Laboratory 7

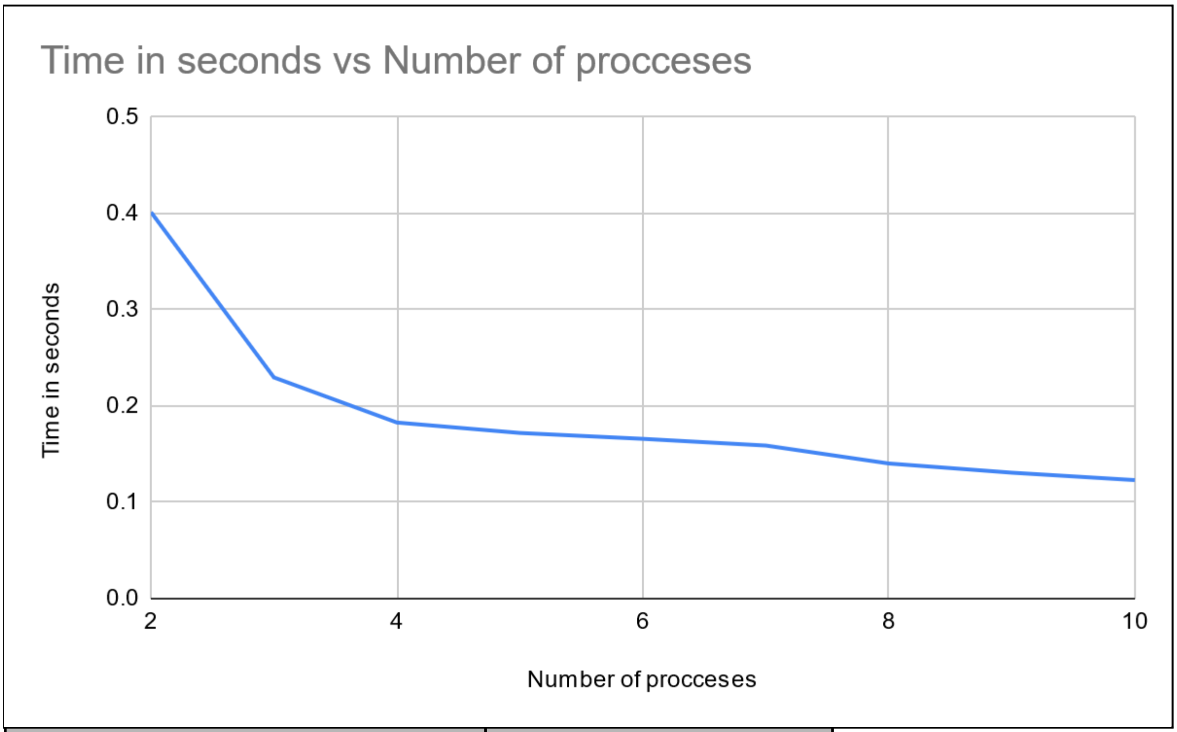
- **Code of the matrix-multiplication program.**(Modified or added code marked as _

```
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N 500               /* number of rows in matrixes */

int test(double Matrix1[N][N], double Matrix2[N][N], double Matrix3[N][N])
{
          int i,j,k,chk=0;
          double tmp,**Mtest;

          for (i = 0; i < N; i++)
          {
          for (j = 0; j < N; j++)
          {
     tmp = 0;
     for (k = 0; k < N; k++)
     {
          tmp += Matrix1[i][k] * Matrix2[k][j];
          }
     if(Matrix3[i][j]!=tmp)
     {
           printf("Error in element %d,%d!\n",i,j);
           chk=1;
          }
          }
          }
          return chk;
}

int main (int argc, char *argv[])
{
int    size,                /* number of processes */
   rank,          /* a process identifier */
   numworkers,          /* number of worker processes */
   source,              /* process id of message source */
   dest,                /* process id of message destination */
   rows,                /* rows of matrix A sent to each worker */
   averow, extra, offset, /* used to determine rows sent to each worker */
   i, j, k, rc;   /* misc */
double    a[N][N],      /* matrix A to be multiplied */
```

```
    b[N][N],        /* matrix B to be multiplied */
    c[N][N];        /* result matrix C */
MPI_Status status;

MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
MPI_Comm_size(MPI_COMM_WORLD,&size);
if (size < 2 ) {
  printf("Need at least two MPI processes. Quitting...\n");
  MPI_Abort(MPI_COMM_WORLD, rc);
  exit(1);
  }
numworkers = size-1;

/**************************** master task ************************************/
   if (rank == 0)
   {
            printf("Program has started with %d processes.\n",size);
            printf("Initializing arrays...\n");
            srand(time(NULL));
            for(i=0;i<N;i++){
            for(j=0;j<N;j++)
            {
            a[i][j]=rand()%1001/1000.*100;
            b[i][j]=rand()%1001/1000.*100;
            }
            }
            double start= MPI_Wtime();
            /* Send matrix data to the worker tasks */
            averow = N/numworkers;
            extra = N%numworkers;
            offset = 0;
            for (dest=1; dest<=numworkers; dest++)
            {
            rows = (dest <= extra) ? averow+1 : averow;    //to compute rest of the division
of matrix size by the numworkers
            printf("Sending %d rows to process %d offset=%d\n",rows,dest,offset);
            //MPI procedure for sending the offset
            MPI_Send(&offset,1,MPI_INT,dest,0,MPI_COMM_WORLD);
            //MPI procedure for sending the number of rows
            MPI_Send(&rows,1,MPI_INT,dest,0,MPI_COMM_WORLD);
            //MPI procedure for sending rows from offset in "a" array
            MPI_Send(&a[offset][0],rows*N,MPI_DOUBLE,dest,0,MPI_COMM_WORLD);
            offset = offset + rows;
            //MPI(group) procedure for sending the whole "b" array to all workers
            MPI_Send(&b, N*N, MPI_DOUBLE, dest, 0, MPI_COMM_WORLD);
            }
            //MPI_Bcast(&b,N*N, MPI_DOUBLE,0,MPI_COMM_WORLD); //another option

            /* Receive results from worker tasks */

            for (i=1; i<=numworkers; i++)
```

```
        {
        source = i;
         //MPI procedure for receiving an offset for the result array
        MPI_Recv(&offset,1,MPI_INT,source,1,MPI_COMM_WORLD,&status);
        //MPI procedure for receiving a number of rows for the result array
        MPI_Recv(&rows,1,MPI_INT,source,1,MPI_COMM_WORLD,&status);
        //MPI procedure for receiving rows from offset in "c" array

MPI_Recv(&c[offset][0],rows*N,MPI_DOUBLE,source,1,MPI_COMM_WORLD,&status);
        printf("Received results from process %d\n",source);
        }
        double end= MPI_Wtime();
        printf("The time for multiplication with %d procceses is: %f\n",size,end-start);
        rc=test(a,b,c);
        if(rc==1)
   printf("Error in matrix multiplication!\n");
  }

/**************************** worker task ************************************/
  if (rank > 0)
  {
        //MPI procedure for receiving an offset for the "a" array
        MPI_Recv(&offset,1,MPI_INT,0,0,MPI_COMM_WORLD,&status);
        //MPI procedure for receiving a number of rows for the "a" array
        MPI_Recv(&rows,1,MPI_INT,0,0,MPI_COMM_WORLD,&status);
        // MPI procedure for receiving rows from offset in "a" array
        MPI_Recv(&a,rows*N,MPI_DOUBLE,0,0,MPI_COMM_WORLD,&status);

        //MPI(group) procedure for receiving the whole "b" array
        // no needed if it was a broadcast
        MPI_Recv(&b,N*N,MPI_DOUBLE,0,0,MPI_COMM_WORLD,&status);

        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 procedure for sending an offset for the "c" array
        MPI_Send(&offset,1,MPI_INT,0,1,MPI_COMM_WORLD);
        //MPI procedure for sending a number of rows for the "c" array
        MPI_Send(&rows,1,MPI_INT,0,1,MPI_COMM_WORLD);
        //MPI procedure for sending computed rows of the "c" array
        MPI_Send(&c,rows*N,MPI_DOUBLE,0,1,MPI_COMM_WORLD);
  }
  MPI_Finalize();
}
```

- **Test results**



Number of processes	Time in seconds
2	0.401417
3	0.229663
4	0.182828
5	0.171995
6	0.165894
7	0.158912
8	0.140279
9	0.13072
10	0.123047

Interesting details: 2 processors with 6 cores each one (12 cores in total) were setted to a VMWARE of Ubuntu 20.04 LTS to reach 12 slots available to set with the np flag of mpirun command. For this experiment from 2 to 10 processes were set to see the change in seconds.