Laborator 10
MPI – Înmulțirea a două matrice

Să se implementeze un algoritm paralel MPI care să realizeze înmulțirea a două matrice pătratice NxN. Să se afișeze timpul de execuție. Dacă N<10 să se afișeze și matricea rezultat. Firul de execuție cu ID-ul 0 (MASTER) se va ocupa cu generarea matricelor, partiționarea și trimiterea datelor către celelalte fire de execuție precum și cu colectarea rezultatelor parțiale de la acestea.

Pentru rularea programului nu uitați să efectuați setările necesare pentru mediul MPI: Project -> properties -> c/c++-> All options -> Additional Include Directories : $(MSMPI_INC); $(MSMPI_INC)\x86 ; Project -> Properties -> Linker -> All options -> Additional Dependencies : msmpi.lib; Project -> Properties -> Linker -> All options -> Additional Library Directories : $(MSMPI_LIB32)

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

#define N 200
#define MASTER 0
#define FROM_MASTER 1
#define FROM_WORKER 2

int main(int argc, char *argv[])
{
      int   numtasks,
            taskid,
            numworkers,
            source,
            dest,
            mtype,
            rows,
            averow, extra, offset,
            i, j, k, rc;
      double     a[N][N], b[N][N], c[N][N];
      clock_t start, stop;
      double durata;
      MPI_Status status;
```

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

if (taskid == MASTER)
{
      start = clock();
      printf("mpi_mm has started with %d tasks.\n", numtasks);
      printf("Initializing arrays...\n");
      for (i = 0; i<N; i++)
            for (j = 0; j<N; j++)
                  a[i][j] = i + j;
      for (i = 0; i<N; i++)
            for (j = 0; j<N; j++)
                  b[i][j] = i*j;
      averow = N / numworkers;
      extra = N%numworkers;
      offset = 0;
      mtype = FROM_MASTER;
      for (dest = 1; dest <= numworkers; dest++)
      {
            rows = (dest <= extra) ? averow + 1 : averow;
            printf("Sending %d rows to task %d offset=%d\n", rows,
                  dest, offset);
            MPI_Send(&offset, 1, MPI_INT, dest, mtype,
                  MPI_COMM_WORLD);
            MPI_Send(&rows, 1, MPI_INT, dest, mtype,
                  MPI_COMM_WORLD);
            MPI_Send(&a[offset][0], rows*N, MPI_DOUBLE, dest,
                  mtype, MPI_COMM_WORLD);
            MPI_Send(&b, N*N, MPI_DOUBLE, dest, mtype,
                  MPI_COMM_WORLD);
            offset = offset + rows;
      }
      mtype = FROM_WORKER;
      for (i = 1; i <= numworkers; i++)
      {
            source = i;
            MPI_Recv(&offset, 1, MPI_INT, source, mtype,
```

```
                    MPI_COMM_WORLD, &status);
               MPI_Recv(&rows, 1, MPI_INT, source, mtype,
                    MPI_COMM_WORLD, &status);
               MPI_Recv(&c[offset][0], rows*N, MPI_DOUBLE, source,
                    mtype, MPI_COMM_WORLD, &status);
               printf("Received results from task %d\n", source);
          }
          if (N<10) {
               printf("****************************************\n");
               printf("Result Matrix:\n");
               for (i = 0; i<N; i++)
               {
                    printf("\n");
                    for (j = 0; j<N; j++)
                         printf("%6.2f   ", c[i][j]);
               }
               printf("\n****************************************\n");
          }
          printf("Done.\n");
          stop = clock();
          durata = (double)(stop - start) / CLOCKS_PER_SEC;
          printf("DURATA: %2.10f seconds\n", durata);
     }
     if (taskid > MASTER)
     {
          mtype = FROM_MASTER;
          MPI_Recv(&offset, 1, MPI_INT, MASTER, mtype,
                         MPI_COMM_WORLD, &status);
          MPI_Recv(&rows, 1, MPI_INT, MASTER, mtype, MPI_COMM_WORLD,
                         &status);
          MPI_Recv(&a, rows*N, MPI_DOUBLE, MASTER, mtype,
                         MPI_COMM_WORLD, &status);
          MPI_Recv(&b, N*N, MPI_DOUBLE, MASTER, mtype,
                         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];
               }
          mtype = FROM_WORKER;
          MPI_Send(&offset, 1, MPI_INT, MASTER, mtype,
                         MPI_COMM_WORLD);
          MPI_Send(&rows, 1, MPI_INT, MASTER, mtype, MPI_COMM_WORLD);
```

```
            MPI_Send(&c, rows*N, MPI_DOUBLE, MASTER, mtype,
                             MPI_COMM_WORLD);
      }
      MPI_Finalize();
    return 0;
}
```

Sarcini de lucru:

Explicați cum se realizează partiționarea matricelor de intrare. Modificați programul propus pentru o eficientizare a distribuției matricei b.

Modificați programul propus a. î. partiționarea și distribuția să se realizeze la nivel de bloc matrice de intrare.