re/Dyna-micro%200576.pdf)

Chapter 4: Distributed and Parallel Computing. (2017). Retrieved April 25, 2017, from : <http://wla.berkeley.edu/~cs61a/fall1/lectures/communication.html>

Cox, S. J., Cox, J. T., Boardman, R. P., Johnston, S. J., Scott, M., & O'Brien, N. S. (2014). Iridis-pi: a low-cost, compact demonstration cluster. *Cluster Computing*. *Cluster Computing*, 17(2), 349-358.

CAT5E Ethernet. (n.d). *14 Feet Ethernet Dump Bin - 70 Cables* Retrieved May 05, 2017, from [buyprocable.com: http://buyprocable.com/CAT5E-Ethernet.aspx](http://buyprocable.com/CAT5E-Ethernet.aspx)

Dalcin, L. (2015). *MPI for python. user manuel release 2.0.0*.

Dalcin, L. (2017). *The MPI for Python*. Retrieved June 05, 2017, from [mpi4py.readthedocs.io: https://mpi4py.readthedocs.io/en/stable/](https://mpi4py.readthedocs.io/en/stable/)

David E. Culler, J. P. (1999). *Parallel Computer Architecture - A Hardware/Software Approach*. Morgan Kaufmann Publishers.

Diamond Systems Corp. (n.d). Retrieved June 20, 2017, from [whitepaper.opsy.st:
http://whitepaper.opsy.st/WhitePaper.diamondsys-combased-sbcs-wpfinal-.pdf](http://whitepaper.opsy.st/WhitePaper.diamondsys-combased-sbcs-wpfinal-.pdf)

Hawick, K. A., Grove, D. A., & Vaughan, F. A. (1999). *Beowulf - a New Hope for Parallel Computing, DHPC Technical Report DHPC-061*. University of Adelaide

Hewlett Packard Enterprise. (2016) *HPE OfficeConnect 1920 Switch Series*
Retrieved April 05, 2017, from [www.hpe.com:
https://www.hpe.com/h20195/V2/Getdocument.aspx?docname=4AA5-4095ENW](http://www.hpe.com:https://www.hpe.com/h20195/V2/Getdocument.aspx?docname=4AA5-4095ENW)

HPE. (n.d). *HPE OfficeConnect 1920 24G Switch*. Retrieved July 10, 2017, from [www.hpe.com:
https://www.hpe.com/uk/en/product-catalog/networking/networking-switches/pip.hpe-officeconnect-1920-24g-switch.6783404.html](https://www.hpe.com:https://www.hpe.com/uk/en/product-catalog/networking/networking-switches/pip.hpe-officeconnect-1920-24g-switch.6783404.html)

File:Foxconn SuperPI top.JPG. (2016). Retrieved April 15, 2017, from http://linux-sunxi.org: http://linux-sunxi.org/File:Foxconn_SuperPI_top.JPG

Flynn, M. J. (1966). Very high-speed computing systems. *IEEE*, 54(12), 1901-1909.

Foxconn Super Pi. (2016). Retrieved May 28, 2017, from [linux-sunxi.org:
http://linux-sunxi.org/Foxconn_Super_Pi](http://linux-sunxi.org: http://linux-sunxi.org/Foxconn_Super_Pi)

Global Market Insights, Inc. (2017). *Single Board Computer Market worth more than \$1.2bn by 2024*. Retrieved from [www.gminsights.com:
https://www.gminsights.com/pressrelease/single-board-computer-sbc-market](http://www.gminsights.com:https://www.gminsights.com/pressrelease/single-board-computer-sbc-market)

Gustafson, J. L. (1988). Reevaluating Amdahl's law. *Communications of the ACM*, 31(5), 532-533 .

Hawick, K. A., Grove, D. A., & Vaughan, F. A. (1999). *Beowulf - a New Hope for Parallel Computing*, *DHPC Technical Report DHPC-061*. University of Adelaide.

IBM. (1996). *Technical presentation*. Retrieved July 05, 2017, from ibm.com: <http://www.redbooks.ibm.com/redbooks/pdfs/sg244868.pdf>

intel co. (n.d.). *Intel Timeline: A History of Innovation*. Retrieved July 05, 2017, from www.intel.co.uk:https://www.intel.co.uk/content/www/uk/en/history/historic-timeline.html

Kiepert, J. (2013). Creating a raspberry pi-based beowulf cluster. *Boise State University*, 1-17.

Kingston Technology. (n.d). *microSDHC/microSDXC Class 10 UHS-I Kart*. Retrieved July 15, 2015, from www.kingston.com:https://www.kingston.com/tr/flash/microsd_cards/sdc10g2

LeMaker Super Pi . (2017, July). Retrieved from çizgi market : <https://market.cizgi.com.tr/product/education/lmk-Super%20Pi>

linux-sunxi. (2016). *Foxconn_SuperPI_top.JPG*. Retrieved December 05, 2016, from linux-sunxi.org: http://linux-sunxi.org/File:Foxconn_SuperPI_top.JPG

Module for Monte Carlo Pi. (n.d). Retrieved July 05, 2017, from mathfaculty.fullerton.edu:
http://mathfaculty.fullerton.edu/mathews/n2003/montecarlopi/MonteCarloPiMod/Links/MonteCarloPiMod_ink_4.html

N.E.A.Khalid, S. N. (2011). Parallel approach of Sobel Edge Detector on Multicore Platform. *INTERNATIONAL JOURNAL OF COMPUTERS AND COMMUNICATIONS*, 236-244.

Ortmeyer, C. (2014). Then and now: a brief history of single board computers. *Electron. Des. Uncovered*, 6, 1-11.

Pacheco, P. S. (2011). *An introduction to parallel programming*. Elsevier.

Python software foundation. (n.d.). *math — Mathematical functions*. Retrieved June 05, 2017, from docs.python.org: <https://docs.python.org/2/library/math.html>

Raspberry Pi 3. (2017, July). Retrieved from n11.com:
<http://urun.n11.com/diger/raspberry-pi-3-P172444709>

Rosch, W. (1999). *Hardware Bible 5th Edition*. Indianapolis: Que.

Snell, A. (2014). *Beyond Beowulf: Clusters, Cores, and a New Era of TOP500*. Retrieved May 29, 2017, from www.top500.org:
<https://www.top500.org/news/beyond-beowulf-clusters-cores-and-a-new-era-of-top500/>

Specification. (n.d). Retrieved April 05, 2017, from [www.cizgi.com.tr:](http://www.cizgi.com.tr:www.cizgi.com.tr/upload/pms_dosya/373/21526_44905.pdf)
http://www.cizgi.com.tr/upload/pms_dosya/373/21526_44905.pdf

Sterling, T. (2001). *Beowulf Cluster Computing with Linux*. MIT Press.

Top500 List - June 2017. Retrieved July 25, 2015, from [www.top500.org:](http://www.top500.org:www.top500.org/lists/2017/06/)
<https://www.top500.org/lists/2017/06/>

Upton, E. (2011). *Raspberry Pi – 2006 Edition*. Retrieved April 05, 2017, from [www.raspberrypi.org:](http://www.raspberrypi.org:www.raspberrypi.org/blog/raspberry-pi-2006-edition/) <https://www.raspberrypi.org/blog/raspberry-pi-2006-edition/>

Zinner, S. (2012). *High Performance Computing Using Beowulf Clusters*, Retrieved June 20, 2017 from [www2.hawaii.edu:](http://www2.hawaii.edu:www2.hawaii.edu/~zinner/101/students/MitchelBeowulf/cluster.html)
<http://www2.hawaii.edu/~zinner/101/students/MitchelBeowulf/cluster.html>

Wilson, G. V. (1994). *The History of the Development of Parallel Computing*. Retrieved July 25, 2017, from [http://ei.cs.vt.edu:](http://ei.cs.vt.edu:http://ei.cs.vt.edu/~history/Parallel.html)
<http://ei.cs.vt.edu/~history/Parallel.html>