University of Mumbai
2020-2021

M.H. SABOO SIDDIK COLLEGE OF ENGINEERING 8,
Saboo Siddik Road, Byculla, Mumbai - 400 008

Department of Computer Engineering

Project Report On

“Matrix Multiplication using MPI”

By

Ms. Ansari Asma Ayaz	3117002
Ms. Indorewala Fatema Mustafa	3117015
Mr. Tiwari Shivam	5117060

Guided By
Er. Mohammed Ahmed

ACKNOWLEDGEMENT

We would like to thank our Almighty God for granting us wisdom, under his guidance, never-ending grace, and mercy our project wouldn't have been accomplished. It is because of HIS blessing we have completed our project with great enthusiasm. We wish to express our sincere thanks to our director Dr. Mohiuddin Ahmed and our I/C principal Dr. Ganesh Kame, M.H. Saboo Siddik College of Engineering for providing us all the facilities, support, and wonderful environment to meet our project requirements.

We would also take the opportunity to express our humble gratitude to our Head of Department of Computer Engineering Dr. Zainab Pirani for supporting us in all aspects and for encouraging us with her valuable suggestions to make our project a success.

We are highly thankful to our internal project guide Er. Mohammed Ahmed Shaikh who gave us the golden opportunity to do this bodacious report on the topic " Matrix Multiplication using MPI", also helped us in doing a lot of research work, imbuing team spirit and enhancing our learning process, whose valuable guidance helped us understand the project better, her constant guidance and willingness to share her vast knowledge made us understand this project in great depths and helped us to complete the project successfully.
Deepest thanks and appreciation to our parents, family and others for their cooperation, encouragement, constructive suggestion and full of support for the report completion, from the beginning till the end. We would like to express our gratitude and appreciate the guidance given by other supervisors and project guides, their comments and tips helped us in improving our presentation skills.

Although there may be many who remain unacknowledged in this humble note of appreciation there are none who remain unappreciated.

Table of Contents

Table of Figures

Abstract

With the increasing requirements of high performance computing, parallel programming has become popular and important. One of the reasons is parallel programming gives an opportunity to use computer resources more efficiently. Thus, running time decreases and software quality increases. In this study, matrix multiplication is done by using a message passing interface (MPI) parallel programming approach and it is tested on various core machines with different sizes of matrices such as 500X500, 1000X1000 and 2000X2000. It is tested with the process number 1, 2, 4, and 8. Performance of matrix multiplication with MPI is studied. Relationship between computation time –matrix size– number of cores and processes is examined, experimental results are collected and analyzed .

Chapter 1

Introduction and Motivation

1.1 Introduction

Parallel programming means programming more than one computer or computers with more than one core to solve a problem in a short time with greater computational speed [1]. Here, the problem is divided into smaller parts and executed concurrently. After execution is completed, results from all parts are combined and the problem is solved. With the increasing requirements of high performance computing, parallel programming has become very popular. The reason for this is that it is easy to solve larger and complex problems in a short time thanks to the computing resources that do multiple things at a time concurrently.

[2]. In order to write efficient parallel programs, hardware knowledge is as important as software knowledge for the reason that using all cores and memory in an efficient way. Parallel programming models can be classified as POSIX (Portable Operating System Interface) Threads, Shared Memory OpenMP (Open Multi-Processing), Message Passing, CUDA (Compute Unified Device Architecture), OpenCL (Open Computing Language), DirectCompute and Array Building Blocks.

[3]. In this study, matrix multiplication is done by using the Message Passing Model on Visual Studio platform and coded with C++ programming language. It is tested on various core machines with different size of matrices such as 500X500, 1000X1000, 2000X2000, and with the process number 1, 2, 4, and 8 for the aim of observing relationship between computation time -matrix size- number of cores and processes. In the second part of this study, materials and methods are explained, information about Message Passing Model and Message Passing Interface is given and an algorithm which is used to compute matrix multiplication is explained. In the third part, computation time of matrix multiplication with different sizes of matrices such as 500X500, 1000X1000, 2000X2000, and with the process number 1, 2, 4, and 8 is given with tables and graphics. In the last part, the relationship between computation time–matrix size–number of cores and processes is explained. Finally, conclusions are given and future work is explained.

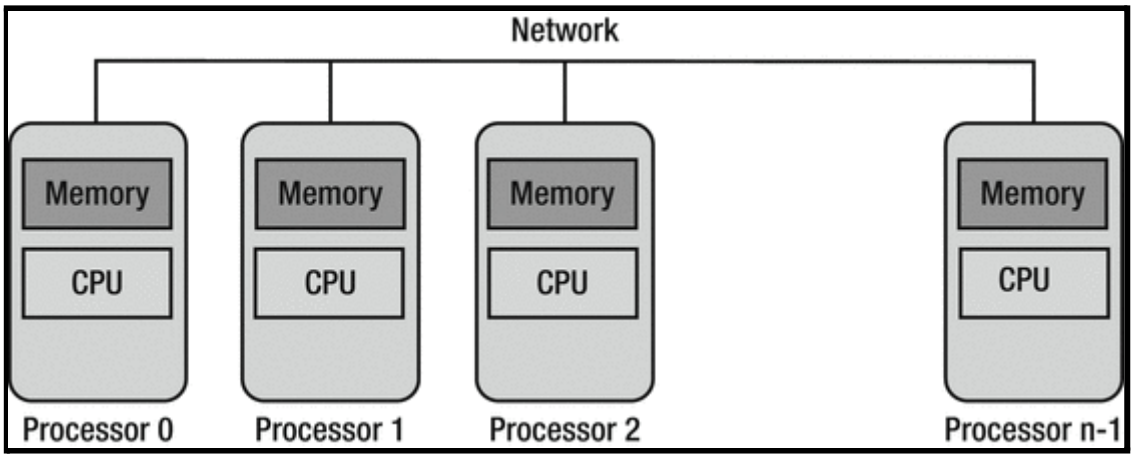

Fig 1: MPI

1.2 Motivation

When matrix size is larger, the number of cores and the number of processes start to be more important. The reason for this; for larger matrix size, when more processes are created and a machine which has more cores is used, calculation time decreases greatly. Creating more processes on single core, dual core or quad core machines for all sizes of matrices and using multi core machines decreased computation time.

1.3 Aim

The main aim is to reduce the time complexity and space complexity by normal matrix multiplication and yield faster and convenient execution for the same. To make Execution time reduce and to increase speedup. Get a simple computation for each processor.

Chapter 2

Problem Statement

In mathematics, particularly in linear algebra, matrix multiplication is a binary operation that produces a matrix from two matrices. For matrix multiplication, the number of columns in the first matrix must be equal to the number of rows in the second matrix. The resulting matrix, known as the matrix product, has the number of rows of the first and the number of columns of the second matrix. The product of matrices A and B is denoted as AB.

Matrix multiplication is thus a basic tool of linear algebra, and as such has numerous applications in many areas of mathematics, as well as in applied mathematics, statistics, physics, economics, and engineering. Computing matrix products is a central operation in all computational applications of linear algebra.

The goal of this problem is to perform matrix multiplication by using MPI so as to improve the efficiency of the code and get quicker results. Matrix Multiplication is an important concept that is used in various fields like Machine Learning, Image Processing. By using MPI we would reduce the cost of huge Matrix Multiplication for faster flow of execution and better performance.

Chapter 3

Analysis

In this study, matrix multiplication is done by using a message passing interface in Ubuntu . In the Message Passing Model, each process has its own address space and communication is performed by sending and receiving messages . Message Passing Interface (MPI) is a language-independent communications protocol and message passing system that is used with the aim of high performance, scalability, and portability. Implementation of Message Passing Model is performed by defining a library of routines such as point to point communication functions and collective operations . Message passing is realized by sending and receiving messages. To send data to the whole group of processes it is broadcasted .

MPI_Send(void* data, int count, MPI_Datatype datatype,int destination, int tag, MPI_Comm communicator) MPI communication routine which is shown above is used to send data in size of “count”, in data type of “datatype“ to the process whose id is “destination” with the message tag “tag” to the communicator “communicator“.

MPI_Recv(void* data, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm communicator, MPI_Status* status) MPI communication routine which is shown above is used to receive data in expected size of “count“, in data type of “datatype“, from the process whose process id is “source“ with the message tag “tag” to the communicator “communicator“, in status “status“.

int MPI_Bcast(void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm) MPI communication routine which is shown above is used to send the same data to all processes in the communicator . According to the above equation, data in size of “count“, in data type of “datatype“ is sent to all processes in communicator “comm“ from “root“. Other processes in communicator must call MPI_Bcast routine to receive data because there is no MPI call to receive MPI_Bcast routine. Most commonly used MPI routines are MPI_Init, MPI_Comm_size, MPI_Comm_rank, MPI_Abort, MPI_Get_processor_name,

MPI_Get_version, MPI_Initialized, MPI_Wtime, MPI_Wtick and MPI_Finalize. MPI_Init is used to initialize an MPI execution environment, so it must be called by every MPI program only once before all MPI functions. MPI_Comm_size is used to learn the total number of processes for a specified communicator. MPI_Comm_rank routine returns rank number of MPI process. MPI_Abort is used to terminate all MPI processes for specified communicators whereas MPI_Finalize is used to terminate MPI environments. To learn processor name MPI_Get_processor_name; to learn MPI version and subversion MPI_Get_version; to learn elapsed wall clock time in seconds for specified processor MPI_Wtime routines functions are used .

Theory and calculations :

Matrix multiplication is done by using asynchronous and synchronous message passing. Firstly A and B matrices are created randomly on mode 10 and on size N which can be 500, 1000 and 2000.

```
for(i=0;i<N;i++)
     for(j=o;j<N;j++)
          a[i][j] = rand()  %  10 ;
          b[i][j] = rand()  %  10 ;
```

Then, the interval which is equal to division of N by Total Number Of Processes and remainder which is equal to mode of N on Total Number Of Processes is calculated. While matrix A is sent to processes interval by interval, matrix B is broadcasted to all processes. To send matrix A interval by interval to the processes, synchronous message passing is used. As a result, sender and receiver wait for each other to transfer intervals of matrix A.

MPI_Bcast(B,N*N,MPI_DOUBLE,0,MPI_COMM_WORLD);
Matrix B and intervals of matrix A are received by worker processes. **MPI_Bcast(B,N*N,MPI_DOUBLE,0,MPI_COMM_WORLD);**

MPI_Recv(A+(rank*interval),interval*N,MPI_DOUBLE,0,rank, MPI_COMM_WORLD,&status);

Multiplication of the first interval and remainder part of matrix A with matrix B is calculated by root process. Multiplication of other intervals with matrix B are calculated by worker processes.

MPI_Irecv(AB+(s*interval),interval*N,MPI_DOUBLE,s,s,MPI_COMM_WORL D,ireq+s);

Calculation time is found by using MPI_Wtime()function. After randomly creating A and B matrices, time is recorded.

```
time1 = MPI_Wtime();
```

After root gets the results from worker processes, time is recorded for the second time.

```
time2 = MPI_Wtime();
```

To get calculation time, these two times are subtracted from each other.

```
time = time2 - time1;
```

Chapter 4

Technology Stack

SOFTWARE SPECIFICATIONS:

Language Used : C

Operating System : Ubuntu

Communication Model Used : MPI_COMM_WORLD

HARDWARE SPECIFICATIONS:

Processor : Intel core I7

RAM : 16 GB RAM

Graphics : 1050 Ti Graphics

No of Cores: 4

Chapter 5

Results

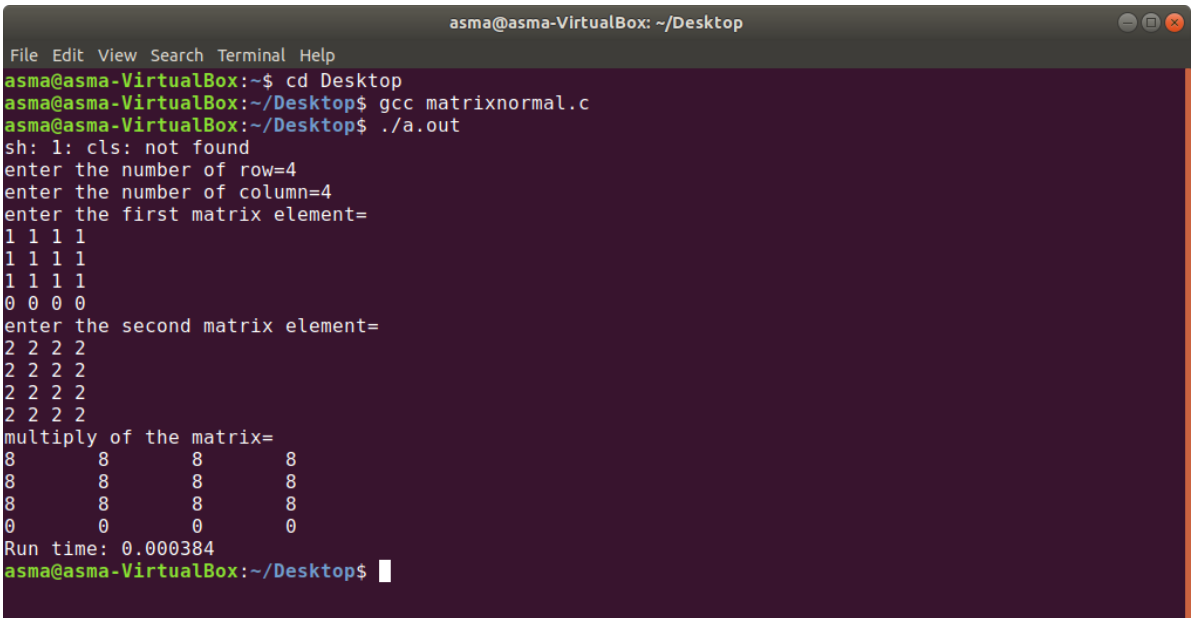

Fig2 : Matrix Multiplication with MPI

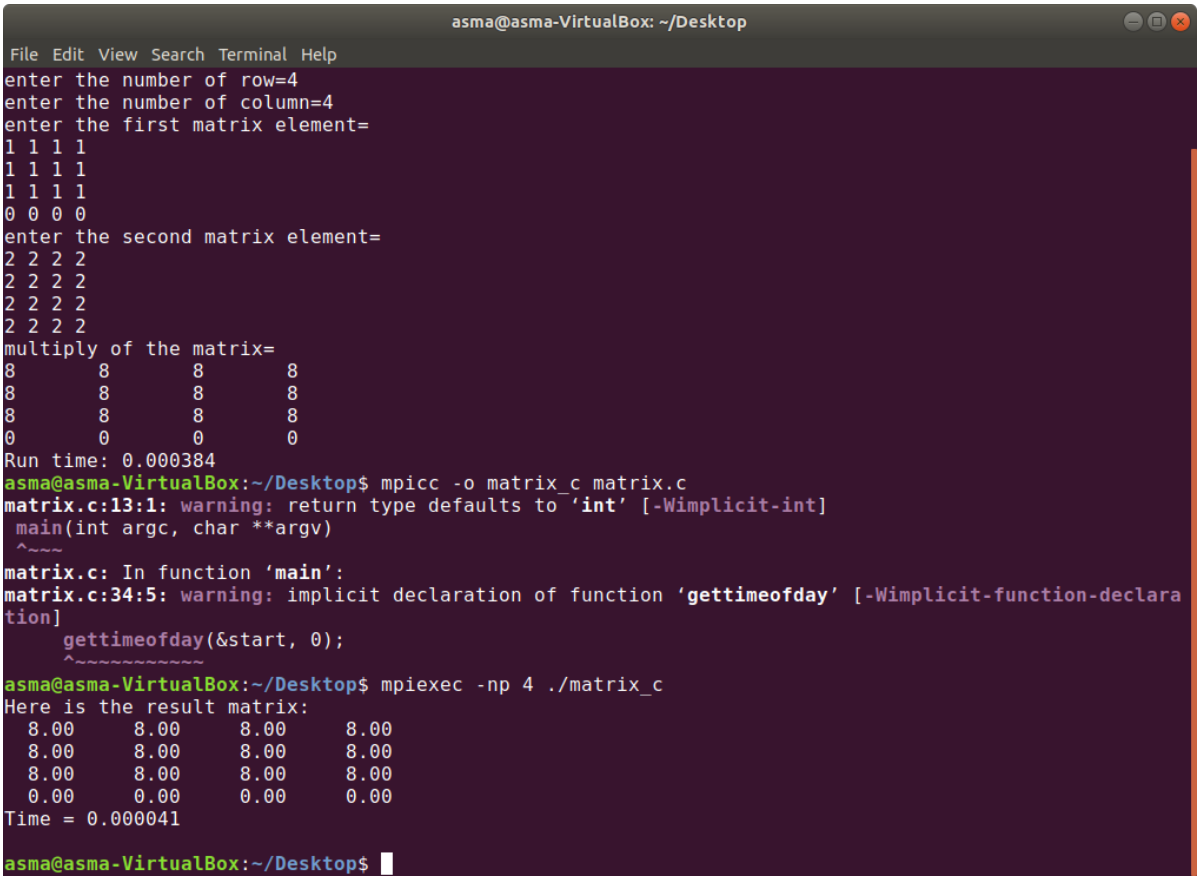

Fig3 : Matrix Multiplication without MPI

Chapter 6

Conclusion

By performing Matrix Multiplication by using Message Passing Interface Model we have seen a significant drop in the amount of execution time that is taken by the normal systems to execute the same code. We have compared both the mechanisms of Matrix - Matrix multiplication i.e. with and without using MPI by displaying the execution times.

By using the concept of dividing the execution procedure routine to various threads by using various high performance computing models we can achieve a higher sense of parallelism and improved performance. Using a suitable communication mode like MPI to distribute the task by using send receive primitives has also drastically reduced the cost that is required by the processes to successfully synchronize with each other.

References

- https://www.cdac.in/index.aspx?id=ev_hpc_hypack_matrix-vector-matrix-comp-mpi
- https://www.sciencedirect.com/topics/computer-science/matrix-multiplication
- http://icl.cs.utk.edu/classes/cosc462/2016/pdf/homework3.pdf
- https://www.mathsisfun.com/algebra/matrix-multiplying.html
- https://mpitutorial.com/tutorials/mpi-introduction
- https://www.isites.info/pastconferences/isites2014/isites2014/papers/B6-ISITES2014ID47.pdf

Source Code :

Code with MPI Implementation

Matrix.c

```
#include <time.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdlib.h>
#include <stdio.h>
#include <mpi.h>
#define N               4        /* number of rows and columns in matrix */

MPI_Status status;

double a[N][N],b[N][N],c[N][N];

main(int argc, char **argv)
{
  int numtasks,taskid,numworkers,source,dest,rows,offset,i,j,k;

  struct timeval start, stop;

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &taskid);
  MPI_Comm_size(MPI_COMM_WORLD, &numtasks);

  numworkers = numtasks-1;

  /*---------------------------- master -----------------------------*/
  if (taskid == 0) {
    for (i=0; i<N; i++) {
      for (j=0; j<N; j++) {
        a[i][j]= 1.0;
        b[i][j]= 2.0;
      }
    }

    gettimeofday(&start, 0);

    /* send matrix data to the worker tasks */
    rows = N/numworkers;
    offset = 0;

    for (dest=1; dest<=numworkers; dest++)
    {
      MPI_Send(&offset, 1, MPI_INT, dest, 1, MPI_COMM_WORLD);
      MPI_Send(&rows, 1, MPI_INT, dest, 1, MPI_COMM_WORLD);
      MPI_Send(&a[offset][0], rows*N, MPI_DOUBLE,dest,1, MPI_COMM_WORLD);
      MPI_Send(&b, N*N, MPI_DOUBLE, dest, 1, MPI_COMM_WORLD);
      offset = offset + rows;
    }

    /* wait for results from all worker tasks */
    for (i=1; i<=numworkers; i++)
    {
```

```
      source = i;
      MPI_Recv(&offset, 1, MPI_INT, source, 2, MPI_COMM_WORLD, &status);
      MPI_Recv(&rows, 1, MPI_INT, source, 2, MPI_COMM_WORLD, &status);
      MPI_Recv(&c[offset][0], rows*N, MPI_DOUBLE, source, 2, MPI_COMM_WORLD,
&status);
    }

    gettimeofday(&stop, 0);

    printf("Here is the result matrix:\n");
    for (i=0; i<N; i++) {
      for (j=0; j<N; j++)
        printf("%6.2f   ", c[i][j]);
      printf ("\n");
    }

    fprintf(stdout,"Time = %.6f\n\n",
         (stop.tv_sec+stop.tv_usec*1e-6)-(start.tv_sec+start.tv_usec*1e-6));

  }

  /*---------------------------- worker----------------------------*/
  if (taskid > 0) {
    source = 0;
    MPI_Recv(&offset, 1, MPI_INT, source, 1, MPI_COMM_WORLD, &status);
    MPI_Recv(&rows, 1, MPI_INT, source, 1, MPI_COMM_WORLD, &status);
    MPI_Recv(&a, rows*N, MPI_DOUBLE, source, 1, MPI_COMM_WORLD, &status);
    MPI_Recv(&b, N*N, MPI_DOUBLE, source, 1, MPI_COMM_WORLD, &status);

    /* Matrix multiplication */
    for (k=0; k<N; k++)
      for (i=0; i<rows; i++) {
        c[i][k] = 0.0;
        for (j=0; j<N; j++)
          c[i][k] = c[i][k] + a[i][j] * b[j][k];
      }


    MPI_Send(&offset, 1, MPI_INT, 0, 2, MPI_COMM_WORLD);
    MPI_Send(&rows, 1, MPI_INT, 0, 2, MPI_COMM_WORLD);
    MPI_Send(&c, rows*N, MPI_DOUBLE, 0, 2, MPI_COMM_WORLD);
  }

  MPI_Finalize();
}
```

Code without MPI Implementation

MatrixNormal.c

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include <math.h>
#include <assert.h>
 int main(){
```

```
   int a[10][10],b[10][10],mul[10][10],r,c,i,j,k,time_spent;
   clock_t start, stop;
    double t = 0.0;

    /* Start timer */
start = clock();
assert(start != -1);

   system("cls");
   printf("enter the number of row=");
   scanf("%d",&r);
   printf("enter the number of column=");
   scanf("%d",&c);
   printf("enter the first matrix element=\n");
   for(i=0;i<r;i++)
   {
   for(j=0;j<c;j++)
   {
   scanf("%d",&a[i][j]);
   }
   }
   printf("enter the second matrix element=\n");
   for(i=0;i<r;i++)
   {
   for(j=0;j<c;j++)
   {
   scanf("%d",&b[i][j]);
   }
   }

   printf("multiply of the matrix=\n");
   for(i=0;i<r;i++)
   {
   for(j=0;j<c;j++)
   {
   mul[i][j]=0;
   for(k=0;k<c;k++)
   {
   mul[i][j]+=a[i][k]*b[k][j];
   }
   }
   }
    clock_t end = clock();
   //for printing result
   for(i=0;i<r;i++)
   {
   for(j=0;j<c;j++)
   {
   printf("%d\t",mul[i][j]);
   }
   printf("\n");
   }

    stop = clock();
t = (double) (stop-start)/CLOCKS_PER_SEC;
printf("Run time: %f\n", t);
   return 0;
   }
```