AL-Bahir
Journal for Engineering and Pure Sciences
BJEPS

Volume 4 | Issue 2 Article 1

Parallel algorithm for testing the singularity of an N-th order matrix

Ehab Alasadi
Kerbala University: University of Kerbala madhatiyah, Babil IRAQ

Source of Funding
No external funding

Conflict of Interest
No conflict of interest

Data Availability
public available data

Author Contributions
The author solely contributed to all aspects of this work, including conceptualization, methodology, data curation, software, formal analysis, writing – original draft preparation, review and editing, and project administration.

ORIGINAL STUDY

Parallel Algorithm for Testing the Singularity of an N-th Order Matrix

Ehab Abdulrazak Alasadi

Dept. Islamic Science, Kerbala University, Kerbala, 65001, Iraq

Abstract

Analyze the possibilities of implementing a parallel algorithm to test the singularity of the N-th order matrix. Design and implement in (C/C++) a solution based on sending messages between nodes using the PVM system library. Distribute the load among the nodes such that the computation time is as small as possible. Find out how the execution time and calculation acceleration depend on the number of nodes and the size of the problem (indicate the table and graphs). Based on the results, estimate the communication latency, for what size the task is (well) scalable on the given architecture, and what is the maximum size when the calculation is still bearable on the available architecture by divide the program into master -slave model which is able to send tasks and collect the results to receive it by master computer.

Keywords: Pthread, PVM, Parallel program, Master\slave model

1. Introduction

PVM to have big tasks and you could to dive it to small tasks and collect the results to the master Computer and display the result on the Screen the aim of the research is how to reduce the exception time when we have big tasks so the effective methods to use the PVM its makes it possible to develop applications on a set of heterogeneous computers connected by network that appear logically to the users as single parallel computer, the PVM offer the powerful set of process control and dynamic recourses Management functions, its provides programmers with the library of routines for initiation and termination of tasks synchronizations and the alteration of virtual machine configuration its also facilities messages passing vie number of simple constructs, PVM application is made from the number of tasks that cooperate to jointly provide a solution to single problem a task may alternate between computation and communication sequential tasks in which each task has its own locus of control and Sequential tasks communication by exchange tasks. and what can the expected solution can be affect ! network speed, processing speed, Algorithm selection and number of nodes [1].

2. Program parameters

The research consists of five files:

master.c – the main part of the assignment. Contains code running on the master node.

slave.c – contains code for other child nodes.

matrix.txt – input file, contains the matrix entered in the format specified in the entry (the line of the matrix corresponds to the line of the file and the column values are separated by spaces or tabs).

Readme - Describes how to compile master.c and slave.c files.

Dps.doc – documentation for implementation and testing.

3. Program datasets

The matrix values entry using a text file contains the matrix specified columns and rows (separated by space or tabs) and uses **"fopen"** to real elements from input text file and the result will be saved an output text file:

//Master.c code

```
#include <stdio.h>
#include <pvm3.h>
#include <stdlib.h>
#include <math.h>
```

Received 15 October 2023; revised 19 November 2023; accepted 4 December 2023.
Available online 11 January 2024
E-mail address: ehab.a@uokerbala.edu.iq.

https://doi.org/10.55810/2313-0083.1055

```
#include <string.h>
#include <malloc.h>
#include <time.h>
clock_t spust;
clock_t zastav;
clock_t cas = 0;
double dlzka,priemer;
long nacitajn(){
  long n = 0;
  FILE *in;
  char cc,c;
  if((in = fopen("matrix.txt","rt")) == NULL)
    {
    printf("miss open file.\n");
    return 1;
    }
  while ((cc = getc(in)) ! = EOF){
    if (cc == '\n') n++;
    }
    fclose(in);
  return n;
```

4. Analysis

4.1. Basic terms

A distributed system is an application comprising of several components running simultaneously on different computers. These computers must be able to communicate with each other and be able to work independently.

Parallel computing is the simultaneous computation of one task on multiple processors in order to speed up the calculation. The processor can be either a CPU in a computer or a single node (computer) in a distributed system. Distributed systems are used for parallel computing in various branches of science. In these systems, the task is divided into several subtasks, which are independent of each other, and these are then calculated simultaneously on individual computers, thereby shortening the calculation time. Such systems are either homogeneous or heterogeneous [11].

4.2. PVM

PVM **(Parallel Virtual Machine)** is a software package that allows the creation of heterogeneous computer clusters by connecting computers with UNIX or Windows system, interconnected by a network. This system is freely available, highly portable and easy to use, which has contributed to its widespread scientific use.

A PVM is a so-called message passing system, i.e. system in which parallel tasks synchronize and exchange information by sending messages.

The PVM system itself consists of a daemon that runs on each cluster node and a library that is compiled into the user program. A program using library services can be written in various programming languages, the most commonly used of which are C, C++ and FORTRAN. This library offers functions for creating new tasks and well as for communicating between them. The PVM daemon then handles the physical execution of these tasks and the transfer of messages [1].

4.3. Program architecture

Program divided tasks into master-slave will be created. Master's task will be to Process input data (number of nodes on which the computation will be executed, dividing data for processing in slave nodes, process the data from slave nodes and evaluate results (storing the results in files). The slave's task will be to decide whether a matrix (sent by master) is/is not a singular And to send the result back to master, contains functions and code executed by the master node. The parameter of this program is the number of nodes. However, it is not mandatory. If the number of nodes is not entered, the default number of nodes = 2 is selected. The next step is to load the matrix using the main program (mastro). Based on the entered number of nodes, the program calculates how many columns the slave will count and immediately sends the assigned columns to it. This is repeated for each slave. In order for the slave to start counting, it is necessary to send it also pivots. Pivots are calculated by the master and sent to all slaves who are still participating in the calculation (see Fig. 1).

1- Request
2 - Process
3 -Reply

///**Slave .c. code,** master wait for the result from slave

```
//int main (int argc, char ** argv) (see Fig. 2)
{
long i,j,k,n,sh,hh;
int my_tid = pvm_mytid();
int master_tid = pvm_parent();
pvm_recv(-1, −1);
pvm_upklong(& n,1,1);
pvm_upklong(& sh,1,1);
pvm_upklong(& hh,1,1);
double  *matrix=(double*)malloc(sizeof(double)*
(hh-sh)*n);
```

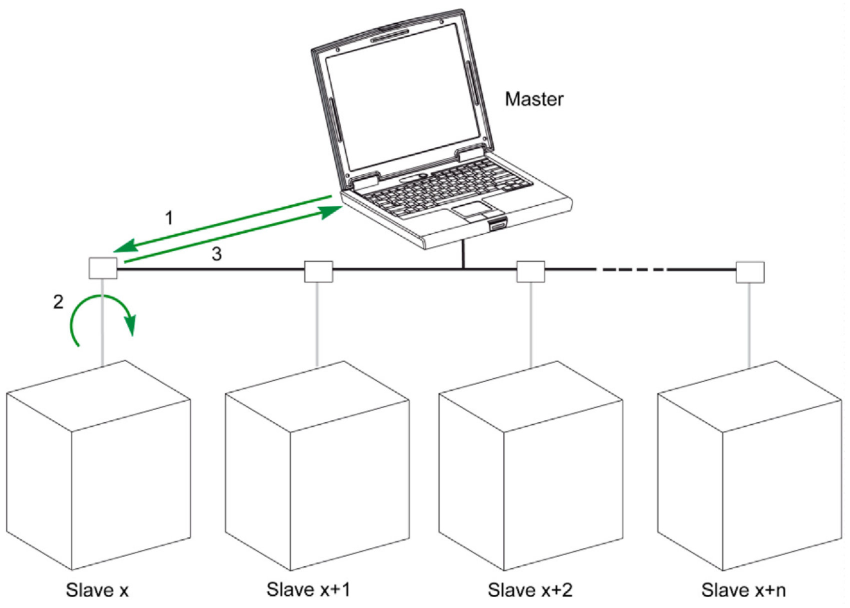

Fig. 1. Master\slave model [6].

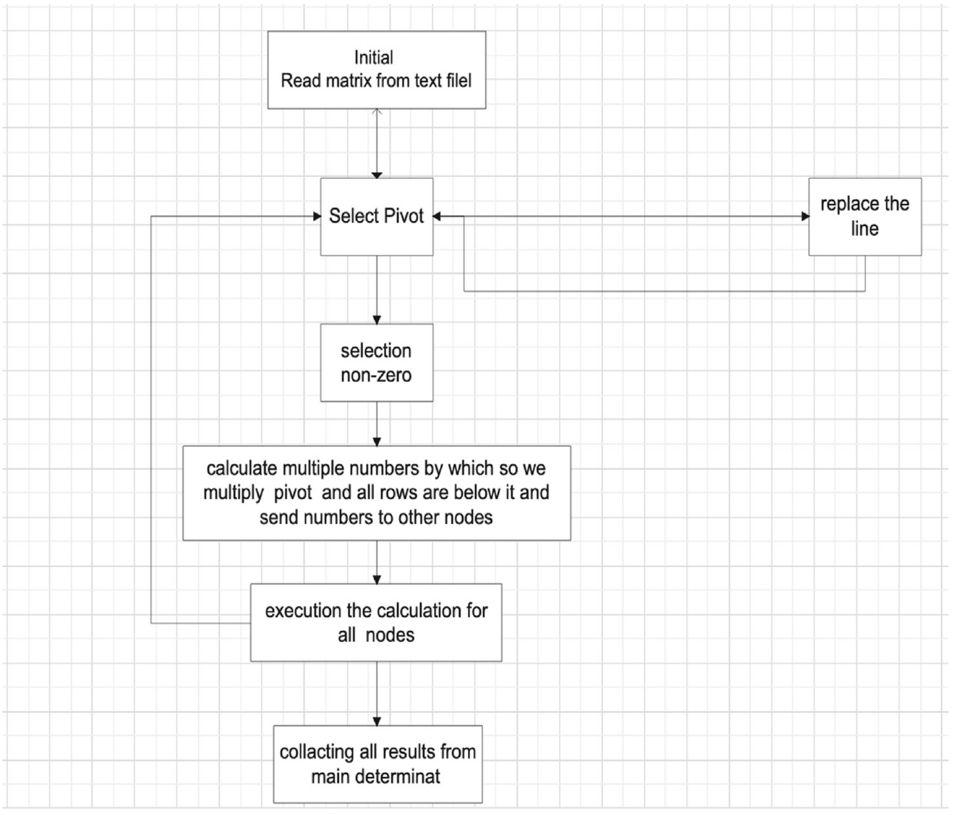
Initial
Read matrix from text filel
Select Pivot
replace the line
selection non-zero
calculate multiple numbers by which so we multiply pivot and all rows are below it and send numbers to other nodes
execution the calculation for all nodes
collacting all results from main determinat

Fig. 2. Design flowchart.

```
for(j = 0; j < hh-sh; j++) pvm_upkdouble
(matica + j*n,n,1);
double *pivot=(double *)malloc(n*sizeof(double));
for(;;)
{
  pvm_recv(-1, −1);
  pvm_upklong(& k,1,1);
  pvm_upkdouble(pivot,n,1);
```

```
    for (j = 0; j < hh-sh; j++)
      for (i = k + 1; i < n; i++)
        matrix[j*n + i] = matrix[j*n + i] - pivot[i] *
matrix[j*n + k];
    if (k ≥ sh-1 && k < hh-1)
    {
      pvm_initsend(PvmDataDefault);
      pvm_pkdouble(matrix+(k+1-sh)*n,n,1);
      pvm_send(master_tid, 0);
    }

  if(k==hh-1) pvm_exit();

  }
  pvm_exit();
  return 0;
  }
```

4.4. Known procedures for solving this problem

The determinant can be calculated in several ways:
i- line number
j- column number

$$|\mathrm{A}| = \sum_{i=1}^{k} \alpha_{ij} C_{ij}, \quad (1)$$

M− matrix, which is created by overlapping column j and row i

$$C_{ij} \equiv (-1)^{i+j} M_{ij} \quad (2)$$

However, this algorithm is not suitable for parallel processing.
Method by adjusting to the upper triangular shape using the Gaussian elimination method and multiplying the elements on the main diagonal:
We obtain this shape by applying:

- by multiplying the row by a non-zero constant c
- by exchanging two lines
- by adding c multiple of the row to the second row

5. Proposal

The assignment gives me the task of determining whether the entered matrix is singular. Because a square matrix is singular if and only if its determinant is equal to zero. It follows that it is necessary to calculate the determinant. There are several methods for calculating the determinant. I will use the Gaussian elimination method to solve this problem. The advantage of this method is the possibility to parallelizing the task.
Input: Square matrix (read from file).
Output: determinant of the matrix and comparison of the determinant with zero [9].

5.1. Algorithm

One main initialization task will be chosen, creating as many subtasks as needed and sending them a range of columns to work on. The column range is assigned to the TID. Each task reads the entire matrix from the input file. During the Gaussian elimination method, the subtask holding the current pivot calculates and sends the corresponding number of multipliers to tasks with higher TIDs. Then each task performs the necessary calculations and sends its partial determinants to the initialization task, which calculates the final size of the determinant.
Algorithm start reading matrix from input text file and program divided into Master-Slave parts [2].
The Master −Slave Model (PVM)

- PVM is not restricted to this model.
- Useful programming paradigm and simple to illustrate
- The master calls pvm_mytid() to
1. Allow it to use the PVM system, and
2. Enable interprocessor communications
- It then call pvm_spawn() to execute a given number of slave programs on other machines
- Each slave program must also call pvm_mytid() to enable processor communications
- Subsequently, pvm_send() and pvm_recv() are used to pass messages between processes
- When finished, all the PVM programs should call pvm_ exit() to finish.

5.2. Synchronization

waits for tasks on nodes that are still computing to be completed before each new pivot selection

5.3. Program environment

The C++ environment under the UNIX OS will be used to develop algorithms for calculating the singularity of the matrix.
Functions:
pvm_mytid find the id of the current job
pvm_parent find parent id
pvm_spawn starts the pvm application
pvm_upkstr extracts the message from the buffer
pvm_initsend initialize the broadcast buffer
pvm_pkstr places the string in the broadcast buffer

pvm_send send message to target process,
pvm_send send a message to the target process specified by its tid
pvm_exit deletes the program from pvm, [10,12].

5.4. Data collection

Using functions
getrusage() - returns the time that the process has consumed (either by executing its own program or in the kernel). Of course, it does not include the time when the process was not running at all (eg when it was waiting) (see Tables 1 and 2).
clock_t times(struct tms *buf) does the same as the previous one, but also returns the number of processor ticks since the system was started premises.

6. Testing data

Several matrices were used for testing. The basic matrix was 5 × 5, the result of which was known to me and therefore it was used for functionality testing.
Furthermore, the 50 × 50 matrix was used,. It was launched every time 20 times but always with a different number of connected nodes. The result was then averaged.
The measurement was carried out using the function clock_t times(struct tms *buf).

- Manual

Using the assignment is very simple. Only one parameter was implemented, namely the number of nodes. The input file must be called matrix.txt and must be located in the same directory as the compiled master [3].

6.1. Speedup and time calculations

The purpose of this study was to use application software and hardware systems (see Fig. 5).

$$\text{Speedup} = S_P = \frac{T_S(n)}{T_P(n)}$$

Efficiency of the parallel algorithm:

$$\text{System efficiency} = E_P = \frac{T_1(n)}{m.T(n)}$$

T1- Duration of a parallel algorithm running on a **single** CPU.
Multiprocessors and minicomputers can be used to solve enormous issues that require considerable memory and time to complete, [4,8].
Matrix dimensions: 50 tested
On time measurement was she employed function *clock_t times(struct tms *buf* by which is possible to Obtain three times: real, user and system.
I managed to test the mentioned matrix with a dimension of 50 × 50 and a limited number of nodes (specifically 1,4,10,15). For one node, the results are very favorable because if the program detects that it will be calculated on only one node, a separate function is performed that does not use pvm and sending messages, which is a great advantage with

Table 1. Time calculation when matrix dimension = 50.

Number of nodes	Number of cycles	Real time (S)	Average time for 1 cycle (S)
1	20	0.03	0,0015
4	20	0.21	0,0105
10	20	0.12	0,006
15	20	0.25	0,0125

Table 2. Time calculation when Matrix dimensions: 50.

Number of nodes	Number of cycles	Real time (S)	Average time for 1 cycle (S)
20	1	0.03	0,0015
20	2	0.98	0,049
20	2	1.05	0,0525
10	2	0.52	0,052
5	2	0.24	0,048
10	3	0.59	0,059
10	3	0.58	0,058
5	4	0.61	0,061
5	5	0.3	0,06
5	6	0.28	0,056
5	8	0.31	0,062
5	10	0.33	0,066

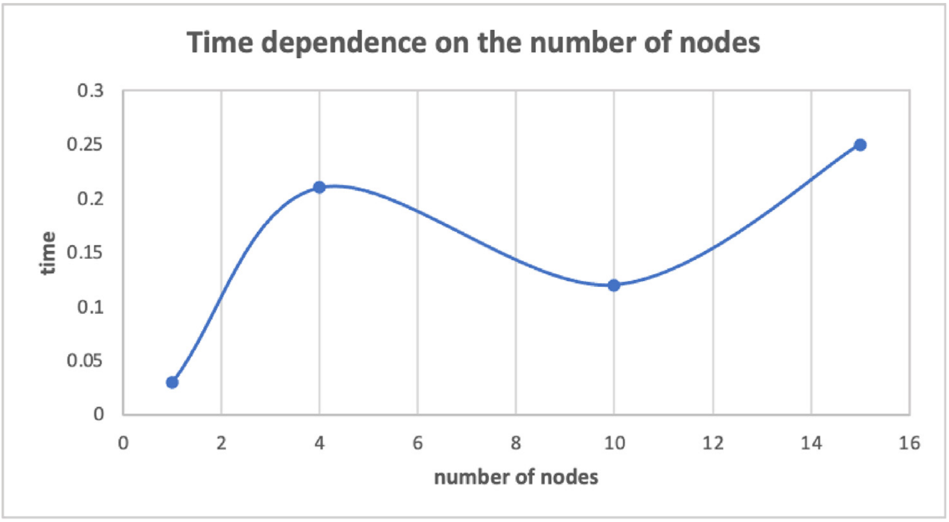

Fig. 3. Time dependance on the number of nodes.

such a small matrix. Very good results were obtained even when using 10 nodes [5,7].

Since I was not satisfied with the amount of data collected, I decided to build my own network of connected computers. A network consisting of five nodes was successfully assembled. The shortcoming of this network was that it was also used by other users during testing, therefore the results are somewhat distorted (see Fig. 4).

Since this case it was a used network, the best results were detected for matrices that used the least number of nodes. As the number of nodes increased, so did the number of sent messages, which extended the length of the calculation.

6.2. Evaluation of the results

Network of five nodes was created. This was used during the calculation and therefore the results are distorted. Especially when using multiple nodes that generate multiple messages.

Acceleration: this calculated from the formula:

Acceleration = calculation time on one node/ calculation time on n nodes.

The above figure was created from the previous formula. In both cases, it is rather a slowdown because the calculation time on one node is very low due to the already mentioned circumstances (see Fig. 3).

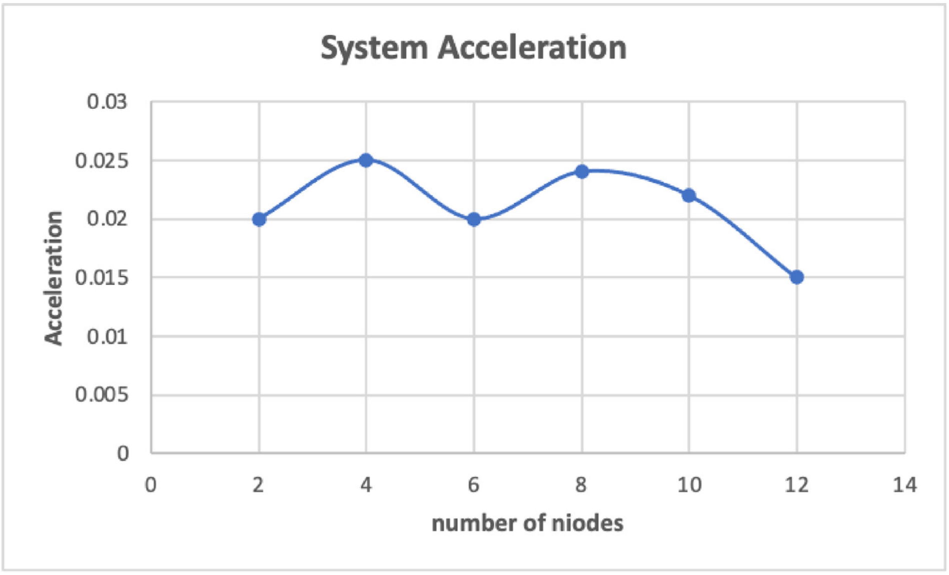

Fig. 4. System acceleration.

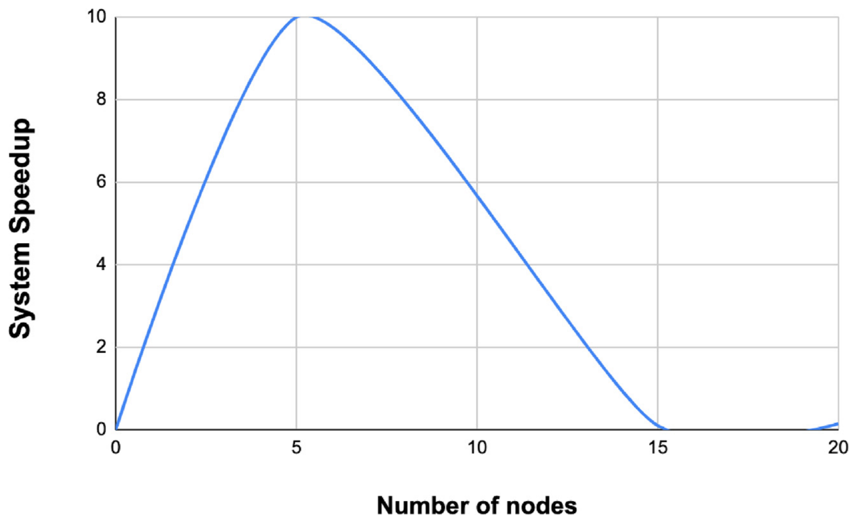

Fig. 5. System Speedup.

When Speedup $= S_P = \frac{T_S(n)}{T_P(n)}$

Ts = time at one process.
Tp:time at n process.

6.3. Other possible modifications, improvements, notes

The main improvements should definitely include the treatment of the program for working with larger matrices, where the computing power of distributed computers would be shown to a greater extent. Furthermore, the size of the message sent using the slave process could be reduced, which would reduce the demands on the network used.

6.4. Program limitations

According to the research, the program will be adapted to work with real numbers. Another limitation will be the amount of dedicated memory per array elements load and the maximum amount of allocated memory.

7. Conclusion

A PVM is a software system that allows a group of disparate computers to be used as a coherent and flexible computing resource. Individual computers can be connected by different types of networks such as Ethernet, and FDDI, etc. the PVM utility software runs on every computer in a user-specified computer group and creates a unified, generic, and powerful environment for programming distributed applications. These can be used in various fields of science and research.

Evaluation of the results, the program had to be tested. Therefore, the network of 5 nodes was created. This was used during the calculation and therefore the results are distorted. Especially when using multiple nodes that generate multiple messages. In these cases, it is rather a slowdown, since the calculation time on one node is very low due to the mentioned circumstances.

Conflict of interest

I hereby certify that there is not an actual conflict of interest or potential for personal gain from any of the organizations.

References

[1] Santos CMP, Aude JS. PM-PVM A portable multithreaded PVM. In: Proceedings 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing, IPPS/SPDP; 1999.
[2] Yang Tao. Lecture notes on parallel scientific computing department of computer science university of California at santa barbara. 1998.
[3] Geist Al, Adam Beguelin, Jack Dongarra, Jiang Weicheng, Manchek Robert, Sunderam Vaidy. PVM: Parallel virtual machine. The MIT Press; 1994.
[4] El-Rewini Hesham. Advance computer architecture and parallel processing. NJ: John Wiley & Sons; 2005.
[5] Jordan HF, Jordan HE. Fundamentals of paralle computing. Prentice Hall; 2002.
[6] Schneider electric, Master-Slave Principle, Digital image. https://product-help.schneider-electric.com/ED/ES_Power/NT-NW_Modbus_IEC_Guide/EDMS/DOCA0054EN/DOCA0054xx/Master_NS_Modbus_Protocol/Master_NS_Modbus_Protocol-2.htm#:~:text=The%20master%2Dslave%20principle%20is,at%20www.modbus.org. [Accessed 18 November 2023].
[7] Akl SG. The Design and analysis of parallel algorithms. Englewood Cliffs, NJ: Prentice Hall; 1989.
[8] Gropp W, et al. The sourcebook of parallel computing. Morgan Kaufmann; 2002.

[9] Joseph J, Fellenstein C. Grid computing. Prentice Hall; 2003.
[10] Craig J. MATH10212 linear algebra textbook, D. Poole, linear algebra: a modern introduction. Thompson; 2006. ISBN 0-534-40596-7.
[11] Maarten V, Andrew S. Distributed systems. third ed. CreateSpace Independent Publishing Platform; 2017.
[12] Gederberg T. Parallel Processing using Parallel Processing using PVM on a Linux Cluster. CENG 2007;6532.