
Laborator 8

MPI – Operații de decompoziție și de reducere

Operațiile cele mai delicate în cazul implementării unui algoritm paralel utilizând modelul MPI sunt operațiile de decompoziție (spargerea setului de date și distribuirea acestuia) și de reducere (colectarea rezultatelor intermediare de la firele de execuție paralele). Pentru a înțelege mai bine vom utiliza următoarea problemă: să se calculeze suma elementelor unui vector prin implementarea unui algoritm paralel MPI; inițial vectorul se află în memoria task-ului 0 (master), task-ul master se va ocupa de partiționarea setului inițial de valori și de distribuția acestora; fiecare task va calcula suma elementelor pentru un interval și va returna către task-ul 0 suma parțială calculată.

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 : msmapi.lib; Project -> Properties -> Linker -> All options -> Additional Library Directories : \$(MSMPI_LIB32)

```
#include "stdafx.h"
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>
#define ARRAYSIZE 16000000
#define MASTER      0

float data[ARRAYSIZE];

int main(int argc, char *argv[])
{
    int  numtasks, taskid, rc, dest, offset, i, j, tag1,
        tag2, source, chunksize;
    float mysum, sum;
    float update(int myoffset, int chunk, int myid);
    MPI_Status status;

    /**** Initializare ****/
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
    MPI_Comm_rank(MPI_COMM_WORLD, &taskid);
    printf("MPI task %d has started...\n", taskid);
    chunksize = (ARRAYSIZE / numtasks);
```

```
tag2 = 1;
tag1 = 2;

/**** Master task *****/
if (taskid == MASTER) {

    /* Initializare vector */
    sum = 0;
    for (i = 0; i<ARRAYSIZE; i++) {
        data[i] = i * 1.0;
        sum = sum + data[i];
    }
    printf("Initialized array sum = %e\n", sum);

    /* Trimitere intervale de calcul */
    offset = chunksize;
    for (dest = 1; dest<numtasks; dest++) {
        MPI_Send(&offset, 1, MPI_INT, dest, tag1, MPI_COMM_WORLD);
        MPI_Send(&data[offset], chunksize, MPI_FLOAT, dest, tag2,
                MPI_COMM_WORLD);
        printf("Sent %d elements to task %d offset= %d\n", chunksize, dest,
                offset);
        offset = offset + chunksize;
    }

    /* Portiune calcul master */
    offset = 0;
    mysum = update(offset, chunksize, taskid);

    /* Primire intervale de calcul modificate */
    for (i = 1; i<numtasks; i++) {
        source = i;
        MPI_Recv(&offset, 1, MPI_INT, source, tag1, MPI_COMM_WORLD,
                &status);
        MPI_Recv(&data[offset], chunksize, MPI_FLOAT, source, tag2,
                MPI_COMM_WORLD, &status);
    }

    /* Operatie de reductie si afisarea noii sume */
    MPI_Reduce(&mysum, &sum, 1, MPI_FLOAT, MPI_SUM, MASTER,
                MPI_COMM_WORLD);
    printf("*** Final sum= %e ***\n", sum);

} /* Sfarsit sectiune master */
```

```

    /**** Non-master tasks *****/

    if (taskid > MASTER) {

        /* Primire intervale de calcul */
        source = MASTER;
        MPI_Recv(&offset, 1, MPI_INT, source, tag1, MPI_COMM_WORLD, &status);
        MPI_Recv(&data[offset], chunksize, MPI_FLOAT, source, tag2,
                 MPI_COMM_WORLD, &status);

        mysum = update(offset, chunksize, taskid);

        /* Trimiterea intervalelor modificate catre master */
        dest = MASTER;
        MPI_Send(&offset, 1, MPI_INT, dest, tag1, MPI_COMM_WORLD);
        MPI_Send(&data[offset], chunksize, MPI_FLOAT, MASTER, tag2,
                 MPI_COMM_WORLD);

        MPI_Reduce(&mysum, &sum, 1, MPI_FLOAT, MPI_SUM, MASTER,
                  MPI_COMM_WORLD);

    } /* Sfarsit sectiune non-master */

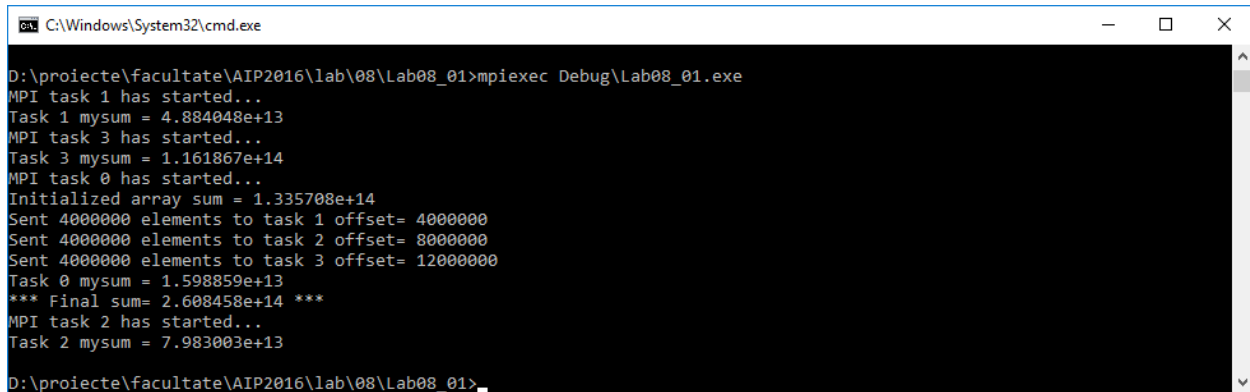
    MPI_Finalize();
    return 0;
}

float update(int myoffset, int chunk, int myid) {
    int i;
    float mysum;
    mysum = 0;
    for (i = myoffset; i < myoffset + chunk; i++) {
        data[i] = data[i] + i * 1.0;
        mysum = mysum + data[i];
    }
    printf("Task %d mysum = %e\n", myid, mysum);
    return(mysum);
}
```

Task-ul 0 (master) va inițializa vectorul (cu valori consecutive), va efectua suma inițială a elementelor și o va afișa. În funcție de numărul de task-uri va împărți vectorul în porțiuni egale și va trimite către celelalte task-uri elementele aferente păstrându-și un interval de calcul propriu. Toate task-urile vor modifica vectorul (vor dubla valoarea elementelor) și vor efectua o sumă parțială pentru elementele primite. Printr-o operație de reducere sumele parțiale vor fi colectate

de către task-ul 0 și noua sumă afișată (în felul acesta se va putea vedea că toate task-urile au prelucrat elementele vectorului și suma finală este suma tuturor sumelor parțiale calculate de task-uri – suma finală va fi dublul sumei inițiale).

Execuția (pe un microprocesor cu 4 nuclee) va avea următorul efect:



```
C:\Windows\System32\cmd.exe
D:\proiecte\facultate\AIP2016\lab\08\Lab08_01>mpiexec Debug\Lab08_01.exe
MPI task 1 has started...
Task 1 mysum = 4.884048e+13
MPI task 3 has started...
Task 3 mysum = 1.161867e+14
MPI task 0 has started...
Initialized array sum = 1.335708e+14
Sent 4000000 elements to task 1 offset= 4000000
Sent 4000000 elements to task 2 offset= 8000000
Sent 4000000 elements to task 3 offset= 12000000
Task 0 mysum = 1.598859e+13
*** Final sum= 2.608458e+14 ***
MPI task 2 has started...
Task 2 mysum = 7.983003e+13
D:\proiecte\facultate\AIP2016\lab\08\Lab08_01>
```

Operația de reducere se va implementa cu ajutorul funcției

```
MPI_Reduce(
    void* send_data,
    void* recv_data,
    int count,
    MPI_Datatype datatype,
    MPI_Op op,
    int root,
    MPI_Comm communicator)
```

ce permite efectuarea unei operații de colectare cumulativă pentru variabila sumă.

Plecând de la exemplul dat să se implementeze un algoritm paralel MPI ce evaluează valoarea unui polinom într-un punct. Similar cu exemplul dat, un singur fir de execuție va inițializa coeficienții polinomului, va efectua partiționarea și distribuția datelor către toate celelalte fire de execuție și va colecta la final, printr-o operație de reducere, rezultatele intermediare pentru a calcula valoarea finală a polinomului.