

**UNIVERSITETI POLITEKNIK I TIRANËS**

**FAKULTETI I TEKNOLOGJISË DHE INFORMACIONIT**

**DEPARTAMENTI I INXHINIERISË INFORMATIKE**

**Punë Laboratori nr. 2**

**Tema**: Përdorimi I funksioneve të librarisë mpi.h

**Lënda:** Sisteme Të Shpërndara

**Grupi:** III-B

**Punoi:**  **Pranoi**:

Piro Gjikdhima MSc.Megi Tartari

**Ushtrimi 1**

Implementoni ne gjuhen C nje program qe realizon shumezimin e 2 matricave A[n][n] dhe B[n][n], dhe rezultatin e ruan ne matricen C[n][n]. Perdorni funksionet e MPI per te realizuar ekzekutimin me procese paralele.

*KODI*

#include <stdio.h>

#include <stdlib.h>

#include <mpi.h>

void populateMatrix(int rows, int cols, int matrix[rows][cols]);

void multiplyMatrix(int rows\_A, int cols\_A, int rows\_B, int cols\_B, int A[rows\_A][cols\_A], int B[rows\_B][cols\_B], int C[cols\_A][cols\_B]);

int main(int argc, char \*argv*[]*) {

srand(102021);

int rank, size, rows\_A, cols\_A, rows\_B, cols\_B;

rows\_A = atoi(argv[1]);

cols\_A = atoi(argv[2]);

rows\_B = atoi(argv[3]);

cols\_B = atoi(argv[4]);

MPI\_Init(&argc, &argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

if (argc != 5) {

if (rank == 0) {

printf("\033[1;34mPerdorimi: %s <rows\_A> <cols\_A> <rows\_B> <cols\_B>\033[0m\n", argv[0]);

}

MPI\_Abort(MPI\_COMM\_WORLD, 1);

}

if (cols\_A != rows\_B) {

printf("\033[1;31mNuk mund të bëhet shumëzimi i matricave! Kolonat e A (%d) duhet të përputhen me rreshtat e B (%d).\033[0m\n", cols\_A, rows\_B);

MPI\_Abort(MPI\_COMM\_WORLD, 1);

}

MPI\_Bcast(&rows\_A, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

MPI\_Bcast(&cols\_A, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

MPI\_Bcast(&rows\_B, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

MPI\_Bcast(&cols\_B, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

if (rows\_A % size != 0) {

if (rank == 0) {

printf("\033[1;31mNumri i rreshtave në Matricën A duhet të jetë i plotpjestueshem nga numri i proceseve.\033[0m\n");

}

MPI\_Abort(MPI\_COMM\_WORLD, 1);

}

int rows\_per\_process = rows\_A / size;

int A[rows\_A][cols\_A], B[rows\_B][cols\_B], C[rows\_A][cols\_B];

int local\_A[rows\_per\_process][cols\_A];

int local\_C[rows\_per\_process][cols\_B];

if (rank == 0) {

populateMatrix(rows\_A, cols\_A, A);

populateMatrix(rows\_B, cols\_B, B);

}

MPI\_Bcast(B, rows\_B \* cols\_B, MPI\_INT, 0, MPI\_COMM\_WORLD);

MPI\_Scatter(A, rows\_per\_process \* cols\_A, MPI\_INT, local\_A, rows\_per\_process \* cols\_A, MPI\_INT, 0, MPI\_COMM\_WORLD);

multiplyMatrix(rows\_per\_process, cols\_A, rows\_B, cols\_B, local\_A, B, local\_C);

MPI\_Gather(local\_C, rows\_per\_process \* cols\_B, MPI\_INT, C, rows\_per\_process \* cols\_B, MPI\_INT, 0, MPI\_COMM\_WORLD);

if (rank == 0) {

printf("Matricë rezultati C:\n");

for (int i = 0; i < rows\_A; i++) {

for (int j = 0; j < cols\_B; j++) {

printf("%d ", C[i][j]);

}

printf("\n");

}

}

MPI\_Finalize();

return 0;

}

void populateMatrix(int rows, int cols, int matrix[rows][cols]) {

for (int i = 0; i < rows; i++) {

for (int j = 0; j < cols; j++) {

matrix[i][j] = rand() % 10;

}

}

}

void multiplyMatrix(int rows\_A, int cols\_A, int rows\_B, int cols\_B, int A[rows\_A][cols\_A], int B[rows\_B][cols\_B], int C[cols\_A][cols\_B]){

for (int i = 0; i < rows\_A; i++) {

for (int j = 0; j < cols\_B; j++) {

C[i][j] = 0;

for (int k = 0; k < cols\_A; k++) {

C[i][j] += A[i][k] \* B[k][j];

}

}

}

}

*Rezultate*

Kodi i mësipërm përdor funksionet e librarisë mpi.h per te bere paralelizimin e kërkesës se dhene. Llogjika është e tille:

1. Fillimisht marrim prej konsoles numrin e rreshtave dhe shtyllave per secilën matrice.
2. Para se të vazhdojmë me paralelizimin, programi kontrollon që përputhshmëria e dimensioneve të matricave të jetë e duhur për shumëzim.Pra a janë kolonat e A = me rreshtat e B.
3. Pastaj kontrollohet nëse numri i rreshtave te matricës A është i plotpjesëtueshëm me numrin e proceseve.Nese jo , do te thotë nuk mundemi ti ndajmë ne menyre te barabarte midis proceseve rreshtat e A dhe kemi gabim. Dalim prej cdo procesi.
4. Nëse nuk ka problem, bëjmë broadcast cdo rresht e shtylle tek cdo proces, gjithashtu dërgojmë dhe matricën B.Nderkohe kemi krijuar matrica lokale A dhe C qe do te perdoren si buffer prej cdo procesi.
5. Programi përdor MPI\_Scatter për të shpërndarë rreshtat e Matricës A midis proceseve.
6. Çdo proces kryen shumëzimin lokal të pjesës së tij të matricës A me matricën B.
7. Pasi secili proces ka përfunduar shumëzimin, MPI\_Gather përdoret për të mbledhur rezultatet nga të gjitha proceset dhe për t'i bashkuar ato në Matricën C të procesit kryesor.
8. Në fund, procesi kryesor (rank 0) printon matricën C, që është rezultati i shumëzimit të matricave A dhe B.

Ky program përdor MPI për të realizuar paralelizimin e shumëzimit të matricave. Ai ndan punën midis proceseve të ndryshëm për të realizuar shumëzimin dhe për të optimizuar kohën e ekzekutimit. Pas përfundimit të operacionit, rezultati i shumëzimit të matricave është mbledhur dhe shfaqur nga procesi kryesor.

A screenshot of a computer program

AI-generated content may be incorrect.

**Ushtrimi 2**

Implementoni ne gjuhen C nje program qe gjen shumatoren e te gjithe elementeve te nje matrice A. Perdorni funksionet e MPI per te realizuar ekzekutimin me procese paralele.

*Kodi*

#include <stdio.h>

#include <stdlib.h>

#include <mpi.h>

void populateMatrix(int rows, int cols, int matrix[rows][cols]);

int main(int argc, char \*argv*[]*) {

srand(102021);

int rank, size, rows, cols;

rows = atoi(argv[1]);

cols = atoi(argv[2]);

MPI\_Init(&argc, &argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

if (argc != 3) {

if (rank == 0) {

printf("\033[1;34mPerdorimi: %s <rows> <cols>\033[0m\n", argv[0]);

}

MPI\_Abort(MPI\_COMM\_WORLD, 1);

}

MPI\_Bcast(&rows, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

MPI\_Bcast(&cols, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

if (rows % size != 0 && cols % size != 0) {

if (rank == 0) {

printf("\033[1;31mNumri i rreshtave: (%d) ose i shtyllave: (%d) duhet plotpjestueshem nga numri i proceseve: (%d).\033[0m\n",rows, cols, size);

}

MPI\_Abort(MPI\_COMM\_WORLD, 1);

}

else if (cols % size == 0)

{

int temp = rows;

rows = cols;

cols = temp;

}

int rows\_per\_process = rows / size;

int A[rows][cols];

int local\_A[rows\_per\_process][cols];

if (rank == 0) {

populateMatrix(rows, cols, A);

}

MPI\_Scatter(A, rows\_per\_process \* cols, MPI\_INT, local\_A, rows\_per\_process \* cols, MPI\_INT, 0, MPI\_COMM\_WORLD);

long local\_sum = 0;

for (int i = 0; i < rows\_per\_process; i++) {

for (int j = 0; j < cols; j++) {

local\_sum += local\_A[i][j];

}

}

printf("Procesi %d llogariti shumen: %ld\n", rank, local\_sum);

long global\_sum = 0;

MPI\_Reduce(&local\_sum, &global\_sum, 1, MPI\_LONG, MPI\_SUM, 0, MPI\_COMM\_WORLD);

if (rank == 0) {

printf("Shuma e matrices: %ld\n", global\_sum);

}

MPI\_Finalize();

return 0;

}

void populateMatrix(int rows, int cols, int matrix[rows][cols]) {

for (int i = 0; i < rows; i++) {

for (int j = 0; j < cols; j++) {

matrix[i][j] = (rand() % 100) + 1;

}

}

}

*Konkluzione*

Kodi i mësipërm përdor funksionet e librarisë mpi.h per te bere paralelizimin e kërkesës se dhene. Llogjika është e tille:

1. **Inicializimi i MPI dhe marrja e të dhënave**:
   * Nga komanda argv, merren dimensionet e matrices (rows dhe cols).
   * Inicializohen proceset dhe identifikohen rank dhe size (numri i proceseve).
2. **Kontrolli i argumenteve**:
   * Nëse nuk jepen saktësisht 2 argumente (përveç emrit të programit), procesi kryesor (rank 0) printon një mesazh gabimi dhe abortohet ekzekutimi.
3. **Broadcast i dimensioneve**:
   * Procesi kryesor dërgon dimensionet rows dhe cols te të gjitha proceset përmes MPI\_Bcast.
4. **Kontrolli i plotpjesëtueshmërisë**:
   * Kontrollohet nëse rows ose cols janë të plotpjesëtueshëm me numrin e proceseve (size).
   * **Nëse cols % size == 0**, kodi ndërron vendet e rows dhe cols.
5. **Përgatitja e matricës dhe shpërndarja e rreshtave**:
   * Procesi kryesor gjeneron matricën A me vlera të rastësishme duke përdorur populateMatrix().
   * Përdoret MPI\_Scatter për të shpërndarë nga rows\_per\_process rreshta (ose kolona, në varësi të kushtit të mësipërm) në çdo proces.
6. **Llogaritja lokale e shumës**:
   * Çdo proces llogarit shumën e elementeve të pjesës së tij të matricës (local\_A).
   * Rezultati lokal ruhet në variablin local\_sum.
7. **Mbledhja e shumave globale**:
   * Përdoret MPI\_Reduce për të mbledhur të gjitha local\_sum në një global\_sum te procesi kryesor.
8. **Shfaqja e rezultatit**:
   * Procesi kryesor (rank 0) printon shumën totale të elementeve të matricës.

A screenshot of a computer screen

AI-generated content may be incorrect.